A Distributed and Clustering-Based Algorithm for the Enumeration Problem in Abstract Argumentation

Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex(✉)

IRIT, UT1-UT3, Toulouse, France
{doutre,mickael.lafages,lagasq}@irit.fr

Abstract. Computing acceptability semantics of abstract argumenta-tion frameworks is receiving increasing attention. Large-scale instances, with a clustered structure, have shown particularly difficult to compute. This paper presents a distributed algorithm, *AFDivider*, that enumer-ates the acceptable sets under several labelling-based semantics. This algorithm starts with cutting the argumentation framework into clusters thanks to a spectral clustering method, before computing simultaneously in each cluster parts of the labellings. This algorithm is proven to be sound and complete for the stable, complete and preferred semantics, and empirical results are presented.

Keywords: Abstract argumentation · Algorithms · Clustering · Enumeration

1 Introduction

Argumentation is a reasoning model which has been of application in multi-agent systems for years (see [16] for an overview). The development of argumentation techniques and of their computation drives such applications.

Among the various argumentation models, the one that is considered in this paper has been defined by Dung [23]: an abstract argumentation framework (AF) considers arguments as abstract entities, and focuses on their attack relationships, hence representing arguments and their underlying conflicts by a directed graph. Which arguments can be accepted is defined by [23] as a collective notion, by a semantics: a set of arguments is collectively acceptable under the semantics. Four semantics (*grounded*, *stable*, *complete* and *preferred*) were defined by Dung, and a variety of other semantics have followed (see [6] for an overview). Several enrichments of the argumentation framework have also been proposed (*e.g.* [7,17]).

Supported by the ANR-11-LABEX-0040-CIMI project of the CIMI International Centre for Mathematics and Computer Science in Toulouse.

https://doi.org/10.1007/978-3-030-33792-6_6

The enumeration of all the acceptable sets of an AF under a given semantics is a problem that has received a lot of attention (see [21] for an overview). This problem has been shown to be computationally intractable for some of the above-mentioned semantics [25]. A competition, ICCMA, that compares argumentation solvers on their ability to solve this problem (and other decision problems) was created a few years ago.[1] The last editions of this competition have been analyzed: [12,34] highlight that some AF instances have been particularly hard to solve, and that others were not solved at all, considering the *preferred* semantics notably. Many of these instances are of Barabási–Albert (BA) type [1], which is a structure found in several large-scale natural and human-made systems, such as the World Wide Web and some social networks [4]. More generally, these hard graphs are non-dense, but contain parts which are dense:[2] such graphs have a *clustered structure*.

Recent algorithms, proposed for an efficient enumeration of the acceptable sets, are based on a cutting of the AF [18,24,27], along with, for some of them, the use of distributed, parallel computation in each part, to construct the acceptable sets [19]. In this research line, our paper presents a new "cutting and distributed computing" algorithm, called *AFDivider*, for the enumeration of the acceptable sets of an AF, under the *stable*, *preferred* and *complete* labelling semantics. The cutting of the AF is done in a new way, using spectral clustering methods. Compared to the existing approaches, the added value of *AFDivider* is its way to split the AF and thus to distribute the solving hardness of the whole AF into smaller parts, the reunifying process requiring less checks than the construction of the labellings over the whole AF. *AFDivider* is shown to be sound and complete. The algorithm has been empirically tested, and the results have been compared to those of two solvers of the ICCMA 2017 edition, *Pyglaf* [3] and *ArgSemSAT* [20].

The paper starts with presenting the background of this work (Sect. 2), before describing the algorithm (Sect. 3). Soundness and completeness of the algorithm are proven in Sect. 4. A preliminary empirical analysis is conducted (Sect. 5). Related works are presented in Sect. 6. Perspectives for future work are then opened.

2 Background

2.1 Abstract Argumentation

According to [23], an abstract argumentation framework consists of a set of arguments and of a binary attack relation between them.

[1] International Competition on Computational Models of Argumentation (ICCMA) http://argumentationcompetition.org/.

[2] The density in an argumentation graph is the ratio "number of existing attacks" over "number of potential attacks" (this last number is equal to n^2 with n being the number of arguments).

Definition 1 (AF). *An* argumentation framework *(AF) is a pair $\Gamma = \langle A, R \rangle$ where A is a finite*[3] *set of abstract arguments and $R \subseteq A \times A$ is a binary relation on A, called the attack relation: $(a, b) \in R$ means that a attacks b.*

Hence, an argumentation framework can be represented by a directed graph with arguments as vertices and attacks as edges. Figure 1 shows an example of an AF.

