# Computing Extensions of Abstract Argumentation Frameworks by Enumerating Closed Sets

**Sergei Obiedkov**[1,2] , **Barış Sertkaya**[3]
[1]Knowledge-Based Systems Group, TU Dresden, Germany
[2]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI) Dresden, Germany
[3]Frankfurt University of Applied Sciences, Frankfurt, Germany
sergei.obiedkov@tu-dresden.de, sertkaya@fra-uas.de

**Abstract**

We present a new approach for computing complete, stable, and preferred extensions of abstract argumentation frameworks. Unlike existing approaches that reduce these problems to the propositional satisfiability problem and solve them with the help of SAT-solvers, our approach solves them directly by making use of the fact that the mentioned extensions are contained in certain closure systems. Our algorithms enumerate these closed sets and filter the searched extensions. Experimental results show that our approach outperforms the existing approaches for a large number of the test cases.

## 1 Introduction

Argumentation is a field of Artificial Intelligence (AI) dealing with formal representation of arguments and relations between arguments. Its aim is, among others, to provide methods for resolving conflicts collaboratively. The most prominent approach in this field, abstract argumentation, has attracted increasing attention in the AI and particularly in the Knowledge Representation communities since its introduction in (Dung 1995). In abstract argumentation, arguments are abstracted away from their actual contents and conflicts are modeled in the form of attacks between arguments. This abstraction allows an intuitive formalization using directed graphs, called *argumentation frameworks* (AF). The semantics of an AF is defined through sets of arguments called extensions. Several types of extensions have been proposed in the literature (Baroni, Caminada, and Giacomin 2011), giving rise to interesting computational problems such as, for instance, deciding whether a given argument appears in at least one extension of a certain type (credulous reasoning), or deciding whether it appears in all extensions of a certain type (skeptical reasoning), or enumerating all extensions of a certain type.

The computational complexity of these and related decision, enumeration, and counting problems has by now been well studied (Dunne and Wooldridge 2009; Kröll, Pichler, and Woltran 2017). On the practical level, there are highly optimized solvers that can handle large problem instances. In the bi-annually organized International Competition on Computational Models of Argumentation (ICCMA), solvers compete in different tracks on several different reasoning tasks. Typically, they encode these tasks as problems from other formalisms such as, for instance, the constraint satisfaction problem (CSP) or the satisfiability problem of propositional logic (SAT), and benefit from existing highly optimized solvers developed there. There are also algorithms specifically tailored for computational problems in AFs that directly solve these problems without reducing them to another formalism. A detailed survey of both types of approaches can be found in (Cerutti et al. 2017).

In this paper, we introduce a novel approach for directly computing extensions in argumentation frameworks. We show that the complete, preferred, and stable extensions of a given AF are contained in a closure system formed by what we call *pseudo-complete sets*, and we present an algorithm for computing extensions of a desired type by enumerating this closure system.

Experimental results obtained with our prototypical implementation CLAS[1] (Closure-based Argumentation Solver) show that, on most benchmarks from ICCMA'25[2], our algorithms outperform state-of-the-art solvers, while lagging behind them on some large benchmarks with multiple strongly connected components. To address this limitation, we combine our closure-based approach with an algorithm that computes preferred extensions by searching for complete extensions in individual components of a progressively decomposed framework.

The paper is organized as follows. In Section 2, we introduce basic notions of closure systems and AFs. In Section 3, we introduce the closure system of pseudo-complete sets and show that complete, preferred and stable extensions are pseudo-complete. In Section 4 we present an algorithm for computing complete and stable extensions based on this closure system. We further present several different optimization techniques for pruning the search space of our algorithm. In Section 5 we present an extension of our approach applicable to AFs with strongly connected components and focus on preferred extensions. In Sections 4.3 and 5.1 we evaluate the presented methods on benchmarks used in the ICCMA'25 competition and provide a comparison with existing AF solvers. In Section 6, we conclude with a summary and future work.

[1]https://github.com/sertkaya/afca/tree/CLAS
[2]https://argumentationcompetition.org/2025

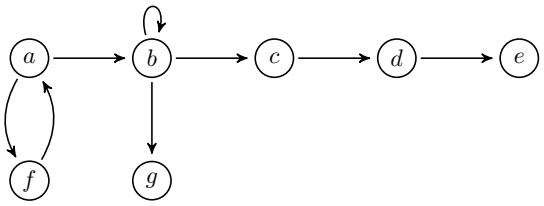


Figure 1: Example of an argumentation framework

## 2 Preliminaries

### 2.1 Abstract Argumentation Frameworks

We recall some basic notions from abstract argumentation as introduced in (Dung 1995). An *argumentation framework* (AF) is a directed graph $\mathcal{F} = (A, R)$, where $A$ is a finite set of *arguments* and $R \subseteq A \times A$ is called the *attack relation*. An edge $(a, b) \in R$ denotes that the argument $a$ *attacks* the argument $b$.

A set of arguments $S \subseteq A$ *attacks* $b \in A$ if there is $a \in S$ such that $(a, b) \in R$, and $b$ *attacks* $S$ if $(b, a) \in R$ for some $a \in S$. $R(S)$ denotes the arguments attacked by $S$, and $R^{-1}(S)$ denotes those attacking $S$. If $S = \{a\}$, we write $R(a)$ and $R^{-1}(a)$ instead of $R(\{a\})$ and $R^{-1}(\{a\})$.

For $(a, b) \in R$, we say that $S \subseteq A$ *defends* $b$ *against* $a$ if $S$ attacks $a$. If $S$ defends $b$ against all its attackers, i.e., if $R^{-1}(b) \subseteq R(S)$, we say that $S$ *defends* $b$. A set $S$ is called *self-defending* if it defends all its elements, i.e., $R^{-1}(S) \subseteq R(S)$.

**Example 1.** *Figure 1 illustrates an argumentation framework over the set of arguments $A = \{a, b, c, d, e, f, g\}$. For example, $a$ attacks $b$ and $f$; the set $\{b, c\}$ attacks $b$, $c$, $d$, and $g$; the argument $d$ attacks every set that includes $e$, such as $\{a, e\}$; and $\{a, e\}$ defends $c$ and $g$, since it attacks their only attacker, $b$.*

A set $S \subseteq A$ is said to be *conflict-free* if $S$ does not attack any of its elements. Self-defending conflict-free sets are called *admissible*. The semantics of an argumentation framework is defined through sets of arguments called *extensions*. Several different types of extensions have been considered in the literature (Dung 1995; Baroni, Caminada, and Giacomin 2011). We introduce only those that are relevant for this paper.

- An admissible set that contains every argument it defends is called a *complete extension*.
- An inclusion-wise maximal admissible set is called a *preferred extension*.
- An admissible set that attacks every argument outside itself is called a *stable extension*.

Since a stable extension $S$ attacks all other elements including all those that attack $S$, every stable extension is also a preferred extension. Since a preferred extension is a maximal admissible set, it contains every argument that it defends, i.e., it is a complete extension.

