Lattice-Based Approaches for Enumerating Stable Extensions in Abstract Argumentation Frameworks

Sergei Obiedkov[1,2][0000−0003−1497−4001] and Barış Sertkaya[3][0000−0002−4196−0150]

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

Abstract. We investigate two approaches for computing stable extensions in abstract argumentation frameworks. The first represents stable extensions using Formal Concept Analysis (FCA), where they form an antichain in the concept lattice of a formal context derived from the framework. To improve the performance of this approach on sparse frameworks, we combine it with a dynamic-programming strategy based on decomposition into strongly connected components. The second approach transforms the attack relation into an implicational system such that the complements of stable extensions are among its models. FCA algorithms can then be used to explore the corresponding lattice and enumerate stable extensions. We experimentally evaluate the proposed techniques.

Keywords: Abstract Argumentation · Stable Extensions · FCA · Strongly Connected Components · Implications

1 Introduction

Argumentation provides formal tools for representing and analyzing conflicting information. In abstract argumentation [**?**], arguments are represented as vertices of a directed graph and attacks between arguments as edges. Different semantics characterize acceptable sets of arguments, giving rise to computational tasks such as deciding acceptance or enumerating extensions [**?**].

This paper extends our previous FCA-based approach for computing stable extensions [**?**]. While the method performed well on dense frameworks, its efficiency decreased on sparse ones. To address this, we investigate two complementary directions. The first decomposes the argumentation framework into connected components and combines solutions of the resulting subproblems. The second is based on the implicational characterization from [**?**], where complements of stable extensions appear among the closed sets of the corresponding closure system.

The remainder of the paper is organized as follows. Section 2 recalls the required background on abstract argumentation and FCA. Section 3 reviews the FCA-based method from our previous work. Section 4 presents the decomposition approaches, and Section 5 describes the implicational one. Section 6 reports experimental results, and Section 7 concludes the paper.

2 Preliminaries

2.1 Abstract Argumentation Frameworks

We briefly recall the notions from abstract argumentation needed in this paper; for a general introduction see [**?**]. An *abstract argumentation framework* (AF) is a pair $F = (A, R)$, where A is a finite set of arguments and $R \subseteq A \times A$ is the attack relation. If $(a, b) \in R$, then a is said to *attack* b.

For a set $S \subseteq A$, we write $R(S)$ for the set of arguments attacked by elements of S and $R^{-1}(S)$ for the set of arguments attacking at least one element of S. For a singleton $\{a\}$, we simply write $R(a)$ and $R^{-1}(a)$. A set S attacks an argument a if $a \in R(S)$, and an argument a attacks a set S if $a \in R^{-1}(S)$. Furthermore, S is said to *defend* an argument a if every attacker of a is itself attacked by some argument from S.

A pair (A_1, R_1) is called a *subframework* of $F = (A, R)$ whenever $A_1 \subseteq A$ and $R_1 \subseteq R$.

Example 1. Figure 1, borrowed from [**?**], shows an AF over the argument set $A = \{a, b, c, d, e\}$. In this framework, argument a attacks b and c, while $\{b, c\}$ attacks both a and d. The argument d attacks c and e, and therefore also any set containing one of these arguments. The set $\{a, e\}$ defends d, since both attackers of d are attacked by at least one of its elements.

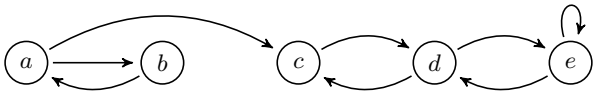

Fig. 1. Example of an argumentation framework.

A set $S \subseteq A$ is *conflict-free* if no two arguments in S attack each other. The family of all conflict-free sets of an AF F is denoted by $cf(F)$:

$$cf(F) = \{S \subseteq A \mid \forall a, b \in S : \ (\{a, b\} \times \{a, b\}) \cap R = \varnothing\}.$$

Let $F = (A, R)$ be an AF, and let $S \in cf(F)$. Then:

- S is *admissible* if it defends each of its elements;
- S is a *preferred extension* if it is inclusion-maximal among admissible sets;

– S is a *stable extension* if every argument outside S is attacked by some argument in S.

Every stable extension is preferred, since a stable extension necessarily defends all its elements. Stable extensions also coincide with independent dominating sets of the directed graph underlying the AF, i.e., sets of vertices that are pairwise non-adjacent and such that every vertex outside the set has an incoming edge from some vertex in the set. For the framework in Figure 1, the preferred extensions are $\{a, d\}$, $\{b, c\}$, and $\{b, d\}$, while the stable extensions are $\{a, d\}$ and $\{b, d\}$.