Acceptability semantics can be defined in terms of labellings [6,15].

Definition 2 (Labelling). *Let $\Gamma = \langle A, R \rangle$ be an AF, and $S \subseteq A$. A* labelling *of S is a total function $\ell : S \to \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$. The set of all labellings of S is denoted as $\mathscr{L}(S)$. A labelling of Γ is a labelling of A. The set of all labellings of Γ is denoted as $\mathscr{L}(\Gamma)$.*

We write $\mathtt{in}(\ell)$ for $\{a | \ell(a) = \mathtt{in}\}$, $\mathtt{out}(\ell)$ for $\{a | \ell(a) = \mathtt{out}\}$ and $\mathtt{und}(\ell)$ for $\{a | \ell(a) = \mathtt{und}\}$.

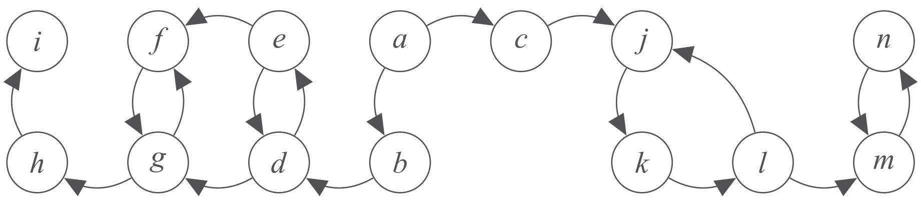


Fig. 1. Example of an argumentation framework AF.

Definition 3 (Legally labelled arguments, valid labelling). *An $\mathtt{in}$-labelled argument is said to be* legally $\mathtt{in}$ *iff all its attackers are labelled $\mathtt{out}$. An $\mathtt{out}$-labelled argument is said to be* legally $\mathtt{out}$ *iff at least one of its attackers is labelled $\mathtt{in}$. An $\mathtt{und}$-labelled argument is said to be* legally $\mathtt{und}$ *iff it does not have any attacker that is labelled $\mathtt{in}$ and one of its attackers is not labelled $\mathtt{out}$. A* valid labelling *is a labelling in which all arguments are legally labelled.*

Let $\Gamma = \langle A, R \rangle$ be an AF, and $\ell \in \mathscr{L}(\Gamma)$ be a labelling. Different kinds of labelling can be defined:

Definition 4 (Admissible, complete, grounded, preferred and stable labellings). *ℓ is an* admissible labelling *of Γ iff for any argument $a \in A$ such that $\ell(a) = \mathtt{in}$ or $\ell(a) = \mathtt{out}$, a is legally labelled. ℓ is a* complete labelling *of Γ iff for any argument $a \in A$, a is legally labelled. ℓ is the* grounded labelling *of Γ iff it is the complete labelling of Γ that minimizes (w.r.t $\subseteq$) the set of $\mathtt{in}$-labelled arguments. ℓ is a* preferred labelling *of Γ iff it is a complete labelling of Γ that maximizes (w.r.t $\subseteq$) the set of $\mathtt{in}$-labelled arguments. ℓ is a* stable labelling *of Γ iff it is a complete labelling of Γ which has no $\mathtt{und}$-labelled argument.*

[3] According to [23], the set of arguments is not necessarily finite. Nevertheless, in this paper, it is reasonable to assume that it is finite.

Note that each complete labelling includes the grounded labelling. This property will be used by the algorithm presented in Sect. 3 in order to compute the AF labellings in a distributed way. Let $\Gamma = \langle A, R \rangle$ be an AF, and $\mathscr{L}(\Gamma)$ be its set of labellings, semantics can be defined.

Definition 5 (Semantics). *A* semantics σ *is a total function* σ *that associates to* Γ *a subset of* $\mathscr{L}(\Gamma)$*. The set of labellings under semantics* σ*, with* σ *being either the complete (co), the grounded (gr), the stable (st) or the preferred (pr) semantics, is denoted by* $\mathscr{L}_\sigma(\Gamma)$*. A labelling* ℓ *is a* σ*-labelling iff* $\ell \in \mathscr{L}_\sigma(\Gamma)$*.*

Example 1. Let us consider the AF given in Fig. 1. Table 1 shows the labellings corresponding to the different semantics (the other possible labellings are not given). Note that this AF has no stable labelling.