**Example 2.** *The complete extensions of the AF from Figure 1 are $\emptyset$, $\{f\}$, and $\{a, c, e, g\}$; its preferred extensions are $\{f\}$ and $\{a, c, e, g\}$; and its only stable extension is $\{a, c, e, g\}$.*

Several interesting decision, counting, and enumeration problems in abstract argumentation have been considered in the literature (Dunne and Wooldridge 2009; Baroni, Dunne, and Giacomin 2010; Kröll, Pichler, and Woltran 2017). Here we list three that are relevant for this paper. For an AF $\mathcal{F} = (A, R)$, an argument $a \in A$, and a semantic $\sigma$:

- Credulous acceptance (DC-$\sigma$): Is $a$ contained in at least one $\sigma$-extension of $\mathcal{F}$?
- Single extension (SE-$\sigma$): Find a $\sigma$-extension of $\mathcal{F}$ if there is one.
- Enumerate extensions (EE-$\sigma$): List all $\sigma$-extensions of $\mathcal{F}$.

It is known that these problems are intractable for many of the interesting semantics (Dunne and Wooldridge 2009; Kröll, Pichler, and Woltran 2017). Existing approaches typically work by reducing them to other formalisms such as constraint satisfaction, propositional logic, or answer-set programming, and benefit from highly optimized solvers developed for these formalisms. To name a few, $\mu$-toksia (Niskanen and Järvisalo 2020), FUDGE (Thimm, Cerutti, and Vallati 2021), and SCALOP (Lagniez, Lonca, and Mailly 2025) encode these problems as the Boolean satisfiability problem and make use of a SAT-solver; PYGLAF (Alviano 2021) reduces these problems to circumscription and employs an existing solver for circumscription; and *ConArg* (Bistarelli and Santini 2011) reduces them to constraint satisfaction and uses a CSP-solver.

### 2.2 Closure Systems

**Definition 1.** *A family $\mathcal{C}$ of subsets of a ground set $A$ is called a* closure system *if $\mathcal{C}$ contains $A$ and is closed under intersection. Elements of a closure system are called* closed sets. *A closure system gives rise to a* closure operator *that maps a subset of $A$ to its (unique) smallest closed superset.*

Closure systems play an important role in several fields, such as FCA (Ganter and Wille 2024) and Frequent Itemset Mining (Han et al. 2007). There are several algorithms for enumerating closed sets (Norris 1978; Bordat 1986; Kuznetsov 1993; Godin, Missaoui, and Alaoui 1995; Nourine and Raynaud 1999; Kuznetsov and Obiedkov 2002; Grahne and Zhu 2005; Uno, Kiyomi, and Arimura 2005; Ganter 2010; Outrata and Vychodil 2012), some of which do this with a polynomial delay (Johnson, Yannakakis, and Papadimitriou 1988).

## 3 Closure Systems and Extensions of AFs

The connection between closure systems and extensions of AFs has recently attracted attention in the FCA community. In (Obiedkov and Sertkaya 2023), we translated a given AF into a formal context, the basic data structure of FCA, and showed that the stable extensions of the AF are among the closed sets, called *concept intents*, of this formal context. We used lattice construction algorithms to enumerate the concept intents, pruning the current computation branch whenever a maximal conflict-free

intent was encountered. If such an intent is a stable extension, it is output. We adapted two lattice-construction algorithms for this purpose: NEXT CLOSURE (Ganter 1984; Ganter 2010), which enumerates all concept intents with polynomial delay, and the incremental algorithm proposed in (Norris 1978). The experimental results show that enumerating stable extensions using these algorithms is efficient for dense frameworks. However, sparse frameworks are tractable only when they are small.

In another work (Elaroussi, Nourine, and Radjef 2023), the authors translate a given AF into a set of implications (i.e., rules of the form $X \to Y$, where the inclusion of arguments in $X$ forces the inclusion of arguments in $Y$). They show that the sets closed under these implications are exactly the complements of the self-defending sets of the original AF, which can then be efficiently enumerated using, for example, NEXT CLOSURE. To compute admissible sets, they subsequently select the conflict-free sets among these complements and provide several optimizations for this approach.

Below we introduce a new closure system that forms the basis of our algorithm for computing complete extensions.

**Definition 2.** *A set $S \subseteq A$ is* semi-complete *if $S$ is conflict-free and contains every argument it defends:*

$$A \setminus (R(A) \cup R(S)) \subseteq S.$$

The only difference from complete extensions is that semi-complete sets are not required to be self-defending.

**Definition 3.** *A set $S \subseteq A$ is* quasi-complete *if $S$ is conflict-free and contains every argument that is attacked only by its attackers:*

$$A \setminus (R(A) \cup R^{-1}(S)) \subseteq S.$$

**Definition 4.** *A set $S \subseteq A$ is* pseudo-complete *if $S$ is both semi-complete and quasi-complete.*

**Example 3.** *Consider the framework shown in Figure 1. The set $\{c, e\}$ is semi-complete, as it does not defend any of the remaining arguments. However, it is not quasi-complete, since the argument $g$ is attacked only by $b$, which is also an attacker of $\{c, e\}$. In contrast, $\{c, g\}$ is quasi-complete: its only attacker is $b$, and there is no other argument attacked only by $b$. It is not semi-complete, since it defends $e$ but does not contain it. The set $\{c, e, g\}$ is both semi- and quasi-complete and is therefore pseudo-complete.*

**Proposition 1.** *Every complete (and therefore, every preferred and stable) extension is pseudo-complete.*

*Proof.* The fact that every complete extension is semi-complete is obvious from the definitions. To see that a complete extension $S$ must be quasi-complete, let $a \in A$ be an argument attacked only by $R^{-1}(S)$ and show $a \in S$. Since $S$ is self-defending, it attacks all arguments in $R^{-1}(S)$ and, by doing so, defends $a$. Therefore, $a \in S$, as required. □

**Proposition 2.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework. The semi-complete subsets of $A$, together with $A$, form a closure system.*

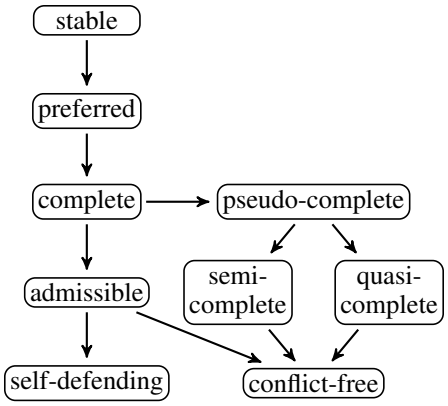


Figure 2: The relation between pseudo-complete sets and different types of extensions

*Proof.* It suffices to show that semi-complete sets are closed under intersection. Let $S$ and $T$ be semi-complete subsets of $A$. Since $S$ and $T$ are conflict-free, so is $S \cap T$. Let $a \in A$ be defended by $S \cap T$. Then $a$ is defended by both $S$ and $T$, and, as $S$ and $T$ are semi-complete, both sets contain $a$. It follows that $S \cap T$ contains every argument it defends and is therefore semi-complete. □

**Proposition 3.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework. The quasi-complete subsets of $A$, together with $A$, form a closure system.*

*Proof.* Similarly, we show that quasi-complete sets are closed under intersection. Let $S$ and be $T$ be quasi-complete subsets of $A$. Since $S$ and $T$ are conflict-free, so is $S \cap T$. Let $a \in A$ be attacked only by attackers of $S \cap T$. Then every attacker of $a$ attacks both $S$ and $T$. Since $S$ and $T$ are quasi-complete, $a$ is contained in both. It follows that $S \cap T$ contains every argument that is attacked only by attackers of $S \cap T$ and is therefore quasi-complete. □

**Corollary 1.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework. The pseudo-complete subsets of $A$, together with $A$, form a closure system.*

In a nutshell, our approach (detailed in the next section) is to enumerate pseudo-complete sets until a required extension appears in the output.

