Computing Stable Extensions of Argumentation Frameworks using Formal Concept Analysis

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

[1] Knowledge-Based Systems Group, Faculty of Computer Science / cfaed / ScaDS.AI
TU Dresden, Germany
sergei.obiedkov@tu-dresden.de

[2] Frankfurt University of Applied Sciences, Germany
sertkaya@fb2.fra-uas.de

Abstract. We propose an approach based on Formal Concept Analysis (FCA) for computing stable extensions of Abstract Argumentation Frameworks (AFs). To this purpose, we represent an AF as a formal context in which stable extensions of the AF are closed sets called concept intents. We make use of algorithms developed in FCA for computing concept intents in order to compute stable extensions of AFs. Experimental results show that, on AFs with a high density of the attack relation, our algorithms perform significantly better than the existing approaches. The algorithms can be modified to compute other types of extensions, in particular, preferred extensions.

1 Introduction

Abstract 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, Argumentation Frameworks (AFs), has attracted increasing attention in the AI and particularly in the Knowledge Representation communities since its introduction by Dung in [10]. In AFs, arguments are abstracted away from their actual contents and conflicts are modelled in form of attacks between arguments. This abstraction allows an intiutive formalization using directed graphs. The semantics is defined through sets of arguments called extensions. Several different types of extensions of AFs have been proposed in the literature [3], which gave 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 have by now been well investigated [11,21]. There are also highly optimized solvers that can handle large problem instances. In the bi-annually organized International Competition on Computational Models of Argumentation (ICCMA), these 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 or the satisfiability problem of propositional logic, 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 [8].

In the present work, we propose an approach for computing extensions of AFs based on Formal Concept Analysis (FCA) [15]. To this purpose, we characterize an AF as a formal context. Such a characterization was first noted in [2]. We exploit the similarity between an AF and a formal context and employ algorithms from FCA to compute stable extensions. Our algorithms can be modified to compute extensions of other types, such as preferred extensions.

The paper is organized as follows. In Section 2, we introduce basic notions of AFs and FCA. In Section 3, we present a translation from AF to FCA and show that stable extensions are closed sets (called concept intents) in this translation with some special properties. We then modify two well-known algorithms from FCA to compute stable extensions. In Section 4, we present an evaluation of our algorithms on randomly generated AFs and provide a comparison with existing tools. In Section 5, we conclude with a summary and future work.

2 Preliminaries

2.1 Abstract Argumentation Frameworks

We recall some basic notions from abstract argumentation frameworks as introduced in [10]. An AF is a directed graph $F = (A, R)$, where A is a finite set of *arguments* and $R \subseteq A \times A$ is the *attack relation*. An edge $(a, b) \in R$ denotes that the argument a *attacks* the argument b (in the AF F). A set of arguments $S \subseteq A$ *attacks* b if there is $a \in S$ such that $(a, b) \in R$, and b *attacks* S if $(b, a) \in R$ for some $a \in S$. We say that $S \subseteq A$ *defends* $a \in A$ if every argument attacking a is attacked by S.

Figure 1 gives an example of an argumentation framework over arguments $A = \{a, b, c, d, e\}$. Here, for example, a attacks b and c; the set $\{b, c\}$ attacks a and d; the argument d attacks the set $\{a, e\}$ and, in fact, every set containing c or e; and $\{a, e\}$ defends d, since it attacks both its attackers, c and e.

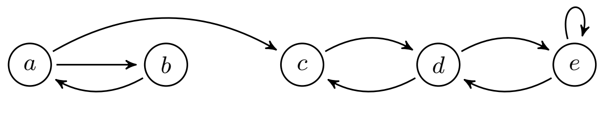

Fig. 1. Example of an argumentation framework.

Given an AF $F = (A, R)$, a set $S \subseteq A$ is said to be *conflict-free* (in F) if S does not attack any of its elements. We denote the set of conflict-free subsets of A as $cf(F)$. That is, $cf(F) = \{S \subseteq A \mid \forall a, b \in S : (a, b) \notin R\}$.

Several different types of semantics expressing different properties of sets of arguments have been considered in the literature [10,3]. We introduce only those of them that are relevant for our work.

Let $F = (A, R)$ be an argumentation framework and $S \in cf(F)$. Then S is called

- an *admissible extension* if every $a \in S$ is defended by S;
- a *preferred extension* if it is a maximal (w.r.t. set inclusion) admissible extension;
- a *stable extension* if S attacks every $a \in A \setminus S$.

Since a stable extension S attacks all other elements including all those that attack S, every stable extension is also a preferred extension.

The preferred extensions of the AF from Figure 1 are $\{a, d\}$, $\{b, c\}$, and $\{b, d\}$, and its stable extensions are $\{a, d\}$ and $\{b, d\}$.

Several interesting decision, counting, and enumeration problems in abstract argumentation have been considered in the literature [11,4,21]. Here we list only two of them that are relevant for us. For an AF $F = (A, R)$, an argument $a \in A$, and a semantic σ:

- Find a σ-extension of F if there is one.
- Enumerate all σ-extensions of F.

It is known that these problems are intractable for many of the interesting semantics [11,21]. Existing approaches typically solve these problems by reducing them to other formalisms such as constraint-satisfaction problem (CSP), propositional logic, or answer-set programming, and benefit from highly optimized solvers developed for these formalisms. To name a few, μ-toksia [27] encodes these problems as the Boolean satisfiability problem and makes use of a SAT-solver; pyglaf [1] reduces these problems to circumscription and employs an existing solver for circumscription; and ConArg [5] reduces them to constraints and uses a CSP-solver.

2.2 Formal Concept Analysis

Formal Concept Analysis [15] is a field of mathematics used for identifying clusters in data and for building a hierarchy of these clusters with tools originating from lattice theory. It has found application in several domains including biology [18], data mining [26], information retrieval [19], knowledge processing [31], and machine learning [24,9,6,30].