A central computational task in abstract argumentation is to determine extensions under a given semantics. This is computationally difficult for many semantics of practical interest [**?**,**?**]. Many state-of-the-art systems tackle them by translating AF reasoning problems into other well-studied formalisms, such as SAT or answer-set programming, and then applying optimized external solvers. Examples include μ-toksia [**?**], taas-fudge [**?**], pyglaf [**?**], and Scalop [**?**]. Other approaches rely on algorithms developed specifically for argumentation. A survey of these methods is given in [**?**]. Benchmark comparisons are regularly provided by the International Competition on Computational Models of Argumentation (ICCMA).

2.2 Formal Concept Analysis

Formal Concept Analysis (FCA) [**?**] provides mathematical tools for describing dependencies and hierarchical structures in data. In FCA, data are represented by a *formal context* $\mathbb{K} = (G, M, I)$, where G is a set of *objects*, M a set of *attributes*, and $I \subseteq G \times M$ the *incidence relation*. An incidence $(g, m) \in I$ means that object g has attribute m.

For $A \subseteq G$ and $B \subseteq M$, define

$$A^{\uparrow} = \{m \in M \mid \forall g \in A : \ (g, m) \in I\},$$

$$B^{\downarrow} = \{g \in G \mid \forall m \in B : \ (g, m) \in I\}.$$

The operators $(\cdot)^{\uparrow\downarrow}$ and $(\cdot)^{\downarrow\uparrow}$ are closure operators on G and M, respectively.

A *formal concept* of $\mathbb{K}$ is a pair (A, B) such that

$$A^{\uparrow} = B \qquad \text{and} \qquad B^{\downarrow} = A.$$

Here, A is the *extent* and B the *intent* of the concept. Formal concepts ordered by extent inclusion (equivalently, reverse intent inclusion) form a complete lattice called the *concept lattice* of $\mathbb{K}$.

For every $A \subseteq G$, the pair $(A^{\uparrow\downarrow}, A^{\uparrow})$ is a formal concept, and $A^{\uparrow\downarrow}$ is the smallest extent containing A. Hence, extents are precisely the subsets of G closed under $(\cdot)^{\uparrow\downarrow}$, and intents are precisely the subsets of M closed under $(\cdot)^{\downarrow\uparrow}$. Since arbitrary intersections of extents and intents are again extents and intents, they form closure systems on G and M [**?**].

Several algorithms for enumerating formal concepts are known, including polynomial-delay [**?**] algorithms; see [**?**] for a comparison.

Implications provide another description of concept intents. For $A, B \subseteq M$, the implication $A \to B$ is *valid* in $\mathbb{K}$ if $A^\downarrow \subseteq B^\downarrow$. A set $X \subseteq M$ satisfies $A \to B$ if either $A \not\subseteq X$ or $B \subseteq X$. Given a set $\mathcal{L}$ of implications, we write $X \models \mathcal{L}$ if X satisfies every implication in $\mathcal{L}$.

Any implication set $\mathcal{L}$ over M induces a closure operator

$$X \mapsto \mathcal{L}(X),$$

where

$$\mathcal{L}(X) = \bigcap \{Y \mid X \subseteq Y \subseteq M, \quad Y \models \mathcal{L}\}.$$

The models of $\mathcal{L}$ form a lattice. In particular, if $\mathcal{L}$ is the set of implications valid in $\mathbb{K}$, then its models are exactly the intents of $\mathbb{K}$ and

$$\mathcal{L}(X) = X \iff X^{\downarrow\uparrow} = X.$$

We also allow implications of the form $A \to \bot$. A set $X \subseteq M$ satisfies such an implication iff $A \not\subseteq X$. Such implications exclude M from the model set, so the resulting structure is generally only a semilattice. Moreover, if $A \subseteq X$ for some implication $A \to \bot \in \mathcal{L}$, we define $\mathcal{L}(X) = \bot$.

3 Stable Extensions as Concept Intents

In [**?**], following the ideas from [**?**], we have presented a characterization of AFs as formal contexts. In this characterization, stable extensions of the original AF are concept intents of the constructed formal context. Using algorithms for enumerating concept intents, we have shown how stable extensions can be enumerated. We briefly recall this approach.

Given an AF $F = (A, R)$, the *induced formal context* of (A, R) is

$$\mathbb{K}(A, R) = (A, A, (A \times A) \setminus R).$$

In this context, for $S \subseteq A$,

$$S^\uparrow = \{t \in A \mid \forall s \in S \colon (s, t) \notin R\}$$

is the set of arguments not attacked by S and

$$S^\downarrow = \{t \in A \mid \forall s \in S \colon (t, s) \notin R\}$$

is the set of arguments that do not attack S. Then

- S defends $x \in A$ iff $S^\uparrow \subseteq \{x\}^\downarrow$;
- S is conflict-free iff $S \subseteq S^\uparrow$ or, equivalently, $S \subseteq S^\downarrow$;
- S is an admissible set iff $S \subseteq S^\uparrow \subseteq S^\downarrow$;
- if S is a preferred extension, then $S = S^{\downarrow\uparrow}$;

– S is a stable extension iff $S = S^{\uparrow}$.