## 4 Computing Extensions by Enumerating Pseudo-complete Sets

We first focus on the credulous acceptance problem for the complete and stable extensions, namely DC-CO and DC-ST. Our approach is based on enumerating pseudo-complete sets until a required extension with the given argument appears.

Corollary 1 implies that there is a closure operator that maps a subset of $A$ to its smallest pseudo-complete superset (or to $A$ if no such superset exists). Let `pseudo-complete`$(\mathcal{F}, S, P)$ be a procedure for computing the closure of $S \subseteq A$ under this operator except that it returns $\perp$ when no pseudo-complete superset of $S$ exists or when the smallest such set contains an argument from $P$.

**Algorithm 1** Procedure $\sigma$-extension$(\mathcal{F}, S, P)$.

**Input**: An argumentation framework $\mathcal{F} = (A, R)$ and disjoint argument sets $S \subseteq A$ and $P$ such that $S$ is $P$-pseudocomplete in $\mathcal{F}$
**Output**: A $\sigma$-extension $\emptyset \neq X \supseteq S$ of $\mathcal{F}$ with $X \cap P = \emptyset$ if it exists; $\bot$ otherwise

```
1: if S ≠ ∅ is a σ-extension then
2:    return S
3: Form set C ⊆ A \ (R(S) ∪ R^{-1}(S) ∪ P) of candidates
4: for all d ∈ C do
5:    T := pseudo-complete(F, S ∪ {d}, P)
6:    if T ≠ ⊥ then
7:       X := σ-extension(F, T, P)
8:       if X ≠ ⊥ then
9:          return X
10:   P := P ∪ {d}.
11: return ⊥
```

There is no pseudo-complete superset of $S$ if, for example, $S$ defends one of its attackers. The role of $P$ will become clear later.

The procedure pseudo-complete$(\mathcal{F}, S, P)$ can be implemented in a straightforward way by applying the definitions of semi-complete and quasi-complete sets to the conflict-free set $S$ until a fixed point is reached:

- when there is $a \in A \setminus S$ such that $R^{-1}(a) \subseteq R(S)$, add $a$ to $S$ if $a \notin P$ and $a$ does not attack $S$ and return $\bot$ otherwise;
- when there is $a \in A \setminus S$ such that $R^{-1}(a) \subseteq R^{-1}(S)$, add $a$ to $S$ if $a \notin P$ and $a$ does not attack $S$ and return $\bot$ otherwise.

For computing a complete extension containing a given argument $c$, we start by checking if $c$ attacks itself. If so, then no such extension with $c$ exists. Otherwise, we form the smallest pseudo-complete set $S_0$ containing $c$:

$$S_0 := \texttt{pseudo-complete}(\mathcal{F}, \{c\}, \emptyset)$$

and then enumerate its pseudo-complete supersets until we find one that is also self-defending and thus a complete extension. For computing a stable extension containing $c$, the enumeration continues until a self-defending set that attacks everything outside itself is found. Enumeration is performed with the $\sigma$-extension procedure in Algorithm 1 where $\sigma$ stands for "complete" or "stable".

In addition to a set $S$ to be extended, this procedure takes as input a set $P$ of *prohibited* arguments, which are known to be unsuitable for extending $S$. Initially, $P$ contains arguments that are in conflict with $S_0$ and self-attacking arguments. Thus, the search for a $\sigma$-extension of $\mathcal{F} = (A, R)$ containing $S_0$ is initiated as

$$\sigma\texttt{-extension}(\mathcal{F}, S_0, R(S_0) \cup R^{-1}(S_0) \cup \{a \in A \mid aRa\}).$$

In the first line of Algorithm 1, we check whether $S$ is a $\sigma$-extension; if so, the algorithm terminates and returns $S$. Otherwise, in line 3, we construct a set of candidates $C$ consisting of arguments that can be added to $S$, i.e., arguments that are not prohibited and do not conflict with $S$. The exact construction of the candidate set depends on the extension type and is described in Section 4.1.

For each candidate argument $d \in C$, we tentatively add $d$ to $S$ and extend the result to its smallest pseudo-complete superset $T$ (line 5). If such a $T$ exists and consists entirely of arguments outside $P$, we attempt to expand it to a $\sigma$-extension by recursively calling the $\sigma$-extension procedure. If the call succeeds by producing a $\sigma$-extension, we return it (line 9). If, on the contrary, adding $d$ to $S$ does not allow us to obtain a $\sigma$-extension, we add $d$ to $P$ (line 10) so as to be able to prune dead-end branches while considering alternative candidates from $C$.

We note that the same approach can be used to solve the single-extension versions of these problems, SE-CO and SE-ST. The only modification is in the initialization of $S_0$, which, in this case, must be set to the minimal pseudo-complete set:

$$S_0 := \texttt{pseudo-complete}(\mathcal{F}, \emptyset.\emptyset).$$

Note that $S_0$ computed in this way is always a complete extension; so the SE-CO problem is trivial.

The same approach could, in principle, be used to solve SE-PR. However, verifying whether a given subset of arguments is a preferred extension, which is required in the first line of Algorithm 1, is a coNP-complete problem (Dunne and Wooldridge 2009). Fortunately, this verification is not necessary. A preferred extension is a maximal admissible set; thus, even if the current set $S$ is already a complete extension, it must still be extended with all suitable arguments. Accordingly, if $S$ is complete, the set of candidates in line 3 must include all arguments from $A \setminus (R(S) \cup R^{-1}(S) \cup P)$. If none of the candidates can be used to extend $S$, but $S$ is admissible, then it is a preferred extension and must be returned in line 11. Otherwise, if $S$ is not admissible, the current computation branch cannot yield a preferred extension, and $\bot$ is returned.

Also, Algorithm 1 can be easily extended to support enumeration of complete or stable extensions. For example, to enumerate stable extensions, it suffices not to terminate the **for all** loop after outputting the extension $X$ in line 9, and instead to let the algorithm explore alternative ways of forming stable extensions containing $S$. Because of the update to the set $P$ in line 10, none of these alternative extensions will contain the argument $d$. Thus, every stable extension will be output exactly once.

Our algorithm broadly follows the strategy of the Close by One algorithm for enumerating closed sets of a closure operator (Kuznetsov 1993). However, it differs in several important aspects (beyond the necessary adaptations to abstract argumentation). In particular, Close by One uses a fixed order on the elements of the ground set and attempts to extend the current set by each element in that order. In contrast, our algorithm dynamically selects an argument $d$ from a typically small candidate set $C$ and prunes the search branch if no candidate in $C$ is suitable. This significantly reduces the search space, especially in relatively sparse frameworks.