We will introduce only those notions and results from FCA that are relevant for our purposes. In FCA, one represents data using a *formal context* specifying which objects have which attributes. A formal context is usually denoted by $\mathbb{K} = (G, M, I)$, where G is a set of *objects*, M is a set of *attributes*, and $I \subseteq G \times M$

is an *incidence relation* between the objects and the attributes. A finite context can be visualized as a cross table, where the rows represent the objects and the columns represent the attributes of the context. A cross in column m of row g means that the object g has the attribute m, and the absence of a cross means that g does not have the attribute m.

For a set of objects $A \subseteq G$, the *derivation operator* $(\cdot)^{\uparrow}$ applied to A produces the set of attributes that are common to all objects in A:

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

Similarly, for a set of attributes $B \subseteq M$, the *derivation operator* $(\cdot)^{\downarrow}$ applied to B yields the set of objects that have all attributes in B:

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

We sometimes abuse the notation and write $x^{\uparrow}$ (resp. $x^{\downarrow}$) instead of $\{x\}^{\uparrow}$ (resp. $\{x\}^{\downarrow}$) for an object (resp. attribute) x.

Proposition 1. *For $A_1 \subseteq A_2 \subseteq G$ (resp. $B_1 \subseteq B_2 \subseteq M$), it holds that*

- $A_2^{\uparrow} \subseteq A_1^{\uparrow}$ *(resp. $B_2^{\downarrow} \subseteq B_1^{\downarrow}$);*
- $A_1 \subseteq A_1^{\uparrow\downarrow}$ *and* $A_1^{\uparrow} = A_1^{\uparrow\downarrow\uparrow}$ *(resp. $B_1 \subseteq B_1^{\downarrow\uparrow}$ and $B_1^{\downarrow} = B_1^{\downarrow\uparrow\downarrow}$).*

As a consequence of this, the combined operator $(\cdot)^{\uparrow\downarrow}$ is a *closure operator* on G and $(\cdot)^{\downarrow\uparrow}$ is a closure operator on M. Using these closure operators, one can describe "natural clusters" in data, which are called formal concepts. A *formal concept* of $\mathbb{K} = (G, M, I)$ is a pair (A, B), where $A \subseteq G$ and $B \subseteq M$, such that $A^{\uparrow} = B$ and $B^{\downarrow} = A$. A is called the *extent*, and B is called the *intent* of the formal concept (A, B).

When ordered w.r.t. subset inclusion of their extents (or, equivalently, w.r.t. inverse inclusion of their intents), formal concepts yield a complete lattice called the *concept lattice* of $\mathbb{K}$. The concept lattice obtained from a dataset allows an intuitive visualization of the data and enables domain experts to spot dependencies between clusters in the data.

For $A \subseteq G$, the set $A^{\uparrow}$ is the intent of some formal concept, since $(A^{\uparrow\downarrow}, A^{\uparrow})$ is always a formal concept. $A^{\uparrow\downarrow}$ is the smallest extent containing A. Consequently, a set $A \subseteq G$ is an extent if and only if $A = A^{\uparrow\downarrow}$. The same applies to intents. The intersection of any number of extents (respectively, intents) is always an extent (intent). Hence, the set of all extents forms a closure system on G, and the set of all intents forms a closure system on M [15].

It is well known that the set of all formal concepts of a context can be exponential in the size of the context and determining the number of formal concepts is #P-complete [23]. There are several algorithms for enumerating formal concepts [28,12,13,7,22,17,29,32], some of which do this with a polynomial delay [20]. For an analysis and evaluation of such algorithms, see [25].

3 An FCA Characterization of AF Semantics

We consider semantics of AFs from the viewpoint of FCA and make use of algorithms developed there for solving the abovementioned problems. To this purpose, we formulate argumentation frameworks as formal contexts, following the connection first noted in [2].

Definition 1. *Let* (A, R) *be an argumentation framework. The* induced formal context *of* (A, R) *is* $\mathbb{K}(A, R) = (A, A, (A \times A) \setminus R)$.

Note that such induced contexts are special in that their sets of objects and attributes coincide, which is not the case in general. Figure 2 shows the induced formal context of the argumentation framework from Figure 1.

$\mathbb{K}(A,R)$	a	b	c	d	e
a	×			×	×
b		×	×	×	×
c	×	×	×		×
d	×	×		×	
e	×	×	×		

Fig. 2. The induced formal context of the argumentation framework in Figure 1. Its concept intents are $\varnothing$, $\{e\}$, $\{d\}$, $\{d, e\}$, $\{b\}$, $\{b, d\}$, $\{b, c\}$, $\{b, c, e\}$, $\{b, c, d, e\}$, $\{a\}$, $\{a, e\}$, $\{a, d\}$, $\{a, d, e\}$, $\{a, b\}$, $\{a, b, d\}$, $\{a, b, c\}$, $\{a, b, c, e\}$, and $\{a, b, c, d, e\}$.

Following the definitions from Section 2.2, an application of the derivation operators of $\mathbb{K}(A, R)$ to $S \subseteq A$ yields the following sets:

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

and

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

Obviously, for $a, b \in A$, a attacks b if and only if $b \notin a^{\uparrow}$ or, equivalently, $a \notin b^{\downarrow}$. More generally, for $S \subseteq A$, the set $S^{\uparrow}$ consists of arguments not attacked by S, and $S^{\downarrow}$ is the set of arguments that do not attack S.

For example, in the context shown in Figure 2, we have $\{a, b\}^{\uparrow} = \{d, e\}$ and $\{a, b\}^{\downarrow} = \{c, d, e\}$. Indeed, d and e are the only arguments attacked neither by a nor by b in the AF from Figure 1, while c, d, and e are the only arguments that attack neither a nor b.

