

AFCA: Searching for Complete Extensions via Enumeration of a Closure System

Sergei Obiedkov

Faculty of Computer Science / cfaed / ScaDS.AI

TU Dresden

Dresden, Germany

Email: sergei.obiedkov@tu-dresden.de

0000-0003-1497-4001

Bariş Sertkaya

Faculty of Computer Science and Engineering

Frankfurt University of Applied Sciences

Frankfurt am Main, Germany

Email: sertkaya@fb2.fra-uas.de

0000-0002-4196-0150

Abstract—AFCA is a solver for abstract argumentation that operates by enumerating closed sets. Unlike most other solvers, it does not rely on SAT or CSP solvers. Instead, AFCA adapts algorithms from formal concept analysis used for enumerating closed sets of a closure system. While the system supports several semantics, this paper focuses on the algorithm used for credulous acceptance under complete semantics. The approach is based on enumerating subsets of arguments that are *semicomplete*—that is, conflict-free sets containing all the arguments they defend. We show that semicomplete sets form a closure system, and we leverage this property to design a targeted enumeration algorithm. In contrast to generic enumeration, our method reduces the search space by dynamically selecting relevant defenders and pruning infeasible branches.

Index Terms—Argumentation frameworks, complete extensions, closure systems.

I. INTRODUCTION

An *argumentation framework* 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* [1]. 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 $R^{-1}(\{a\})$.

We say that $S \subseteq A$ *defends* $a \in A$ if every argument attacking a is attacked by S , i.e., $R^{-1}(a) \subseteq R(S)$. A set S is *self-defending* if it defends all its elements, i.e., $R^{-1}(S) \subseteq R(S)$.

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*. An admissible set that contains every argument it defends is called a *complete extension* of F .

We present an algorithm for the so-called DC-CO problem: Given an argumentation framework $F = (A, R)$ and an argument $c \in A$, decide if F has a complete extension containing c and, if so, to compute such an extension. Our approach is based on enumerating closure systems.

II. CLOSURE SYSTEMS AND COMPLETE EXTENSIONS

A family \mathcal{F} of subsets of a ground set A is called a *closure system* if \mathcal{F} 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) minimal closed superset.

A number of algorithms for enumerating closure systems have been developed, notably, in formal concept analysis [2]–[4]. Some of them have been adapted for enumerating other structures in argumentation frameworks, in particular, stable and preferred extensions [5] and self-defending and admissible sets [6].

To make use of such algorithms for complete extensions, we first relax the requirements associated with them. We call $S \subseteq A$ *semicomplete* if S is conflict-free and contains every argument it defends. The only difference from complete extensions is that semicomplete sets are not required to be self-defending.

Proposition II.1. *Let $F = (A, R)$ be an argumentation framework. The semicomplete subsets of A , together with A , form a closure system.*

Proof. It suffices to show that semicomplete sets are closed under intersection. Let S and T be semicomplete 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 semicomplete, both sets contain a . It follows that $S \cap T$ contains every argument it defends and is therefore semicomplete. ■

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

III. SOLVING THE DC-CO PROBLEM BY ENUMERATING SEMICOMplete SETS

There is a closure operator that maps a subset of A to its smallest superset in the closure system from Proposition II.1. We assume that $\text{semicomplete}(F, S)$ is a procedure computing the closure of a conflict-free set S under this operator, except that it returns \perp instead of A when no semicomplete superset of S exists. This can happen, for example, if S defends one of its attackers. The procedure $\text{semicomplete}(F, S)$ can be implemented in a straightforward way: while there

exists $a \in A \setminus S$ such that $R^{-1}(a) \subseteq R(S)$, add a to S if a does not attack S and return \perp otherwise.

To compute a complete extension containing argument c , we start by checking if c attacks itself. If so, then no complete extension with c exists. Otherwise, we form the minimal semicomplete set S_0 containing c :

$$S_0 := \text{semicomplete}(F, \{c\})$$

and then enumerate all its semicomplete supersets until we find one that is also self-defending and thus complete. Enumeration is performed with the `complete` procedure from Fig. 1.