### 4.1 Candidate Selection

To construct the candidate set in line 3 of Algorithm 1, we employ the following strategy.

For complete extensions, we select an attacker $a$ of the current set $S$ that is not attacked by $S$ and attempt to defend $S$ against $a$ by including one of $a$'s attackers. We consider only those attackers of $a$ that are not in $P$, do not attack $S$, and are not attacked by $S$. If adding none of these attackers of $a$ leads to a complete extension, then $S$ cannot be defended against $a$. In that case, there is no need to consider other attackers of $S$, and the algorithm returns $\perp$ (line 11).

For stable extensions, the same sets of candidates can be used. However, in this case, not only we have to make $S$ self-defending, it should also attack all arguments in $A \setminus S$. This suggests that, for each $a \in A$, a stable extension must contain at least one argument from $\{a\} \cup R^{-1}(a)$. As long as $S$ is disjoint with such set for some $a$, that set, again restricted to arguments outside $P$ unattacked by and not attacking $S$, can be used as a set of candidates to extend $S$. In our experiments, we use this approach to form candidate sets in the case where $S$ is already complete but not yet stable.

Similarly, when computing preferred extensions, the algorithm does not terminate upon obtaining a complete extension $S$. As described in the previous section, in this case, the candidate set is given by $A \setminus (R(S) \cup R^{-1}(S) \cup P)$.

Among multiple sets of candidates, we select the smallest, since this typically makes it possible to detect unsuccessful branches earlier.

### 4.2 $P$-pseudo-complete Sets

Selecting appropriate candidates can help restrict the branching factor of the search tree. Next, we introduce an optimization that can make the search tree shallower.

In our experiments, search trees often contained long paths consisting of arguments with only one allowed attackers. A small part of such a search tree is depicted in Figure 3. It was generated while solving the DC-CO problem on the test framework named `crusti_g2io_125_0.5_31_17.af` in the benchmarks of the ICCMA 2023 competition. The task is to find a complete extension containing the argument 24659, which is depicted as the root node of the search tree.

Consider the node labeled by the argument 12292 in this tree. To make the corresponding set $S$ self-defending, we must include one of its three defenders: 12267, 12242, or 6046. We first add 12267 and find the smallest pseudo-complete superset of the resulting set. It has an attacker, the only available defender against which is 6148. We add 6148 to $S$, `pseudo-complete` the resulting set, then add 12227, `pseudo-complete` again, and detect a conflict: the set defends a conflicting argument. Therefore, this branch does not lead to a solution. We backtrack straight to the node labeled 12292 and attempt its next defender, 12242.

This required three calls to `pseudo-complete`. However, adding 12267 forces us to add 6148 and then 12227, since this is the only way to make the set self-defending. Therefore, instead of adding these arguments one by one, in separate calls to `pseudo-complete`, we can add them together in one call. Similar cases, where the current set has a single defender against one of its attackers, appear several times in this search tree.

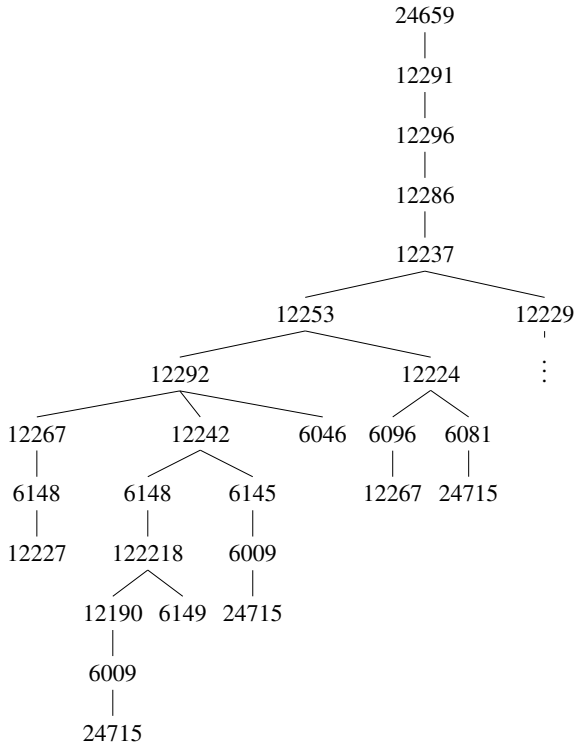


Figure 3: Part of the search tree generated before optimization

In order to avoid such cases, we modify our closure system of pseudo-complete sets from Corollary 1.

**Definition 5.** *Let $S$ and $P$ be disjoint subsets of $A$. We say that $S$ is* $P$-pseudo-complete *if it is pseudo-complete and every attacker of $S$ is either attacked by $S$ or has at least two attackers in $A \setminus (R(S) \cup R^{-1}(S) \cup P)$.*

The following proposition shows that the property required for our algorithm is still satisfied.

**Proposition 4.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework and $P \subseteq A$. The $P$-pseudo-complete subsets of $A$, together with $A$, form a closure system.*

*Proof.* Let $S, T \subseteq A$ be $P$-pseudo-complete. Consider $S \cap T$. Since $S$ and $T$ are conflict-free, so is $S \cap T$.

Let $a \in A$ be defended by $S \cap T$. Then $a$ is defended by both $S$ and $T$, and, as $S$ and $T$ are semi-complete, both sets contain $a$. It follows that $S \cap T$ contains every argument it defends and is therefore semi-complete.

Similarly, let $b$ be an argument attacked only by attackers of $S \cap T$. All these attackers attack both $S$ and $T$. It follows that $b$ belongs to both, as they are quasi-complete. Hence, $b \in S \cap T$, and $S \cap T$ is quasi-complete.

Thus, $S \cap T$ is pseudo-complete. Let $c$ be an attacker of $S \cap T$ not attacked by $S \cap T$. We have that $c$ attacks both $S$ and $T$. If both $S$ and $T$ attack $c$, then $c$ has at least two attackers in $S \cup T \subseteq A \setminus P$.

If $c$ is not attacked, say, by $S$, then, since $S$ is $P$-pseudo-complete, $c$ must have at least two attackers in $A \setminus P$. The statement follows. □

To compute the smallest $P$-pseudo-complete superset of $S$, we use the following rule (in addition to the two rules ensuring that the resulting set is pseudo-complete):

- when there is $a \in R^{-1}(S) \setminus R(S)$ such that the set $R^{-1}(a) \setminus (R(S) \cup R^{-1}(S) \cup P)$ consists of a single argument $b$, add $b$ to $S$.