Proposition 2. *Let* (A, R) *be an AF. A set* $S \subseteq A$ *defends* $c \in A$ *if and only if* $S^{\uparrow} \subseteq c^{\downarrow}$ *holds in* $\mathbb{K}(A, R)$.

Proof. By definition, $S^{\uparrow} \subseteq c^{\downarrow}$ reads as follows: every $a \in A$ not attacked by S does not attack c, which is equivalent to S defending c.

More formally, let $b, c \in A$ and $(b, c) \in R$. Then, by definition, $b \notin c^{\downarrow}$. Assuming $S^{\uparrow} \subseteq c^{\downarrow}$, we have $b \notin S^{\uparrow}$ and S attacks b. Thus, S attacks all attackers of c, or, in other words, S defends c.

Conversely, suppose that S defends c. Take $b \in S^{\uparrow}$, i.e., some b not attacked by S. Since S defends c, we have $(b, c) \notin R$, which is equivalent to $b \in c^{\downarrow}$. Consequently, $S^{\uparrow} \subseteq c^{\downarrow}$ holds in $\mathbb{K}(A, R)$. $\square$

Proposition 3. *Given an AF (A, R), a set $S \subseteq A$ is conflict-free if and only if $S \subseteq S^{\uparrow}$ (or, equivalently, $S \subseteq S^{\downarrow}$) holds in $\mathbb{K}(A, R)$. S is a maximal conflict-free set if and only if $S = S^{\uparrow} \cap S^{\downarrow} \cap \{a \in A \mid a \in a^{\downarrow}\}$.*

Proof. The first statement holds by definition. To prove the second one, assume that S is a conflict-free set and there is some $b \in (S^{\uparrow} \cap S^{\downarrow} \cap \{a \in A \mid a \in a^{\downarrow}\}) \setminus S$. It holds that $b \in S^{\uparrow}$, $b \in S^{\downarrow}$, and b does not attack itself. Then $S \cup \{b\}$ is also conflict-free, and S cannot be a maximal conflict-free set. Conversely, if $S = S^{\uparrow} \cap S^{\downarrow} \cap \{a \in A \mid a \in a^{\downarrow}\}$, then, for every $b \notin S$, either $b \notin S^{\uparrow}$ and b is attacked by S, or $b \notin S^{\downarrow}$ and b attacks S, or $b \notin b^{\downarrow}$ and b attacks itself. In none of these case, $S \cup \{b\}$ is conflict-free. Hence, S is a maximal conflict-free set.

Proposition 4. *Given an AF (A, R), a set $S \subseteq A$ is an admissible extension if and only if $S \subseteq S^{\uparrow} \subseteq S^{\downarrow}$. A preferred (i.e., maximal admissible) extension S is always a concept intent of $\mathbb{K}(A, R)$, i.e., $S = S^{\downarrow\uparrow}$.*

Proof. The first statement follows from Propositions 3 and 2. In particular, since $S^{\downarrow} = \bigcap\{c^{\downarrow} \mid c \in S\}$, Proposition 2 implies that $S^{\uparrow} \subseteq S^{\downarrow}$ is equivalent to S defending all its elements.

To prove the second statement, assume that S is admissible. Then it holds that $S \subseteq S^{\uparrow} \subseteq S^{\downarrow}$. Due to Proposition 1, this implies $S^{\downarrow\uparrow} \subseteq S^{\uparrow}$. Now, suppose that $S \neq S^{\downarrow\uparrow}$, and take any $b \in (S^{\downarrow\uparrow} \setminus S)$. It holds that $b \in S^{\uparrow}$ and thus $b \in S^{\downarrow}$, i.e., S and b do not attack each other. Additionally, $b \in S^{\downarrow} = S^{\downarrow\uparrow\downarrow} \subseteq b^{\downarrow}$, the last inclusion holding due to $\{b\} \subseteq S^{\downarrow\uparrow}$ and Proposition 1. Therefore, b does not attack itself and $S \cup \{b\}$ is conflict-free.

To see that S defends b, take an $a \in A$ that attacks b, i.e., $b \notin a^{\uparrow}$. If S does not attack a, then $a \in S^{\uparrow} \subseteq S^{\downarrow}$. Hence, $S^{\downarrow\uparrow} \subseteq a^{\uparrow}$ and $b \in a^{\uparrow}$, which is a contradiction. Thus, S attacks a.

We have shown that S does not attack, is not attacked by, and defends every element of $S^{\downarrow\uparrow}$. No such element a can attack another such element b, since, otherwise, a would have been attacked by S, which defends b. Therefore, if S is an admissible set, then so is $S^{\downarrow\uparrow}$. This means that preferred extensions are concept intents (the reverse is not necessarily true). $\square$

Indeed, the three preferred extensions of the argumentation framework in Figure 1, $\{a, d\}$, $\{b, c\}$, and $\{b, d\}$, are among concept intents of the induced context from Figure 2. Since every stable extension S is also a preferred extension, from Proposition 4, we have

Corollary 1. *Given an AF (A, R), a stable extension $S \subseteq A$ is always a concept intent of $\mathbb{K}(A, R)$, i.e., $S = S^{\downarrow\uparrow}$.*

It turns out that we can give a compact and precise characterization of stable extensions in terms of derivation operators of the induced formal context. Note that $S = S^{\uparrow}$ reads as follows: "The set of elements not attacked by S is equal to S," which is the definition of a stable extension. Thus, we have

Proposition 5. *Given an AF (A, R), a set $S \subseteq A$ is a stable extension if and only if $S = S^{\uparrow}$.*

It is easy to see that no proper subset of a stable extension can be a stable extension; therefore, stable extensions form an antichain (a subset of incomparable elements) in the lattice of concept intents.