The stable and preferred extensions are thus concept intents of $\mathbb{K}(A, R)$. Moreover, the concepts whose intents are conflict-free form an order filter in the concept lattice of $\mathbb{K}(A, R)$, and the antichain of preferred extensions constitutes the border of this filter containing the stable extensions. As an example, Fig. 2 shows the concept lattice of the formal context induced by the argumentation framework in Fig. 1.

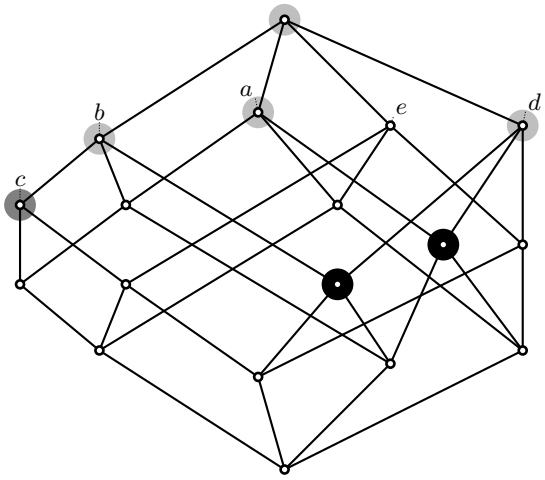

Fig. 2. The concept lattice of the formal context induced by the argumentation framework in Fig. 1. Object labels are omitted. The stable extensions correspond to the intents of the concepts depicted as black circles; the only preferred extension that is not stable, $\{b, c\}$, is depicted as a dark-gray circle; and the remaining conflict-free intents are depicted as light-gray circles.

In [**?**], we use FCA lattice-construction algorithms to enumerate all conflict-free concept intents. When a maximal conflict-free intent is encountered, the current computation branch is pruned. If this intent is stable, we output it.

We adapted two lattice construction algorithms in this way: Next Closure [**?**] and the incremental algorithm proposed in [**?**]. Next Closure enumerates all concept intents with a polynomial delay. It lists intents included in intent S before it produces S, thus being suitable for exploring the order filter of the concept lattice. Its other advantage is that it needs memory that is only linear in the number of arguments. Finally, it can be used to enumerate closed sets of an arbitrary closure operator, which will be important for Section 5.

The incremental algorithm processes attributes of the input formal context one by one and stores all generated concepts to ease generation of new concepts. This means that it may need exponential amount of memory, but, when it is available, the algorithm is usually faster than Next Closure.

As can be seen from the experimental results presented in [?], enumerating stable extensions with concept-generation algorithms is efficient for dense frameworks. However, sparse frameworks are only feasible when they are small. This suggests that, to scale these algorithms to sparse frameworks, one could try to first decompose the input framework into smaller parts.

4 Scaling Up with Connected Components

In this section, we present two approaches to argumentation framework decomposition. The first approach yields a simple divide-and-conquer algorithm based on decomposing the framework into weakly connected components. The second approach subsumes the first by using strongly connected components instead. In this case, we propose a dynamic-programming approach in which stable extensions of individual components induce smaller subproblems in the remaining part of the framework, which are then solved recursively. Both approaches assume that another algorithm is used to compute stable extensions in the base case. In our experiments, we use lattice-based algorithms for this purpose, although other algorithms could also be used.

4.1 Weakly Connected Components

An immediate option is to decompose the framework into *weakly connected components* (WCC), i.e., components of the underlying undirected graph.

Proposition 1. *Let $F = (A, R)$ be an argumentation framework and $F_1 = (A_1, R_1), \ldots, F_k = (A_k, R_k)$ be all its weakly connected components. Then $S \subseteq A$ is a stable extension of F if and only if $S \cap A_i$ is a stable extension of F_i for every $1 \leq i \leq k$.*

Proof. Since there are no attacks among arguments from different components, S is conflict-free if and only if $S \cap A_i$ is conflict-free for every $1 \leq i \leq k$. For the same reason, S attacks $a \in A_i \setminus S$ if and only if $S \cap A_i$ attacks a. As $A = \bigcup_i A_i$, the statement follows.

Example 2. Consider the argumentation framework shown in Fig. 3. It comprises two WCCs induced by argument subsets $\{a, b\}$ and $\{c, d, e\}$. The first component has two stable extensions, $\{a\}$ and $\{b\}$, while the second component has only one, $\{d\}$. The stable extensions of the entire framework are obtained by forming all possible unions of stable extensions from different components, resulting in $\{a, d\}$ and $\{b, d\}$.