**Example 4.** *Consider again the framework in Figure 1. To compute a complete extension containing $c$, we begin by computing the minimal $\emptyset$-pseudo-complete superset of $S := \{c\}$. To make $S$ semi-complete, we add the argument $e$, which is defended by $c$. To make it pseudo-complete, we add $g$, whose only attacker is among the attackers of $S$. The resulting set $S' = \{c, e, g\}$ is pseudo-complete. However, it is not $\emptyset$-pseudo-complete. Indeed, it has two attackers: $b$ and $d$. While the latter is attacked by $S'$, the former has a single attacker, $a$, which does not conflict with $S'$. Therefore, $a$ must be added to $S'$ to obtain a $\emptyset$-pseudo-complete set. In this case, the resulting $\emptyset$-pseudo-complete set $\{a, c, e, g\}$ happens to be a complete extension, so the computation terminates.*

Note that, as the computation in Algorithm 1 proceeds and $P$ grows, the set of $P$-pseudo-complete sets forms a progressively shrinking subset of the pseudo-complete sets, thereby reducing the search space.

### 4.3 Experimental Evaluation

We evaluated our approach on 322 benchmarks from the ICCMA'25 competition. For comparison, we selected three state-of-the-art solvers that ranked in the top three across the tracks of interest: DC-CO, DC-ST, SE-ST, and SE-PR. Each solver was run with a timeout of 1200 seconds per benchmark. Experiments were conducted on a Linux machine with a 2.9 GHz processor and 256 GB of RAM.

Table 1 reports, for each solver, the number of fastest runs, the number of failures (timeouts or, rarely, out-of-memory), and the PAR2 score (the standard ICCMA metric, where each failure counts as twice the timeout). CLAS is the fastest solver on the vast majority of benchmarks, but also exhibits substantially more failures, resulting in the highest PAR2 score. While this may suggest that CLAS performs better on easy instances and worse on harder ones, this interpretation is not entirely accurate: there are instances solved by CLAS within the timeout where all other solvers fail, indicating that CLAS can succeed on instances that are hard for the other solvers.

Figure 4 shows the wall-clock runtimes of all four solvers on a logarithmic scale for DC-CO. Each point corresponds to one solver on one framework. The $x$-coordinate is the average runtime across all solvers on that framework, while the $y$-coordinate is the runtime of the respective solver. Timeouts are counted as 1200 seconds.

For example, the rightmost blue point in Fig. 4 represents CLAS on a framework solved in 363 seconds. The other solvers timed out and therefore coincide at 1200 seconds. The shared $x$-coordinate reflects the average runtime and thus the instance difficulty (for this selection of solvers).

Eight benchmarks were not solved by any solver (rightmost red point). The next two hardest instances (in terms of the average time) were solved only by CLAS (in 363 and 184 seconds), while all other solvers timed out. A similar pattern is observed for DC-ST (Fig. 5), as well as for SE-ST and SE-PR (plots omitted due to space constraints).

The datasets on which CLAS struggles are from the `crusti*` and `scc*` collections. These frameworks exhibit a specific structure with clearly separated connected components, which our current approach does not exploit. In the next section, we show how this structure can be leveraged to compute preferred extensions.

## 5 Leveraging Connected Components: The Case of Preferred Extensions

In this section, we present an approach applicable to a wide category of argumentation frameworks: those with multiple *strongly connected components*, i.e., maximal subgraphs where there is a directed path between every two vertices. The role of connected components in argumentation semantics has been explored in (Baroni, Giacomin, and Guida 2005). Here, we use a simpler characterization, more amenable to algorithm design. We focus on preferred extensions and leave stable semantics for future work.

**Definition 6.** *A* source component *of an argumentation framework $(A, R)$ is its strongly connected component $(C, R_C)$ such that $(a, b) \notin R$ for all $a \in A \setminus C$ and $b \in C$.*

Our approach is to identify a source component, find a complete extension there, and, depending on that extension, restrict further work to a residual framework of one of the following two types:

**Definition 7.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework, and let $S \subseteq A$. Define*

$$\mathcal{F} - S = (\overline{S}, (\overline{S} \times \overline{S}) \cap R),$$
$$\mathcal{F} \dot{-} S = (\overline{S}, (\overline{S} \times \overline{S}) \cap (R \cup R_S)),$$

*where $\overline{S} = A \setminus S$ and $R_S = \{(v, v) \mid v \in R(S)\}$.*

In $\mathcal{F} - S$, we simply remove arguments in $S$ from $\mathcal{F}$. In $\mathcal{F} \dot{-} S$, we additionally add loops to all remaining arguments attacked by $S$ in $\mathcal{F}$, thus preventing their inclusion in any conflict-free set and in any extension.

**Proposition 5.** *Let $\mathcal{F}_C = (C, R_C)$ be a source component of $\mathcal{F} = (A, R)$. If $S \subseteq A$ is admissible in $\mathcal{F}$, then $S \cap C$ is admissible both in $\mathcal{F}$ and in $\mathcal{F}_C$.*

*Proof.* As $S$ is conflict-free, so is $S \cap C$. As $S$ is self-defending, whenever $(a, c) \in R$ for some $c \in S \cap C$, we must have $(d, a) \in R$ for some $d \in S$. Since $\mathcal{F}_C$ is a source component and $c \in C$, both $a$ and $d$ must belong to $C$. In particular, $d \in S \cap C$ and $(d, a) \in R_C$. Hence, $S \cap C$ is self-defending in $\mathcal{F}$ and in $\mathcal{F}_C$. □

**Proposition 6.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework, and let $B \subseteq A$. If $S \subseteq A \setminus B$ is admissible in $\mathcal{F}$, then it is also admissible in $\mathcal{F} \dot{-} B$.*

| | DC-CO | | | DC-ST | | | SE-ST | | | SE-PR | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | fastest | failure | PAR2 | fastest | failure | PAR2 | fastest | failure | PAR2 | fastest | failure | PAR2 |
| CLAS | 280 | 26 | 204 | 268 | 32 | 243 | 228 | 32 | 246 | 223 | 43 | 327 |
| FUDGE | 13 | 10 | 86 | 29 | 8 | 71 | 44 | 12 | 100 | 37 | 11 | 100 |
| $\mu$-toksia | 14 | 10 | 85 | 9 | 8 | 71 | 29 | 11 | 98 | 38 | 10 | 100 |
| SCALOP | 7 | 11 | 93 | 8 | 9 | 78 | 11 | 11 | 94 | 15 | 10 | 95 |
| All failed | 8 | | | 8 | | | 10 | | | 9 | | |

Table 1: Results for DC-CO, DC-ST, SE-ST, and SE-PR with CLAS implementing Algorithm 1