3.1 Enumerating Stable Extensions using Next-Closure Algorithm

Proposition 5 suggests that we can enumerate stable extensions of an AF (A, R) by computing concept intents of the induced context $\mathbb{K}(A, R)$ and outputting only those intents S for which $S = S^{\uparrow}$. One well-known way of enumerating concept intents in FCA is enumerating them in a so-called lectic order, which helps avoiding multiple computation of the same intent. The lectic order is defined as follows:

Definition 2. *Let $\mathbb{K} = (G, M, I)$ be a formal context. Fix some some linear order $m_1 < m_2 < \cdots < m_n$ on the set of attributes $M = \{m_1, \ldots, m_n\}$. This order induces a linear order on the power set of M, called the* lectic order*, which we also denote by $<$. For $m_i \in M$ and $A, B \subseteq M$, we define:*

$$A <_i B \quad \text{iff} \quad m_i \in B,\ m_i \notin A \text{ and } \forall j < i\ (m_j \in A \Leftrightarrow m_j \in B).$$

The order $<$ is the union of these orders $<_i$, i.e.,

$$A < B \quad \text{iff} \quad A <_i B \text{ for some } i \in M.$$

Note this is the same order as the one obtained when we map sets to binary numbers in a standard way, so that attribute m_i adds 2^{n-1} to the number. Obviously, $<$ extends the strict subset order, and thus $\varnothing$ is the smallest and M is the largest set w.r.t. $<$. The following proposition from [12,14] shows how to compute the lectically next concept intent set for a given set.

Proposition 6. *Given a formal context $\mathbb{K} = (G, M, I)$ and a set $A \subsetneq M$, the smallest concept intent greater than A w.r.t. the lectic order is*

$$((A \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})^{\downarrow\uparrow},$$

where j is the maximal attribute satisfying $A <_j ((A \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})^{\downarrow\uparrow}$.

In order to enumerate all concept intents of $\mathbb{K}$, one starts with the lectically smallest intent $\varnothing^{\downarrow\uparrow}$ and applies the proposition successively until the lectically largest intent M is reached. This algorithm, known as the *next-closure algorithm*, enumerates all concept intents of a given context with polynomial delay [12], and, although the proposition may suggest otherwise, the algorithm is quite easy to understand and implement.

Roughly speaking, the algorithm follows a computation tree, where a node corresponds to an intent and a child is formed by adding an attribute to the intent, computing the closure of the resulting set, and keeping it only if it does not contain an attribute smaller than the one that was added. Along every branch of the tree, attributes are added in the increasing order; that is, if an intent S results from adding an attribute m_i to its parent, its children are formed by adding only attributes m_j, where $j > i$, to S. For a more detailed explanation and analysis of the algorithm, see [14].

To enumerate stable extensions of an argumentation framework, we can compute the concept intents of the induced context with the next-closure algorithm, cutting a computation branch as soon as we reach an intent that is not conflict-free. Algorithm 1 enumerates all stable extensions of a given AF using this idea.

Algorithm 1 All Stable Extensions(A, R)

```
Input: Argumentation framework $(A, R)$ with $A = \{a_1, \dots a_n\}$.
Output: Stable extensions of $(A, R)$ in the lectic order.
1: Construct the induced context $\mathbb{K}(A, R)$
2: Fix a total order $a_1 < a_2 < \dots < a_n$ on $A$
3: $S = \varnothing^{\downarrow\uparrow}$                              {lectically smallest intent}
4: while $S \neq A$ do
5:     if $S = S^{\uparrow}$ then                          {Check if $S$ is a stable extension}
6:         output $S$
7:     $S :=$ Next-Conflict-Free-Intent$(\mathbb{K}(A, R),\ S)$
```

Termination of Algorithm 1 is guaranteed since A is finite. It is correct due to Proposition 5. Termination and correctness of Algorithm 2 is guaranteed due to Proposition 6 [14]. The only modification we have done is checking for conflicts in line 6 of Algorithm 2, which does not influence termination and correctness.

Algorithm 1 computes all conflict-free concept intents $S \subseteq A$ of the induced context $\mathbb{K}(A, R)$ with polynomial delay. However, only those that are stable extensions are output. Between two stable extensions there can potentially be exponentially many concept intents S that do not satisfy the criteria $S = S^{\uparrow}$ and, hence, are not stable extensions. The runtime of our algorithm heavily depends on the number of concept intents of $\mathbb{K}(A, R)$, which can be exponential in the size of $\mathbb{K}(A, R)$. It is known that determining the number of concept intents is #P-complete [23]. It was shown in [21] that enumerating stable extensions is not output-polynomial unless P = NP; thus we cannot expect to enumerate them efficiently. On a positive side, it is easy to see that the memory requirements of the algorithm depend only linearly on the number of arguments, since it needs to store a constant number of sets and maintain a constant number of indices.

Algorithm 2 Next Conflict Free Intent($\mathbb{K}(A,R), S$)

```
Input: Induced context $\mathbb{K}(A,R)$ with a total order $a_1 < a_2 < \ldots < a_n$ on $A$ and a set $S \subseteq A$.
Output: Lectically next conflict-free intent of $\mathbb{K}$ coming after $S$.
1: for $a_i := a_n$ to $a_i := a_1$ do                                   {iterate in reverse order}
2:     if $a_i \in S$ then
3:         remove $a_i$ from $S$
4:     else
5:         $T := (S \cup \{a_i\})^{\downarrow\uparrow}$
6:         if $T \subseteq T^{\downarrow}$ then                                  {check for conflict}
7:             if $a_j \notin (T \setminus S)$ holds for every $a_j < a_i$ then   {lectic-order check}
8:                 return $T$                          {lectically next conflict-free intent}
```

3.2 Norris-based Algorithm for Stable Extensions

Next, we present an adaptation of an algorithm that was originally developed by E.M. Norris for computing the maximal rectangles in a binary relation [28]. This algorithm, being used in the FCA community for enumerating concept intents, has proven to be fast for different types of formal contexts in practice [25]. Similar to the next-closure algorithm, it uses the lectic-order check to avoid multiple generation of the same intent. Unlike next-closure, it keeps a list of candidates from which intents are incrementally computed, which makes closure computation more efficient.