Input: An argumentation framework $F = (A, R)$ and argument sets $S, P \subseteq A$, such that S is semicomplete in F

Output: A complete extension $X \supseteq S$ of F with $X \cap P = \emptyset$ if it exists; \perp otherwise

```

1: if  $R^{-1}(S) \subseteq R(S)$  then           { $S$  is self-defending}
2:   return  $S$ 
3: choose  $a \in R^{-1}(S) \setminus R(S)$     {must defend against  $a$ }
4: for all  $d \in R^{-1}(a) \setminus P$  do      { $d$  attacks  $a$ }
5:    $T := \text{semicomplete}(F, S \cup \{d\})$ 
6:   if  $T \neq \perp$  and  $T \cap P = \emptyset$  then
7:      $X := \text{complete}(F, T, P \cup R(d) \cup R^{-1}(d))$ 
8:     if  $X \neq \perp$  then
9:       return  $X$ 
10:   $P := P \cup \{d\}$ .
11: return  $\perp$ 

```

Fig. 1. Procedure `complete`(F, S, P)

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

$$\text{complete}(F, S_0, R(S_0) \cup R^{-1}(S_0) \cup \{a \in A \mid aRa\}).$$

In lines 1–2 of the algorithm in Fig. 1, we check if S is self-defending, in which case the algorithm terminates by returning it. Otherwise, we find an attacker a of S that is not attacked by S (line 3) and try to defend S against it by including one of the attackers of a . We consider only those attackers of a that are not in P (line 4).

For each attacker d of a , we tentatively add d to S and extend the result to its minimal semicomplete superset T (line 5). If such T exists and consists entirely of arguments outside P , we attempt to expand it to a complete extension by recursively calling the `complete` procedure. For this call, we extend P with arguments conflicting with d (line 7). If the call succeeds by producing a complete extension, we return it (line 9). If, on the contrary, adding d to S does not allow us to obtain a complete extension, d is added to P (line 10) so as to be able to prune dead-end branches (in line 6) while considering alternative defenders of S .

If adding none of the attackers of a leads to a complete extension, it is not possible to defend S against a . There is

then no need to try to defend S against its other attackers, so the algorithm returns \perp (line 11).

There may be different strategies for choosing the attacker of S against which to defend S first (line 3). In our implementation, we choose one that is attacked by the smallest number of arguments outside P ; in other words, we minimize the number of arguments to be explored as potential defenders against a (line 4).

The algorithm in Fig. 1 roughly follows the strategy of the `Close by One` algorithm for enumerating closed sets of a closure operator [7]. However, it differs from it in some important aspects (apart from necessary adjustments to the context of abstract argumentation). Thus, `Close by One` employs a fixed order on elements of the ground set and always follows that order when trying to augment the current set. In our algorithm, we dynamically select the argument d to be added to the current set S depending on the arguments against which S must still be defended. `Close by One` attempts adding every possible element, whereas our algorithm considers only the attackers of a single attacker of the current set. This greatly reduces the search space, especially, in relatively sparse frameworks.

IV. IMPLEMENTATION

We implemented the presented algorithm within the AFCA toolkit, which also includes algorithms for several other computational tasks in abstract argumentation (e.g., enumeration of stable and preferred extensions). For clarity reasons, we presented a recursive version of the algorithm; however, the AFCA implementation is non-recursive.

AFCA is implemented in the C programming language. It does not require any external libraries or solvers. The source code is available on GitHub.¹

REFERENCES

- [1] P. M. Dung, “On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games,” *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995.
- [2] B. Ganter and R. Wille, *Formal Concept Analysis – Mathematical Foundations*. Springer Cham, 2 ed., 2024.
- [3] S. O. Kuznetsov and S. Obiedkov, “Comparing performance of algorithms for generating concept lattices,” *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 14, no. 2-3, pp. 189–216, 2002.
- [4] B. Ganter and S. Obiedkov, *Conceptual Exploration*. Springer, 2016.
- [5] S. Obiedkov and B. Sertkaya, “Computing stable extensions of argumentation frameworks using formal concept analysis,” in *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings* (S. A. Gaggl, M. V. Martinez, and M. Ortiz, eds.), vol. 14281 of *Lecture Notes in Computer Science*, pp. 176–191, Springer, 2023.
- [6] M. Elaroussi, L. Nourine, and M. S. Radjef, “Lattice point of view for argumentation framework,” *Ann. Math. Artif. Intell.*, vol. 91, no. 5, pp. 691–711, 2023.
- [7] S. O. Kuznetsov, “A fast algorithm for computing all intersections of objects from an arbitrary semilattice,” *Nauchno-Tekhnicheskaya Informatsiya Seriya 2-Informatsionnye Protssessy i Sistemy*, no. 1, pp. 17–20, 1993.

¹<https://github.com/sertkaya/afca>.