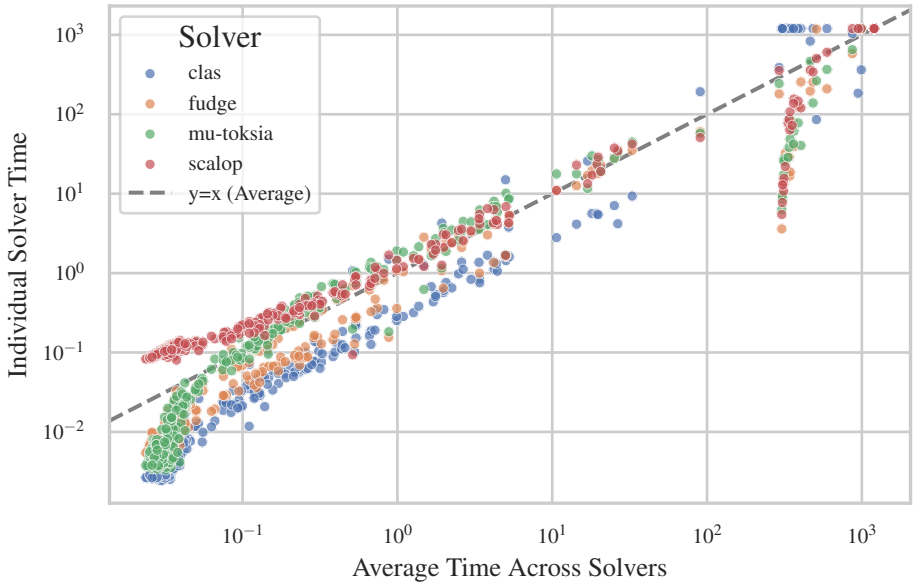


Figure 4: DC-CO: Solver times vs. average time (timeouts treated as 1200)

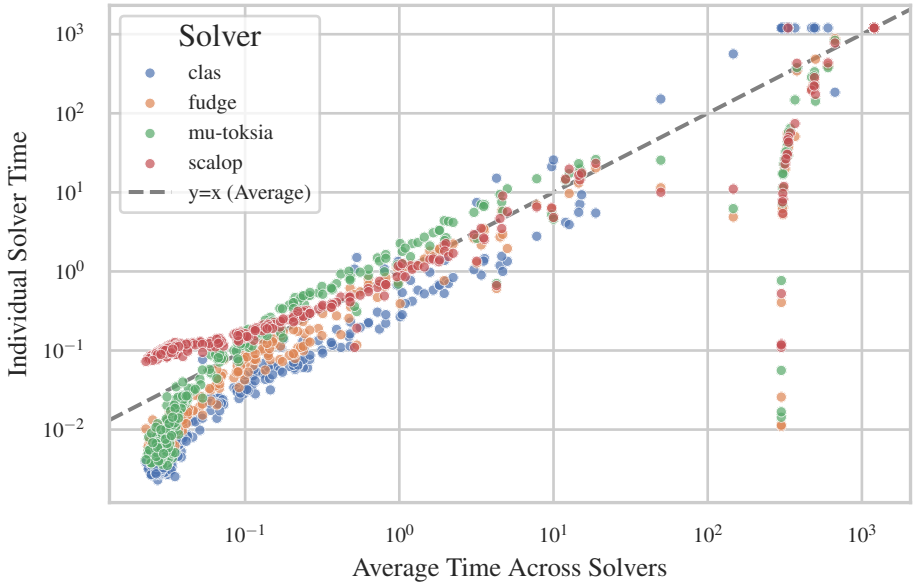


Figure 5: DC-ST: Solver times vs. average time (timeouts treated as 1200)

*Proof.* Let $(a, s) \in R$ for some $a \in A$ and $s \in S$. Since $S$ is self-defending in $\mathcal{F}$, there is $d \in S$ such that $(d, a) \in R$. As $S \cap B = \emptyset$, we have $d \notin B$, which means $d$ defends $S$ against $a$ also in $\mathcal{F} \doteq B$ (unless $a \in B$, in which case it is not an element $\mathcal{F} \doteq B$). Hence, $S$ is self-defending and thus admissible in $\mathcal{F} \doteq B$. □

**Lemma 1.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework, and let $\mathcal{F}_C = (C, R_C)$ be a source component of $\mathcal{F}$ such that its only complete extension (or, equivalently, its only admissible set) is empty. Then $S \subseteq A$ is a preferred extension of $\mathcal{F}$ if and only if $S \cap (R(C) \cup C) = \emptyset$ and $S$ is a preferred extension of $\mathcal{F} \doteq C$.*

*Proof.* Let $S \subseteq A \setminus C \setminus R(C)$ be a preferred extension of $\mathcal{F} \doteq C$. It is admissible in $\mathcal{F}$, since no argument from $C$ attacks $S$. Hence, $\mathcal{F}$ must have a preferred extension $T \supseteq S$. By Proposition 5, $T \cap C$ is admissible in $\mathcal{F}_C$ , which implies $T \cap C = \emptyset$. By Proposition 6, $T$ is admissible in $\mathcal{F} \doteq C$. From $S$ being a preferred extension of $\mathcal{F} \doteq C$, it follows $T = S$, and $S$ is a preferred extension of $\mathcal{F}$.

In the other direction, suppose that $S$ is a preferred extension of $\mathcal{F}$. In this case, $S \cap C$ is admissible in $\mathcal{F}_C$ by Proposition 5, and $S \cap C = \emptyset$, i.e., $S \subseteq A \setminus C$. By Proposition 6, $S$ remains admissible in $\mathcal{F} \doteq C$. Let $T \supseteq S$ be a preferred extension of $\mathcal{F} \doteq C$. We have $T \cap R(C) = \emptyset$, since, in $\mathcal{F} \doteq C$, all arguments from $R(C)$ attack themselves. Suppose that $T \neq S$. Then $T$ is not admissible in $\mathcal{F}$, because $S$ is a preferred extension of $\mathcal{F}$. This means that there is $a \in A$ such that $a$ attacks $T$ but $a \notin R(T)$. It is necessary that $a \in C$, since $T$ is admissible in $\mathcal{F} \doteq C$ and, hence, defends itself against all attackers from $A \setminus C$. However, as shown above, $T \cap R(C) = \emptyset$; therefore, no such attacker $a \in C$ is possible. Hence, $T = S$ and $S$ is a preferred extension of $\mathcal{F} \doteq C$. □

**Lemma 2.** *Let $\mathcal{F} = (A, R)$ be an argumentation framework, let $E \subseteq A$ be a complete extension of $\mathcal{F}$, and let $D$ be a preferred extension of $\mathcal{F} - E - R(E)$. Then $D \cup E$ is a preferred extension of $\mathcal{F}$.*

*Proof.* Both $D$ and $E$ are conflict-free. $E$ does not attack $D$, since $D \subseteq A \setminus E \setminus R(E)$. As $E$ is self-defending in $\mathcal{F}$, we have $R^{-1}(E) \subseteq R(E)$. Hence, $D \cap R^{-1}(E) = \emptyset$ and $D$ does not attack $E$. It follows that $D \cup E$ is conflict-free.

Suppose that $(a, b) \in R$ for some $a \in A$ and $b \in D \cup E$. If $a \in R(E)$, then $D \cup E$ defends itself against $a$. Otherwise, $a \in A \setminus E \setminus R(E)$, and, since $D$ is a preferred extension of $\mathcal{F} - E - R(E)$, there must be $d \in D$ such that $(d, a) \in R$. Therefore, $D \cup E$ is self-defending and admissible in $\mathcal{F}$.