Algorithm 3 is an adaptation of this approach. It iteratively processes subcontexts $(A, B, (A \times B) \cap R)$, where $B \subseteq A$, of the context $\mathbb{K}(A, R)$, starting with $B = \varnothing$ and adding one argument at a time in an arbitrary order. The algorithm maintains a list of 4-tuples of the form $(S^{\downarrow}, S, S^{\uparrow}, S^{\downarrow} \cap S^{\uparrow})$, where S is potentially a subset of a stable extension and the derivation operators are taken with respect to the current subcontext.

Algorithm 3 Incremental Stable Extensions(A, R)

```
Input: Argumentation framework $(A, R)$.
Output: Stable extensions of $(A, R)$ yielded by the Add subrprocedure.
1: $B := \varnothing$
2: $\mathcal{C} := \{(A, \varnothing, A, A)\}$
3: for all $a \in A$ do
4:     Add($(A, R), B, a, \mathcal{C}$)                                  {Add modifies $\mathcal{C}$}
5:     $B := B \cup \{a\}$
```

When processing an argument a, the algorithm attempts to extend every set S on the list with a if this does not cause a conflict (see line 1 of Algorithm 4). Two cases are possible then. If arguments not attacking S do not attack a either, then every stable extension containing S must contain a, and the algorithm simply updates the components of the tuple corresponding to S (lines 2–11). Otherwise (lines 12–22), the algorithm generates a new tuple corresponding to $S \cup \{a\}$ and, unless it cannot be further extended without introducing conflicts, the algorithm adds the new tuple to the list (lines 21–22). This new tuple is not further processed in the current call to Algorithm 4. Note also the lectic-order check in line 14. In both cases, if $S \cup \{a\}$ turns out to be a stable extension, it is reported as such (lines 6 and 18) and the corresponding tuple is removed from (or not added to) the list since, in this case, supersets of $S \cup \{a\}$ cannot be stable extensions.

Algorithm 4 Add$((A, R), B, a, \mathcal{C})$

```
Input: Argumentation framework $(A,R)$, $B \subseteq A$, $a \in A \setminus B$, and set $\mathcal{C}$,
    which, at the point of the call, must be equal to
        $\{(S^{\downarrow}, S, S^{\uparrow}, S^{\downarrow} \cap S^{\uparrow}) \mid S^{\downarrow\uparrow} \cap B = S \subseteq B$ and $S \subsetneq S^{\downarrow} \cap S^{\uparrow}\}$.
Output: Stable extensions $S \subseteq B \cup \{a\}$ of $(A,R)$ containing $a$ and updated $\mathcal{C}$.
 1: for all $(X,S,Y,Z) \in \mathcal{C}$ such that $a \in Z$ do        {$S \cup \{a\}$ is conflict-free}
 2:     if $X \subseteq a^{\downarrow}$ then        {arguments not attacking $S$ do not attack $a$}
 3:         $S := S \cup \{a\}$        {update the components of the existing tuple}
 4:         $Y := Y \cap a^{\uparrow}$
 5:         if $Y = S$ then        {$S$ attacks everything but itself}
 6:             output $S$
 7:             remove $(X,S,Y,Z)$ from $\mathcal{C}$
 8:         else
 9:             $Z := Z \cap a^{\uparrow}$
10:             if $Z = S$ then
11:                 remove $(X,S,Y,Z)$ from $\mathcal{C}$
12:     else
13:         $U := X \cap a^{\downarrow}$        {arguments not attacking $S \cup \{a\}$}
14:         if $U \subseteq b^{\downarrow}$ for no $b \in B \setminus S$ then        {$(S \cup \{a\})^{\downarrow\uparrow} \cap B = S \cup \{a\}$}
15:             $T := S \cup \{a\}$
16:             $V := Y \cap a^{\uparrow}$        {arguments not attacked by $T$}
17:             if $T = V$ then        {$T$ attacks everything but itself}
18:                 output $T$
19:             else
20:                 $W := U \cap V$
21:                 if $W \neq T$ then        {$T$ can be extended}
22:                     add $(U,T,V,W)$ to $\mathcal{C}$
```

Termination of Algorithm 4 is guaranteed, since $\mathcal{C}$ is finite. Termination of Algorithm 3 is guaranteed, since Algorithm 4 is invoked exactly once for every $a \in A$.

Algorithm 4 differs from the original algorithm for computing concept intents in several aspects. It stores additional information in the last two components of tuples in list $\mathcal{C}$; however, if memory is an issue, this information can be recomputed from the second component and the original framework. In line 1, it is checked if adding a to S causes a conflict. In line 5, it is checked whether S is a stable extension; in this case, it is removed from the list, since its supersets cannot be stable extensions. Finally, a new tuple generated after line 12 is added to the list only if the corresponding intent is not a stable extension (which is checked in line 17) and there remain arguments that can be added to it without causing conflicts (which is checked in line 21). Apart from these changes, the algorithm acts as the original algorithm. Therefore, Algorithm 3 generates all conflict-free intents, which guarantees its correctness.

3.3 Preferred Extensions

With some effort, the Algorithms 1 and 3 can be adapted to other semantics. In this subsection, we outline a possible adaptation to preferred semantics.

As shown in Proposition 4, every preferred (i.e., maximal admissible) extension is a concept intent. Each of the two presented algorithms implicitly builds a tree of intents, from small to large, cutting a branch as soon as it stumbles upon a maximal conflict-free set, which may or may not be a stable extension. While doing so, it necessarily generates all preferred extensions. To identify them among generated intents, one can check admissibility for every generated intent and keep track of the largest generated admissible extension on each branch. If a branch terminates with a stable extension, it is the only preferred extension on this branch. The other preferred extensions are among admissible extensions that are either terminal nodes in the tree or intermediate nodes with no admissible extensions among descendants. Let us call nodes satisfying this condition *preferred candidates*. They have to be checked for subset-maximality (unless they are stable).