Proposition 1 suggests a straightforward divide-and-conquer approach: decompose the argumentation framework into WCCs, compute stable extensions in each component using any available algorithm, and then form the unions of extensions resulting from different components. If a component has no stable extension, then the entire framework has none either. If we are only interested

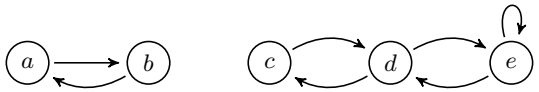

Fig. 3. An argumentation framework with two weakly connected components.

in finding a single extension, it is sufficient to find a single extension in each component and then form their union.

Since decomposition into WCCs can be achieved in linear time, this approach incurs very little overhead. However, its applicability is limited to frameworks that are not weakly connected, i.e., have multiple WCCs.

4.2 Strongly Connected Components

In this section, we present an approach applicable to a much wider category of argumentation frameworks: those with multiple *strongly connected components* (SCC), i.e., maximal subgraphs where there is a directed path between every two vertices.

As an example, the directed graph in Fig. 1 has one WCC, but two SCCs, coinciding with the WCCs of the graph in Fig. 3: $G_1 = (\{a, b\}, \{(a, b), (b, a)\})$ and $G_2 = (\{c, d, e\}, \{(c, d), (d, c), (d, e), (e, d), (e, e)\})$. SCCs of a directed graph can be computed in linear time using Tarjan's algorithm [**?**].

Decomposing frameworks into SCCs and computing stable extensions independently in each component, as we did for weakly connected components, is not sufficient, since there may be attacks between different components. A subtler approach is needed.

Definition 1. *A strongly connected component* (A_1, R_1) *of an argumentation framework* (A, R) *is called its* source component *if* $(a, b) \notin R$ *for all* $a \in A \setminus A_1$ *and* $b \in A_1$.

An argumentation framework can have multiple source components.

Definition 2. *Let* $F = (A, R)$ *be an argumentation framework,* $F_1 = (A_1, R_1)$ *be its subframework, and* $S \subseteq A_1$. *The* residual (sub)framework *of* F *w.r.t.* F_1 *and* S*, denoted by* $F \mid (F_1, S)$*, is the subframework of* F *obtained by removing from* A *the arguments in* A_1*, as well as the arguments attacked by* S*:*

$$F \mid (F_1, S) = (B, (B \times B) \cap R),$$

where $B = A \setminus (A_1 \cup R(S))$.

Theorem 1. *Let* $F = (A, R)$ *be an argumentation framework, and let* C *be any of its source components.* $E \subseteq A$ *is a stable extension of* F *if and only if* $E = E_1 \cup E_2$*, where* E_1 *is a stable extension of* C *and* E_2 *is a stable extension of the residual framework* $F \mid (C, E_1)$.

Algorithm 1 SCC-STABLE-EXTENSIONS(F)

Input: Argumentation framework $F = (A, R)$
Parameter: A base procedure STABLEEXTENSIONS(H) computing stable extensions for its input framework H
Output: The stable extensions of F

1. Find a source component C in F.
2. Compute its stable extensions with the base procedure:
   $\mathcal{E} :=$ STABLEEXTENSIONS(C).
3. For each extension $E_1 \in \mathcal{E}$:
   (a) Form the residual framework $F_r := F \mid (C, E_1)$.
   (b) Recursively compute stable extensions $\mathcal{E}_r$ of F_r:
   $\mathcal{E}_r :=$ SCC-STABLE-EXTENSIONS(F_r)
   (c) For every $E_2 \in \mathcal{E}_r$, output $E_1 \cup E_2$.

Proof. Suppose that E_1 is a stable extension of C and E_2 is a stable extension of $F_r = F \mid (C, E_1)$, and let $E = E_1 \cup E_2$. Arguments from E_1 do not attack E_2, since the residual framework F_r contains no arguments attacked by E_1. Arguments from E_2 do not attack E_1, since arguments from E_1 belong to a source component of F and those from E_2 belong to a different component. Therefore, given that E_1 and E_2 are both conflict-free, so is E.

Take an argument $a \in A \setminus E$. We know that E_1 attacks all arguments in C. Hence, if a is not attacked by E_1, it is in F_r, and then it is attacked by E_2. Thus, E attacks every argument in $A \setminus E$ and, since it is conflict-free, it is a stable extension.

Conversely, assume that E is a stable extension of F and let $E_1 = E \cap C$, $E_2 = E \setminus E_1$, and $F_r = F \mid (C, E_1)$. As subsets of the conflict-free set E, both E_1 and E_2 are conflict-free, too.

Since C is a source component, arguments outside C cannot attack C. Hence, for E to be a stable extension, it is necessary that E_1 attacks all other arguments in C. Therefore, E_1 is a stable extension of C.