Let $T \supseteq S$ be a preferred extension of $\mathcal{F}$. Since $E \subseteq T$ and $T$ is conflict-free, $T \setminus E \subseteq A \setminus E \setminus R(E)$. We claim that $T \setminus E$ is admissible in $\mathcal{F} - E - R(E)$. It is conflict-free, since $T$ is conflict-free. Let $(a, b) \in R$ for some $a \in A \setminus E \setminus R(E)$ and $b \in T \setminus E$. Since $T$ is a preferred extension of $\mathcal{F}$, there is $d \in T$ such that $(d, a) \in R$. As $a \notin R(E)$, we have $d \in T \setminus E$, and $d$ defends $b$ in $\mathcal{F} - E - R(E)$. It follows that $T \setminus E$ is self-defending and, hence, admissible in $\mathcal{F} - E - R(E)$. From $D \subseteq T \setminus E$ being a preferred extension of $\mathcal{F} - E - R(E)$, we conclude $D = T \setminus E$ and $T = D \cup E$. Thus, $D \cup E$ is a preferred extension of $\mathcal{F}$. □

**Algorithm 2** Procedure `preferred-extension`($\mathcal{F}$)

**Input**: An argumentation framework $\mathcal{F} = (A, R)$
**Output**: A preferred extension $X$ of $\mathcal{F}$

```
1:  X := ∅
2:  while F is not empty do
3:     Find a source component F_C = (C, R_C) of F
4:     S := semi-complete(F_C, ∅)
5:     E := complete-extension(F_C, S, ∅)
6:     if E = ⊥ then
7:        F := F ∸ C
8:     else
9:        E := semi-complete(F, E)
10:       X := X ∪ E
11:       F := F − E − R(E)
12: return X
```

Lemmas 1 and 2 suggest the following approach to solving the SE-PR task: find a source component $\mathcal{F}_C = (C, R_C)$; if its only complete extension is empty, continue the search in $\mathcal{F} \doteq C$ (Lemma 1); otherwise, use a complete extension of $\mathcal{F}_C$ to form for a complete extension $E$ of $\mathcal{F}$, compute a preferred extension $D$ in $\mathcal{F} - E - R(E)$, and return $D \cup E$ (Lemma 2). Algorithm 2 implements this approach. It uses a call to Algorithm 1 to compute a non-empty complete extension in line 5.

**Theorem 1.** *Algorithm 2 outputs a preferred extension of its input argumentation framework $\mathcal{F} = (A, R)$.*

*Proof.* The computed preferred extension is accumulated in $X$, which is initially empty. Each iteration of the algorithm starts by semi-completing the empty set in the framework $\mathcal{F}_C$ induced by a source component of $\mathcal{F}$ and setting $S$ to the result. Since every complete extension is semi-complete, any complete extension of $\mathcal{F}_C$ must include $S$. We use the version of Algorithm 1 for complete extensions to find a complete extension of $\mathcal{F}_C$ that includes $S$.

If no such extension is found, we are in the situation described by Lemma 1. Hence, the algorithm procedes with a search for a preferred extension in $\mathcal{F} \doteq C$, which is guaranteed to be a preferred extension of $\mathcal{F}$.

If a complete extension $E$ is found in $\mathcal{F}_C$, it is semi-completed to include all arguments defended by $E$ in $\mathcal{F}$. Since $E \subseteq C$, all attackers of $E$ are in $C$, and, since $E$ is self-defending in $\mathcal{F}_C$, it is self-defending in $\mathcal{F}$, too. Obviously, $E$ remains self-defending after being extended with arguments it defends and, hence, becomes a complete extension of $\mathcal{F}$. The conditions of Lemma 2 apply. Therefore, the algorithm adds $E$ to $X$ and computes the remaining part of the preferred extension of $\mathcal{F}$ as a preferred extension of $\mathcal{F} - E - R(E)$. □

Note that Algorithm 2 is not required to use Algorithm 1 to compute complete extensions; any algorithm that produces a non-empty complete extension containing a given set $S$, or correctly reports that none exists, can be used to implement the `complete-extension` procedure. In this

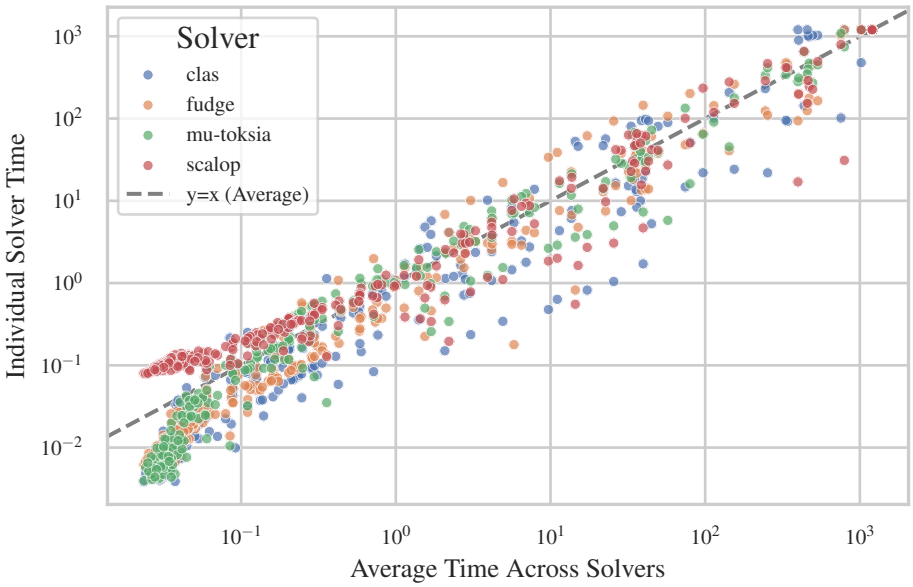


Figure 6: SE-PR: Solver times vs. average time (timeouts treated as 1200)

| | SE-PR | | |
|---|---|---|---|
| | fastest | failure | PAR2 |
| CLAS | 167 | 12 | 113 |
| FUDGE | 80 | 11 | 100 |
| $\mu$-toksia | 49 | 10 | 100 |
| SCALOP | 17 | 10 | 95 |
| All failed | | 9 | |

Table 2: Results for SE-PR with CLAS implementing Algorithm 2

context, it would be interesting to investigate whether incorporating the decomposition approach of Algorithm 2 into other solvers is beneficial. However, this lies beyond the scope of this paper.

### 5.1 Experimental Evaluation

The experimental results on the same 322 benchmarks are presented in Table 2 and Figure 6. While CLAS remains the fastest solver in the majority of cases, the number of failures has decreased significantly and is now on par with the other solvers.

Each of the twelve frameworks on which CLAS fails either consists of a single strongly connected component or has a very large source component. In such cases, the size of the component prevents the algorithm from computing a non-empty complete extension within the time limit (or verifying that none exists), causing it to time out in line 5. Consequently, the decomposition in Algorithm 2 provides little to no benefit in these cases, although it incurs only minimal overhead. Nevertheless, our experiments indicate that the SCC-based approach significantly speeds up computation in many other cases.

## 6 Conclusion

We have presented a new approach for computing extensions of AFs. Our approach is direct in the sense that it does not translate the given AF into other formalisms, such as Boolean formulas or CSPs, as is common in many state-of-the-art approaches. We show that the extensions of an AF are contained in a closure system of what we call pseudo-complete sets and compute the extensions by enumerating these closed sets.