If the goal is to compute a single preferred extension, the maximality check can be avoided. Traversing the tree of intents in the left-to-right depth-first-order, we can be sure that the first preferred candidate S is, in fact, a maximal admissible extension. This is so, because all intents containing S as a subset are either among its descendants in the tree or precede S in this order. Neither of the two presented algorithms follows this left-to-right depth-first-order when computing intents; so, some care should taken when implementing this trick with them. Alternatively, it is possible to modify the algorithms so that they follow this order (resulting in an algorithm similar to what is known as Close by One in the FCA community [22]), and then the first preferred candidate as computed by the algorithm will be a preferred extension.

If the goal is to enumerate all preferred extensions, we do need to check for maximality. A simple way to do this is as follows: we start with an empty list $\mathcal{L}$ of potentially preferred extensions and, as soon as we obtain a new preferred candidate S, we check if it is a subset of any set from $\mathcal{L}$ (in which case we ignore S; otherwise, we add S to $\mathcal{L}$) or a superset of some sets from $\mathcal{L}$ (in which case, these sets are removed from $\mathcal{L}$). Upon the termination of the algorithm, $\mathcal{L}$ will be the set of all preferred extensions.

4 Experimental results

Most of the available tools for argumentation frameworks do no support enumerating all stable extensions, but only allow finding a single extension. The latter problem is called "SE-ST" at the ICCMA competitions, and, in this section, we evaluate the performance of our algorithms on this problem. This means that we terminate our algorithms as soon as they produce the first extension or allow them to run to completion if the AF contains no stable extensions.

We have evaluated the runtime performance of Algorithms 1 and 3 from Section 3 on randomly generated AFs with different sizes and densities. By the size of an AF (A, R), we mean the number of arguments of the AF, i.e., $|A|$. By its density, we mean the proportion $|R|/|A|^2$. We have generated test frameworks containing 200, 600, 800, 1k, 2k, 3k, 4k, 5k, 6k, 7k, 8k, 9k, and 10k arguments, each with densities ranging from 0.1 to 0.9. Thus we have generated altogether 117 test files.[3] While generating the test frameworks, we did not allow arguments to attack themselves. Our implementation[4] supports the new input format introduced for ICCMA 2023. The tools used for the comparison accept input in the `apx`-format. Therefore, the test data contains each framework in both formats.

The tests were performed under Ubuntu Linux on a hardware with a 32 core-CPU running at 2.9 GHz and 256 GB of main memory. For comparing our approach with the existing ones, we have evaluated three other tools from the ICCMA competitions with the same test frameworks. These are μ-toksia[5], pyglaf[6], and a-folio-dpdb[7]. As time limit, we fixed five minutes for all the five approaches. We did not put any constraints on their memory usage.

The results of the experiments on frameworks with densities from 0.2 to 0.9 are shown in Figure 3, which refers to Algorithm 1 as `affca-nc` and to Algorithm 3 as `affca-norris`. For density 0.1, all approaches failed with a timeout on almost all framework sizes. In the diagrams, discontinued lines are due to timeouts for some framework sizes. For instance, for density 0.9, pyglaf terminated within the time limit for frameworks of sizes 200, 600, 2k, 3k, 4k, and 5k but not for frameworks of sizes 800, 1k, and greater than 5k. Therefore, there is a gap in the corresponding line between the framework sizes 600 and 2k.

The results of the evaluation show that Algorithm 3 performs significantly better than all other four approaches on test frameworks with density of 0.5 and above (the exceptions being the 4k-framework with density 0.6, and the 2k-framework with density 0.5, which only μ-toksia was able to process within the time limit). The performance difference is most visible on frameworks with large densities. For instance, for density 0.9 and size 9k, Algorithm 3 is almost ten times faster than μ-toksia and two times faster than Algorithm 1; the other

[3] The test frameworks are available via the GitHub Repository of the project.
[4] `https://github.com/sertkaya/afca`
[5] `https://bitbucket.org/andreasniskanen/mu-toksia/src/master/`
[6] `https://alviano.com/software/pyglaf`
[7] `https://github.com/gorczyca/dp_on_dbs`