By the definition of a residual framework, arguments of F_r are not attacked by E_1. Still, they are all attacked by E, and this must be due to E_2, which is then a stable extension of F_r.

Theorem 1 gives rise to Algorithm 1, which enumerates all stable extensions of an argumentation framework $F = (A, R)$. This recursive algorithm is parameterized with a procedure STABLEEXTENSIONS(H), which is used to compute stable extensions in the base case, i.e., for the source component of F. Residual frameworks are then formed w.r.t. the component and each of its stable extensions, and the extensions in the residual frameworks are computed recursively.

Theorem 2. *Algorithm 1 outputs only stable extensions of F. It outputs every stable extension of F exactly once.*

Proof. Theorem 1 guarantees that Algorithm 1 outputs all and only the stable extensions of F. It may generate a stable extension E more than once only if

there are at least two ways to represent E as the union of stable extensions of respective subframeworks: $E_1 \cup E_2 = E = E'_1 \cup E'_2$, where E_1 and E'_1 are stable extensions of the same source component C, while E_2 and E'_2 are stable extensions of the residual frameworks $F \mid (C, E_1)$ and $F \mid (C, E'_1)$. Note that, $E_1 = E'_1$ if and only if $E_2 = E'_2$, since each of E_1 and E'_1 is disjoint with both E_2 and E'_2.

If $E_1 \neq E'_1$, there must be $a \in C$ that belongs to only one of E_1 and E'_1. Without loss of generality, assume that $a \in E_1$. From $E_1 \cup E_2 = E'_1 \cup E'_2$, it then follows that $a \in E'_2$, which is impossible, since $E'_2 \cap C = \varnothing$. Therefore, $E_i = E'_i$ for $i \in \{1, 2\}$ and, hence, every stable extension of F is produced exactly once.

Example 3. Consider the framework F in Figure 1. It has only one source component, C, induced by $\{a, b\}$. This component has two stable extensions: $\{a\}$ and $\{b\}$.

Since c is the only argument outside C attacked by a, the residual framework $F \mid (C, \{a\})$ is obtained from F by removing c, along with a and b. Hence, it is induced by $\{d, e\}$. This framework is strongly connected, so its stable extensions are computed using the base algorithm. Its only stable extension is $\{d\}$, which yields the stable extension $\{a, d\}$ of F.

Similarly, the component extension $\{b\}$ gives rise to the residual framework induced by $\{c, d, e\}$. This framework is also strongly connected, and its only stable extension is again $\{d\}$. Consequently, we obtain another stable extension of F, namely $\{b, d\}$.

As formulated, Algorithm 1 may have to process the same residual framework more than once, since it may happen that $F \mid (C, E_1) = F \mid (C, E_2)$ for some stable extensions $E_1 \neq E_2$ of C. Identical residual frameworks can also occur at different levels of recursion. In recursive implementations of dynamic-programming algorithms, such unnecessary computation is usually avoided by memoizing solutions for subproblems. Here, we store sets of stable extensions computed for residual frameworks in a hashtable. The key used to retrieve the stable extensions for a residual framework is its set of arguments. Before issuing a recursive call in step 3b, we check if the hashtable already contains the stable extensions of F_r and, if so, use them in subsequent computations.

5 Stable Extensions via Implication Systems

In this section, we present quite a different approach extending the results from [**?**]. While our FCA-based approach represents stable extensions as elements of the lattice of concept intents of a formal context derived from the argumentation framework, here the idea is to translate the framework into a set of implications and explore the lattice of its models.

Elaroussi et al. [**?**] focus on admissible sets. Recall that an admissible set is a conflict-free set that defends all its elements. Consider two arguments x and y such that $(x, y) \in R$. If $(y, x) \in R$, then y defends itself against x. Otherwise, every self-defending set S containing y must contain an attacker of x. In other

words, if S contains no attacker of x, it cannot contain y, which means that $A \setminus S$ is a model of the implication $R^{-1}(x) \to \{y\}$. This leads to the following result from [**?**]:

Theorem 3. *Let $F = (A, R)$ be an argumentation framework. Define*

$$\mathcal{SD}(F) := \{R^{-1}(x) \to \{y\} \mid (x, y) \in R \text{ and } (y, x) \notin R\}.$$

$S \subseteq A$ is a self-defending set if and only if $A \setminus S$ is a model of $\mathcal{SD}(F)$.

Example 4. Consider again the framework F in Figure 1. There is only one attack that does not reciprocate, (a, c). Thus, $\mathcal{SD}(F) = \{\{b\} \to \{c\}\}$. Self-defending sets are those that contain b or do not contain c.

As defined, $|\mathcal{SD}(F)| \leq |R|$. Note that $\mathcal{SD}(F)$ can be equivalently defined as follows:

$$\mathcal{SD}(F) := \{R^{-1}(x) \to R(x) \mid x \in A\}.$$

This representation contains $|A|$ implications; those with $R(x) \subseteq R^{-1}(x)$ can be removed.