Our experiments show that, while this approach outperforms state-of-the-art solvers on most benchmarks from the latest International Competition on Computational Models of Argumentation, it lags behind them on a few larger instances that exhibit a particular structure with multiple connected components. This observation motivates the second contribution of our paper, which is specific to preferred semantics: an algorithm that extracts a source strongly connected component of a given AF, attempts to find a non-empty complete extension within it, and then computes a smaller residual framework. This process is repeated until the framework becomes empty. The union of the computed complete extensions then forms a preferred extension. When combining this decomposition approach with our closure-based algorithm for computing complete extensions, we observe a significant improvement on benchmarks where the purely closure-based algorithm performs worse than existing solvers.

In future work, we plan to extend the approach based on strongly connected components to complete and stable semantics. In particular, we will explore whether the SCC decomposition approach for computing complete extensions presented in (Rodrigues 2018) can be efficiently combined with Algorithm 1, by using our algorithm as a base procedure for computing extensions in individual components, or

with Algorithm 2, where their approach could be used to compute a complete extension in a source component.

We also plan to optimize the search in Algorithm 1. More specifically, we aim to investigate whether techniques similar to conflict-driven clause learning (CDCL), as used in SAT solvers, can be employed to learn indirect conflicts between arguments and prune the search tree more effectively.

The approach presented in Algorithm 1 can be extended to skeptical reasoning. Analogous to credulous reasoning, one can enumerate pseudo-complete sets until either (i) an extension not containing the queried argument is found (answer: "no"), or (ii) all such sets have been enumerated (answer: "yes"). This applies directly to complete and stable semantics. For preferred semantics, it is also applicable but requires additional bookkeeping, since checking whether a set is preferred is non-trivial. We consider an efficient implementation of this extension an interesting direction for future work.

## Acknowledgements

This work is partly supported by the Federal Ministry of Research, Technology and Space of Germany and by Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus in the program Center of Excellence for AI-research "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig" (ScaDS.AI); by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project 389792660 (TRR 248, Center for Perspicuous Systems); and in DAAD project 57616814 (SECAI, School of Embedded Composite AI) as part of the program Konrad Zuse Schools of Excellence in Artificial Intelligence.

## AI Declaration

The authors used ChatGPT in order to improve writing style. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

## References

Alviano, M. 2021. The pyglaf argumentation reasoner (ICCMA2021). *CoRR* abs/2109.03162.

Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An introduction to argumentation semantics. *Knowl. Eng. Rev.* 26(4):365–410.

Baroni, P.; Dunne, P. E.; and Giacomin, M. 2010. On extension counting problems in argumentation frameworks. In Baroni, P.; Cerutti, F.; Giacomin, M.; and Simari, G. R., eds., *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, 63–74. IOS Press.

Baroni, P.; Giacomin, M.; and Guida, G. 2005. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence* 168(1):162–210.

Bistarelli, S., and Santini, F. 2011. Conarg: A constraint-based computational framework for argumentation systems. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*, 605–612. IEEE Computer Society.

Bordat, J.-P. 1986. Calcul pratique du treillis de Galois d' une correspondance. *Mathématiques, Informatique et Sciences Humaines* 96:31–47.

Cerutti, F.; Gaggl, S. A.; Thimm, M.; and Wallner, J. P. 2017. Foundations of implementations for formal argumentation. *FLAP* 4(8).

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2):321–358.

Dunne, P. E., and Wooldridge, M. J. 2009. Complexity of abstract argumentation. In Simari, G. R., and Rahwan, I., eds., *Argumentation in Artificial Intelligence*. Springer. 85–104.

Elaroussi, M.; Nourine, L.; and Radjef, M. S. 2023. Lattice point of view for argumentation framework. *Ann. Math. Artif. Intell.* 91(5):691–711.

Ganter, B., and Wille, R. 2024. *Formal Concept Analysis – Mathematical Foundations*. Springer Cham, 2 edition.

Ganter, B. 1984. Two basic algorithms in concept analysis. Technical Report Preprint-Nr. 831, Technische Hochschule Darmstadt, Darmstadt, Germany.

Ganter, B. 2010. Two basic algorithms in concept analysis. In Kwuida, L., and Sertkaya, B., eds., *Proceedings of the 8th International Conference on Formal Concept Analysis, (ICFCA 2010)*, volume 5986 of *Lecture Notes in Artificial Intelligence*, 329–359. Springer-Verlag. Reprint of (Ganter 1984).

Godin, R.; Missaoui, R.; and Alaoui, H. 1995. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence* 11(2):246–267.

Grahne, G., and Zhu, J. 2005. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Transactions on Knowledge and Data Engineering* 17(10):1347–1362.

Han, J.; Cheng, H.; Xin, D.; and Yan, X. 2007. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery* 15(1):55–86.

Johnson, D. S.; Yannakakis, M.; and Papadimitriou, C. H. 1988. On generating all maximal independent sets. *Information Processing Letters* 27(3):119–123.

Kröll, M.; Pichler, R.; and Woltran, S. 2017. On the complexity of enumerating the extensions of abstract argumentation frameworks. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 1145–1152. ijcai.org.

Kuznetsov, S. O., and Obiedkov, S. 2002. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence* 14(2-3):189–216.

Kuznetsov, S. O. 1993. A fast algorithm for computing all intersections of objects in a finite semi-lattice. *Automatic Documentation and Mathematical Linguistics* 27(5):11–21.

Lagniez, J. M.; Lonca, E.; and Mailly, J.-G. 2025. Counterexample-Guided Abstraction Refinement for Assumption-based Argumentation. In *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning*, 694–706.

Niskanen, A., and Järvisalo, M. 2020. $\mu$-toksia: An efficient abstract argumentation reasoner. In Calvanese, D.; Erdem, E.; and Thielscher, M., eds., *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, 800–804.

Norris, E. M. 1978. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées* 23(2):243–250.

Nourine, L., and Raynaud, O. 1999. A fast algorithm for building lattices. *Information Processing Letters* 71(5-6):199–204.

Obiedkov, S., and Sertkaya, B. 2023. Computing stable extensions of argumentation frameworks using formal concept analysis. In Gaggl, S. A.; Martinez, M. V.; and Ortiz, M., eds., *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, volume 14281 of *Lecture Notes in Computer Science*, 176–191. Springer.

Outrata, J., and Vychodil, V. 2012. Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. *Information Sciences* 185(1):114–127.

Rodrigues, O. 2018. A forward propagation algorithm for the computation of the semantics of argumentation frameworks. In Black, E.; Modgil, S.; and Oren, N., eds., *Theory and Applications of Formal Argumentation*, 120–136. Cham: Springer International Publishing.

Thimm, M.; Cerutti, F.; and Vallati, M. 2021. Fudge: A light-weight solver for abstract argumentation based on SAT reductions. *CoRR* abs/2109.03106.

Uno, T.; Kiyomi, M.; and Arimura, H. 2005. Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, OSDM '05, 77–86. New York, NY, USA: Association for Computing Machinery.