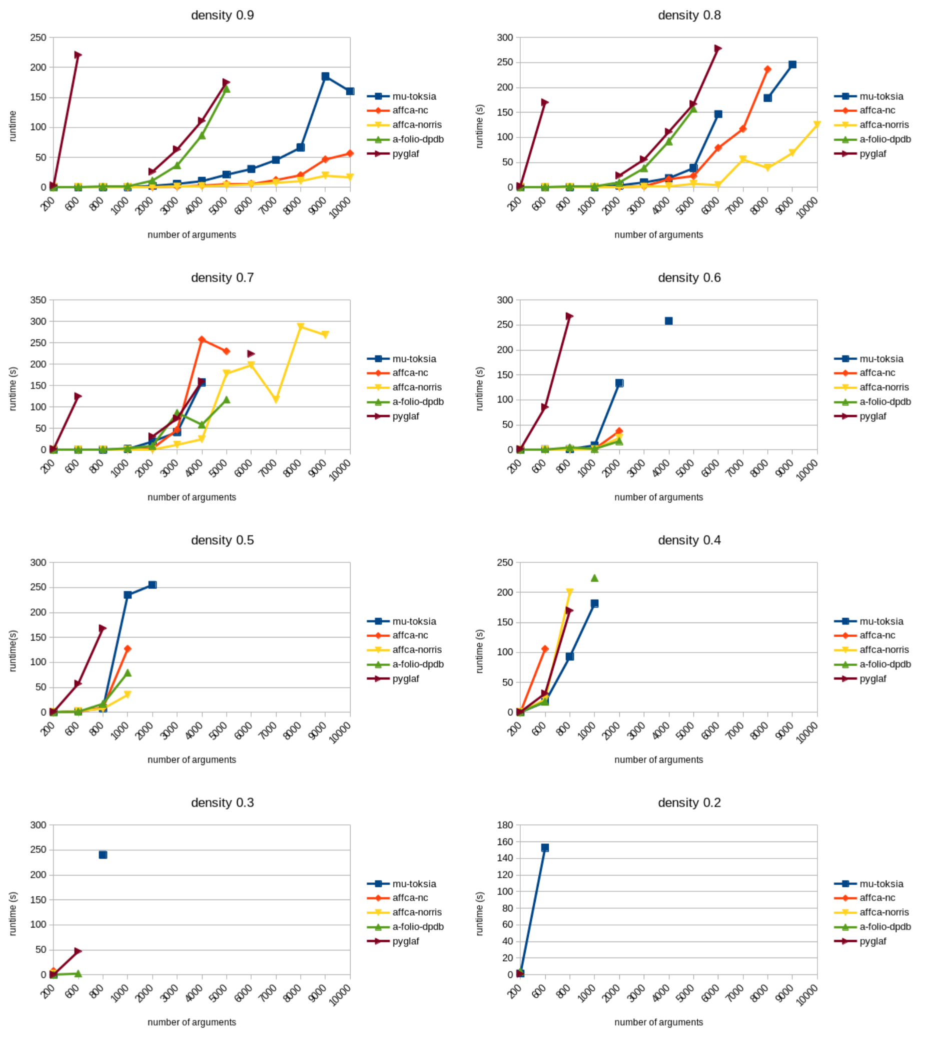

Fig. 3. Experimental results

tools get timeout. As the density of the frameworks decreases, the performance of our algorithms deteriorates. For density 0.3 and below, they terminate within the time limit only for the framework with the smallest number of arguments, namely, 200 arguments. For such test frameworks, μ-toksia is the only tool that still terminates within the time limit.

The reason why our algorithms perform better on AFs with denser attack relations is, in fact, clear: for such AFs, the induced context representing the not-attack relation is sparser, and the number of concepts in a sparse context is usually small.

5 Conclusion and Future Work

We have presented a characterization of AFs as formal contexts and adapted two algorithms from FCA for computing stable extensions of AFs. Experimental results with randomly generated test data show that our approach performs significantly better than the existing approaches for AFs with dense attack relations. Our Algorithm 1, based on the Next Closure algorithm, has the advantage that its memory requirements depend linearly on the number of arguments and do not depend on any other parameters of the argumentation framework. The other one, Algorithm 3, can more efficiently prune the search tree and, therefore, often has the best performance among all approaches we have evaluated. However, it has the disadvantage that it stores all stable-extension candidates and, because of this, has a high memory requirement, in the worst case, exponential in the number of arguments.

We plan to improve the algorithms so that they could skip larger number of intents when searching for stable extensions and, in particular, prune larger parts of the search tree that will not lead to stable extensions. Various heuristics could be used to speed up the computation [16]. One heuristic that could work for both algorithms when run on frameworks with self-attacking arguments would be to fix the linear order on arguments so that such arguments (which cannot be part of any stable extension) are easily skipped. Another option would be to fix an order where arguments that are more likely to appear in stable extensions are used first. These can be, for instance, arguments that are attacked by a small number of arguments but attack a large number of other arguments. This heuristic can especially be useful for the problem of finding a single stable extension.

Acknowledgements This work is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project 389792660 (TRR 248, Center for Perspicuous Systems), by the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) in the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), and by BMBF and DAAD (German Academic Exchange Service) in project 57616814 (SECAI, School of Embedded Composite AI).

References

1. Alviano, M.: The pyglaf argumentation reasoner (ICCMA2021). CoRR **abs/2109.03162** (2021), `https://arxiv.org/abs/2109.03162`
2. Amgoud, L., Prade, H.: A formal concept view of abstract argumentation. In: van der Gaag, L.C. (ed.) Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 12th European Conference, ECSQARU 2013, Utrecht, The Netherlands, July 8-10, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7958, pp. 1–12. Springer (2013). `https://doi.org/10.1007/978-3-642-39091-3_1`, `https://doi.org/10.1007/978-3-642-39091-3_1`
3. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowl. Eng. Rev. **26**(4), 365–410 (2011). `https://doi.org/10.1017/S0269888911000166`, `https://doi.org/10.1017/S0269888911000166`
4. Baroni, P., Dunne, P.E., Giacomin, M.: On extension counting problems in argumentation frameworks. In: Baroni, P., Cerutti, F., Giacomin, M., Simari, G.R. (eds.) Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010. Frontiers in Artificial Intelligence and Applications, vol. 216, pp. 63–74. IOS Press (2010). `https://doi.org/10.3233/978-1-60750-619-5-63`, `https://doi.org/10.3233/978-1-60750-619-5-63`
5. Bistarelli, S., Santini, F.: 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. pp. 605–612. IEEE Computer Society (2011). `https://doi.org/10.1109/ICTAI.2011.96`, `https://doi.org/10.1109/ICTAI.2011.96`
6. Borchman, D., Hanika, T., Obiedkov, S.: Probably approximately correct learning of Horn envelopes from queries. Discrete Applied Mathematics **273**, 30–42 (2020), `https://doi.org/10.1016/j.dam.2019.02.036`
7. Bordat, J.P.: Calcul pratique du treillis de Galois d' une correspondance. Mathématiques, Informatique et Sciences Humaines **96**, 31–47 (1986)
8. Cerutti, F., Gaggl, S.A., Thimm, M., Wallner, J.P.: Foundations of implementations for formal argumentation. FLAP **4**(8) (2017), `http://www.collegepublications.co.uk/downloads/ifcolog00017.pdf`
9. Cimiano, P., Hotho, A., Staab, S.: Learning concept hierarchies from text corpora using formal concept analysis. Journal of Artificial Intelligence Research **24**(1), 305–339 (2005)
10. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–358 (1995). `https://doi.org/10.1016/0004-3702(94)00041-X`, `https://doi.org/10.1016/0004-3702(94)00041-X`
11. Dunne, P.E., Wooldridge, M.J.: Complexity of abstract argumentation. In: Simari, G.R., Rahwan, I. (eds.) Argumentation in Artificial Intelligence, pp. 85–104. Springer (2009). `https://doi.org/10.1007/978-0-387-98197-0_5`, `https://doi.org/10.1007/978-0-387-98197-0_5`
12. Ganter, B.: Two basic algorithms in concept analysis. Tech. Rep. Preprint-Nr. 831, Technische Hochschule Darmstadt, Darmstadt, Germany (1984)
13. Ganter, B.: Two basic algorithms in concept analysis. In: Kwuida, L., Sertkaya, B. (eds.) Proceedings of the 8th International Conference on Formal Concept Analysis, (ICFCA 2010). Lecture Notes in Artificial Intelligence, vol. 5986, pp. 329–359. Springer-Verlag (2010), reprint of [12]