To compute admissible sets, one can enumerate the models of $\mathcal{SD}(F)$ with Next Closure, consider their complements, and output those that are conflict-free.

However, we are interested in stable extensions. Every stable extension is admissible, but not vice versa. The difference is that a stable extension must attack all other arguments and not just attackers of its elements. Put differently, in addition to being self-defending, a stable extension must be a dominating set. We define an implication set $\mathcal{D}(F)$ that captures this condition.

Theorem 4. *Let $F = (A, R)$ be an argumentation framework. Define*

$$\mathcal{D}(F) := \{\{x\} \cup R^{-1}(x) \to \bot \mid x \in A\}.$$

$S \subseteq A$ is a self-defending dominating set if and only if $A \setminus S$ is a model of $\mathcal{SD}(F) \cup \mathcal{D}(F)$.

Proof. Given Theorem 3, it is sufficient to show that being a model of $\mathcal{D}(F)$ is equivalent to being the complement of a dominating set. Indeed, S is a dominating set if and only if, for every $x \in A$, it contains either x or an attacker of x and, therefore, $\{x\} \cup R^{-1}(x) \not\subseteq A \setminus S$.

Example 5. For the framework in Figure 1, $\mathcal{D}$ consists of the following implications:

$\{a, b\} \to \bot$ (from a and b)
$\{a, c, d\} \to \bot$ (from c)
$\{c, d, e\} \to \bot$ (from d)
$\{d, e\} \to \bot$ (from e)

The implications obtained from a and b coincide, since a and b are the only attackers of each other. The implication resulting from d is entailed by the one resulting from e and can be removed.

Theorem 4 makes it possible to enumerate the complements of self-defending dominating sets with Next Closure. For any two listed sets $X \subseteq Y$, the algorithm lists X before Y. Thus, the algorithm starts with the complements of large self-defending dominating sets, moving towards those of smaller ones. As soon as the algorithm encounters the complement of a conflict-free set S, it should output S as a stable extension and the current computation branch can be pruned, since subsets of S cannot be stable extensions.

This is an important difference from the concept-based algorithms: those algorithms explore the conflict-free part of the lattice of concept intents, while here we explore the non-conflict-free part of the semi-lattice of self-defending dominating sets. This suggests that concept-based algorithms may be more efficient on dense frameworks, with a relatively small conflict-free part, while the implication-based approach is more appropriate for sparse frameworks, with relatively few conflicts.

Still, it is desirable to reduce the search space by removing from it at least some of the non-conflict-free sets—or, rather, their complements. In [**?**], it is suggested to add or strengthen implications to capture some type of conflicts. The simplest type is the self-conflict: if $(x, x) \in R$, the implication $\varnothing \to \{x\}$ is added. Since a conflict-free set cannot contain such x, its complement must satisfy this implication.

The intuitive meaning of the implication $R^{-1}(x) \to \{y\}$ is that arguments from $R^{-1}(x)$ defend y against x, and, therefore, every self-defending set with y must contain at least one of these. However, if $(z, y) \in R$ or $(y, z) \in R$ for some $z \in R^{-1}(x)$, then z is useless as a defender of y in a conflict-free set, since y is itself in conflict with z. Therefore, if we are interested only in conflict-free sets, the implication can be strengthened to

$$R^{-1}(x) \setminus (R^{-1}(y) \cup R(y)) \to \{y\}. \tag{1}$$

Similarly, $z \in R^{-1}(x)$ is not a suitable defender of y against x if it is in conflict with every defender of y against some other argument x', i.e., if

$$(x', y) \in R \text{ and } R^{-1}(x') \subseteq R(z) \cup R^{-1}(z). \tag{2}$$

Therefore, the implication $R^{-1}(x) \to \{y\}$ can be strengthened by removing z from its premise.

In the latter case, z is in indirect conflict with y: although neither attacks the other, they cannot belong to the same conflict-free set. The strengthening rules described above can be generalized by replacing the attack relation R with a gradually constructed symmetric *indirect conflict relation*. The conflict relation C initially contains pairs (x, y) such that $\{x, y\}$ is not conflict-free, i.e., either x attacks y, y attacks x, or one of them attacks itself. The implication set $\mathcal{L}$ includes implications from $\mathcal{SD}(\mathcal{F})$ strengthened according to (1), as well as those from the set $\mathcal{D}(F)$ and $\{\varnothing \to \{x\} \mid (x, x) \in R\}$. The following two steps are then repeated until both the conflict relation and the implication set stabilize.

First, the conflict relation is updated. For every $x \in A$, the closure D of $C(x)$ under $\mathcal{L}$ is computed and C is updated with $(\{x\} \times D) \cup (D \times \{x\})$. The reason

for this is as follows. Since D is the closure of $C(x)$ under the implications, every stable extension disjoint with $C(x)$ must be disjoint with D. On the other hand, a conflict-free set containing x must be disjoint with $C(x)$ and, hence, with D. Thus, arguments from D are in (indirect) conflict with x.

Second, the implications from $\mathcal{L}$ are strengthened: implication $X \to \{y\}$ is replaced by $X \setminus C(y) \to \{y\}$. This is similar to (1) but using the indirect conflict relation instead of the attack relation. Due to the conflict-update step, this also covers case (2).

As soon as neither step results in updates, the implication set $\mathcal{L}$ is used to enumerate stable extensions as described above.

6 Experimental Results

To evaluate the performance of the approaches described in Sections 4 and 5, we implemented them in our FCA-based argumentation solver AFCA[4], introduced in [**?**]. We report evaluation results for the SCC-based algorithm from Section 4.2, using our adaptation of the incremental algorithm proposed by Norris in [**?**] (see Section 3) and the implication-based algorithm from Section 5 as base procedures.

For comparison, we used the solvers μ-toksia[5], taas-fudge[6], and an earlier version of scalop known as crustabri [7]. Only μ-toksia supports enumeration of stable extensions, but all three support computing a single stable extension (or detecting that no such extension exists). We therefore ran all three solvers in single-extension mode, while our algorithms were used to enumerate all stable extensions. Note that this is to our disadvantage, as our algorithms were tasked with a computationally harder problem.

To measure the effect of the number of connected components in an argumentation framework, we generated test data with varying numbers of arguments and components, and with varying densities within the components. More precisely, we generated random frameworks with n arguments and at least k strongly connected components using a parameter p with $0 < p < 1$ as follows. We first assigned n arguments to k components uniformly at random, ensuring that every component receives at least one argument. Then, for every argument x in component i and argument y in component j, we added an attack between x and y with probability p: the attack (x, y) if $i \leq j$ and the attack (y, x) otherwise. Thus, p corresponds to the expected density of attacks within components, while attacks between components are oriented so that the component order is respected. This guarantees that no two of the k subsets of arguments are merged into a single component. However, for smaller values of p, there may not be enough attacks within a subset to ensure that it is strongly connected. Therefore, the resulting framework may contain more than k strongly connected components.

[4] `https://github.com/sertkaya/afca`
[5] `https://bitbucket.org/andreasniskanen/mu-toksia`
[6] `https://github.com/aig-hagen/taas-fudge`
[7] `https://github.com/crillab/crustabri`