14. Ganter, B., Obiedkov, S.: Conceptual Exploration. Springer (2016). https://doi.org/10.1007/978-3-662-49291-8, https://doi.org/10.1007/978-3-662-49291-8
15. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag, Berlin, Germany (1999)
16. Geilen, N., Thimm, M.: Heureka: A general heuristic backtracking solver for abstract argumentation. In: Black, E., Modgil, S., Oren, N. (eds.) Theory and Applications of Formal Argumentation - 4th International Workshop, TAFA 2017, Melbourne, VIC, Australia, August 19-20, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10757, pp. 143–149. Springer (2017). https://doi.org/10.1007/978-3-319-75553-3_10, https://doi.org/10.1007/978-3-319-75553-3_10
17. Godin, R., Missaoui, R., Alaoui, H.: Incremental concept formation algorithms based on Galois (concept) lattices. Computational Intelligence **11**(2), 246–267 (1995)
18. Grissa, D., Comte, B., Pétéra, M., Pujos-Guillot, E., Napoli, A.: A hybrid and exploratory approach to knowledge discovery in metabolomic data. Discrete Applied Mathematics **273**, 103–116 (2020). https://doi.org/https://doi.org/10.1016/j.dam.2018.11.025, advances in Formal Concept Analysis: Traces of CLA 2016
19. Ignatov, D.I.: Introduction to formal concept analysis and its applications in information retrieval and related fields. CoRR **abs/1703.02819** (2017), http://arxiv.org/abs/1703.02819
20. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Information Processing Letters **27**(3), 119–123 (1988)
21. Kröll, M., Pichler, R., Woltran, S.: 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. pp. 1145–1152. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/159, https://doi.org/10.24963/ijcai.2017/159
22. Kuznetsov, S.O.: A fast algorithm for computing all intersections of objects in a finite semi-lattice. Automatic Documentation and Mathematical Linguistics **27**(5), 11–21 (1993)
23. Kuznetsov, S.O.: On computing the size of a lattice and related decision problems. Order **18**(4), 313–321 (2001). https://doi.org/10.1023/A:1013970520933, https://doi.org/10.1023/A:1013970520933
24. Kuznetsov, S.O.: Machine learning and formal concept analysis. In: Eklund, P.W. (ed.) Concept Lattices, Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia, February 23-26, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2961, pp. 287–312. Springer (2004)
25. Kuznetsov, S.O., Obiedkov, S.: Comparing performance of algorithms for generating concept lattices. Journal of Experimental and Theoretical Artificial Intelligence **14**(2-3), 189–216 (2002)
26. Lakhal, L., Stumme, G.: Efficient mining of association rules based on formal concept analysis. In: Ganter, B., Stumme, G., Wille, R. (eds.) Formal Concept Analysis, Foundations and Applications. Lecture Notes in Computer Science, vol. 3626, pp. 180–195. Springer (2005)
27. Niskanen, A., Järvisalo, M.: μ-toksia: An efficient abstract argumentation reasoner. In: Calvanese, D., Erdem, E., Thielscher, M. (eds.) Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning,

KR 2020, Rhodes, Greece, September 12-18, 2020. pp. 800–804 (2020). `https://doi.org/10.24963/kr.2020/82`, `https://doi.org/10.24963/kr.2020/82`

28. Norris, E.M.: An algorithm for computing the maximal rectangles in a binary relation. Revue Roumaine de Mathématiques Pures et Appliquées **23**(2), 243–250 (1978)
29. Nourine, L., Raynaud, O.: A fast algorithm for building lattices. Information Processing Letters **71**(5-6), 199–204 (1999)
30. Obiedkov, S.: Learning implications from data and from queries. In: Cristea, D., Ber, F.L., Sertkaya, B. (eds.) Proceedings of the 15th International Conference on Formal Concept Analysis, (ICFCA 2019). Lecture Notes in Artificial Intelligence (2019)
31. Poelmans, J., Ignatov, D.I., Kuznetsov, S.O., Dedene, G.: Formal concept analysis in knowledge processing: A survey on applications. Expert Syst. Appl. **40**(16), 6538–6560 (2013)
32. Valtchev, P., Missaoui, R.: Building concept (Galois) lattices from parts: Generalizing the incremental methods. In: Delugach, H.S., Stumme, G. (eds.) Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001). Lecture Notes in Computer Science, vol. 2120, pp. 290–303. Springer-Verlag, Stanford, CA, USA (2001)