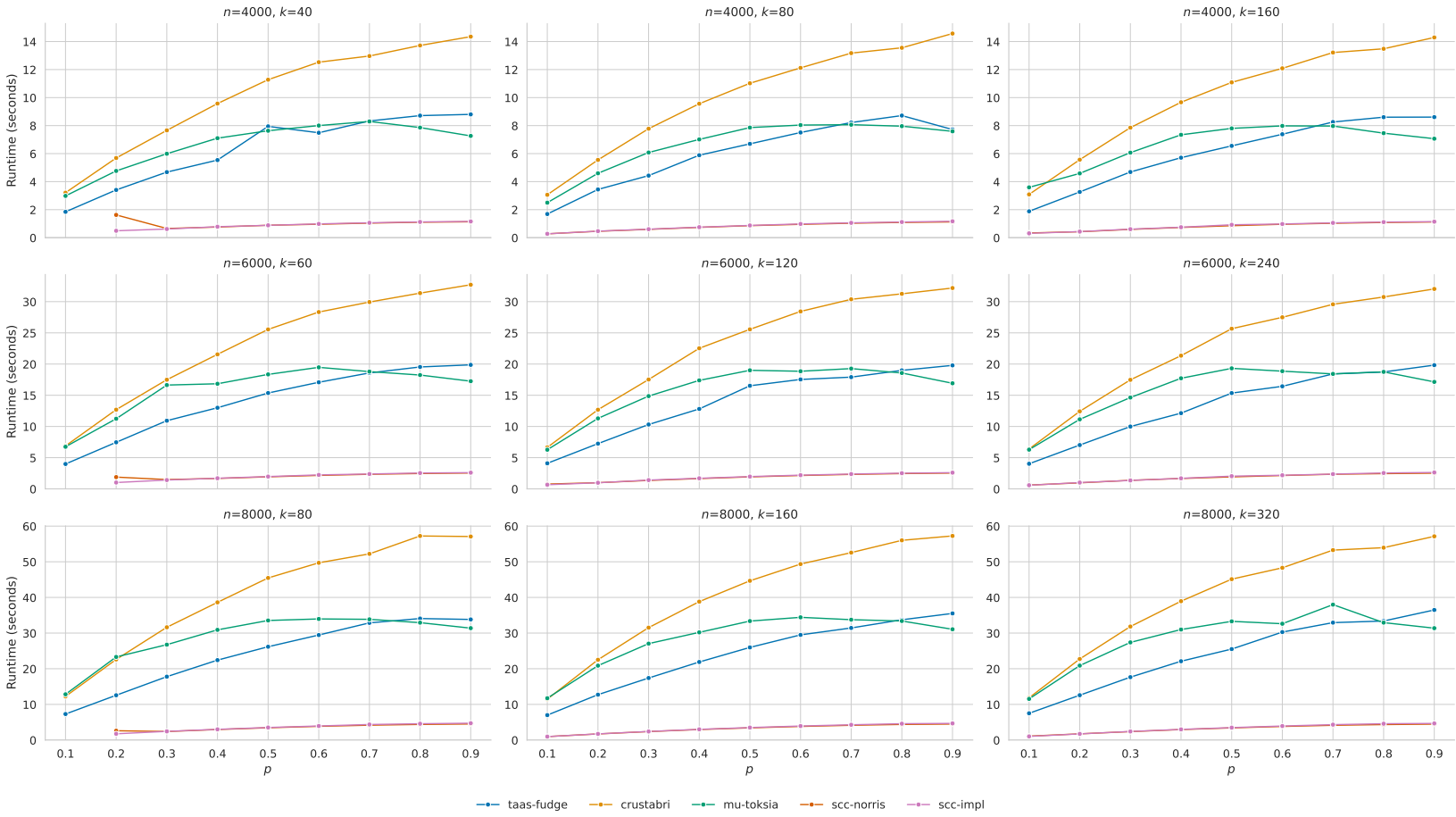

Fig. 4. Runtime comparison of algorithms.

The generated test data are available in the AFCA repository.

We performed the tests on a Linux machine with a 32-core CPU running at 2.9 GHz and 256 GB of main memory. We used a time limit of five minutes but did not impose any limit on memory usage. The runtimes are presented in Figure 4 for a total of 81 argumentation frameworks with 4000, 6000, and 8000 arguments. The minimal number k of connected components took the values 1%, 2%, and 4% of the number of arguments, and p varied from 0.1 to 0.9. Both scc-norris and scc-impl (shown in Figure 4) implement Algorithm 1, but differ in the base procedures: scc-norris uses the Norris-based algorithm, whereas scc-impl uses the algorithm from Section 5. For values of k between 80 and 320, the performances of scc-norris and scc-impl are so close to each other that the corresponding lines in Figure 4 cannot be distinguished.

For $k = n/100$ and $p = 0.1$, both scc-norris and scc-impl timed out for all three values of n (hence, there are no corresponding points in the plots), even though these instances were among the easiest for the other three solvers. This is consistent with what we observed in [**?**]: lattice-based algorithms are not usually efficient on sparse datasets even of moderate size. With $k = n/100$ and $p = 0.1$, we deal with components containing on average 100 arguments each with attack density of 0.1. This is still hard for lattice-based algorithms.

However, when we reduce the component size to 50 by increasing k to $n/50$, scc-norris and scc-impl become the fastest algorithms even when $p = 0.1$. When $p \geq 0.2$, scc-norris and scc-impl are the fastest already for $k = n/100$, although, in our experiments from [**?**], the density 0.2 was typically beyond the reach of the Norris-based algorithm used on its own. This is precisely the effect we hoped to achieve by decomposing sparse frameworks into smaller parts.

7 Conclusion

In our earlier work, we observed that FCA-based concept-enumeration algorithms are effective for enumerating stable extensions in dense frameworks, which induce sparse contexts with relatively few concepts and even fewer conflict-free intents. In this paper, we proposed a decomposition strategy that extends the applicability of these algorithms to sparser frameworks containing multiple dense or moderately sized connected components. As another approach, we presented an algorithm based on enumerating a closure system induced by an implication set, building on ideas from [**?**]. Both approaches were evaluated experimentally against existing solvers for stable extensions.

The experiments show that the SCC-based decomposition substantially improves the performance of concept-enumeration algorithms. In frameworks with many stable extensions, they tend to outperform existing tools by a large margin. Moreover, the approach is naturally parallelizable, which may lead to additional performance gains.

Although we evaluated the SCC decomposition only in combination with lattice-based algorithms, other algorithms for enumerating stable extensions can

also be used to handle the base case. Investigating such combinations is part of our future work.

Although we tested the SCC approach only in combination with lattice-based algorithms, other algorithms for enumerating stable extensions can be used to handle the base case. We plan to investigate the performance of various combinations in the future.

We also plan to adapt the proposed techniques to other semantics, in particular preferred semantics. Related directions have recently been explored in our work on computing single complete, stable, and preferred extensions through enumeration of closed sets [**?**].

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.