2.2 Clustering Methods

A *cluster* in a graph can be defined as a connected subgraph. Finding clusters is a subject that has been widely studied (see [31,35]). The clustering approach implemented in our algorithm is based on a spectral analysis of a defined similarity matrix of the graph. We chose this clustering method as it is well suited for a non-dense graph (see Sects. 3.1 and 3.2 for more explanation). We give here a succinct description of this approach (for details, see [36]):

Table 1. Labellings of the *AF* of Fig. 1 under the grounded, complete, preferred and stable semantics.

	arguments														σ			
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	gr	co	pr	st
ℓ_1	in	out	out	out	in	out	in	out	in	und	und	und	out	in		×	×	
ℓ_2	in	out	out	in	out	in	out	in	out	und	und	und	out	in		×	×	
ℓ_3	in	out	out	out	in	out	in	out	in	und	und	und	und	und		×		
ℓ_4	in	out	out	in	out	in	out	in	out	und	und	und	und	und		×		
ℓ_5	in	out	out	und	und	und	und	und	und	und	und	und	und	und	×	×		
ℓ_6	in	out	out	und	und	und	und	und	und	und	und	und	out	in		×		

- Computation of a similarity matrix of the graph. In this squared matrix, the values represent how much two nodes are similar according to a given similarity criterion, and the rows may be seen as the coordinates of the graph nodes in a similarity space.
- Computation of the Laplacian matrix of this similarity matrix. The rows of this Laplacian matrix represent how much a node is similar to the others and how much each of its neighbours contributes to its global similarity with its neighbourhood.
- Computation of the eigenvectors (see [33]) of the Laplacian matrix with their associated eigenvalues.

- These eigenvalues are sorted by increasing order. A number n of them is kept with their associated eigenvectors.[4]
- A matrix whose columns are the remaining eigenvectors is built. Its rows represent the new node coordinates in a space that maximizes the proximity between similar nodes. In that space, the euclidean distance between two nodes shows how much a node is similar to another.
- Then a simple algorithm of clustering such as *KMeans* is applied to that new data set, seeking for a partition into n parts, based on the coordinates of the nodes (see [30] for more information about *KMeans* algorithm).

An illustration of this method on the running example is given in Sect. 3.2 while the similarity criterion used is explicited in Sect. 3.1.

3 The Algorithm

This section presents the *AFDivider* algorithm designed for the *complete*, *stable* and *preferred* semantics (denoted by σ). It computes the semantics labellings of an AF by first removing trivial parts of the AF (the *grounded* labelling, as done in [18]), then cutting the AF into clusters and computing simultaneously in each cluster labelling parts, before finally reunifying compatible parts to get the σ-labellings of the whole AF. Each of these steps will be presented and then illustrated on the running example.

Algorithm 1: *AFDivider* algorithm.

```
Data: Let $\Gamma = \langle A, R \rangle$ be an AF and $\sigma$ be a semantics
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\Gamma)}$: the set of the $\sigma$-labellings of $\Gamma$
1 $\ell_{gr} \leftarrow ComputeGroundedLabelling(\Gamma)$
2 $CCSet \leftarrow SplitConnectedComponents(\Gamma, \ell_{gr})$
3 for all $\gamma_i \in CCSet$ do in parallel
4     $ClustSet \leftarrow ComputeClusters(\gamma_i)$
5     $\mathscr{L}_\sigma^{\gamma_i} \leftarrow ComputeCompLabs(\sigma, ClustSet)$
6 $\mathscr{L}_\sigma \leftarrow \varnothing$
7 if $\nexists \gamma_i \in CCSet$ s.t. $\mathscr{L}_\sigma^{\gamma_i} = \varnothing$ then $\mathscr{L}_\sigma \leftarrow \{\ell_{gr}\} \times \prod_{\gamma_i \in CCSet} \mathscr{L}_\sigma^{\gamma_i}$
8 return $\mathscr{L}_\sigma$
```

[4] Sorted in ascending order, the eigenvalue sequence represents how the similarity within clusters increases as the number of clusters grows. Obviously, the more clusters, the more homogeneous they will get, but also, the more cases to compute. A compromise between the number of clusters and homogeneity is needed. A heuristic (called "elbow heuristic") to find the appropriate number of dimensions to keep, consists in detecting the jump in the eigenvalues sequence.

3.1 Description

Given an argumentation framework $\Gamma = \langle A, R \rangle$, the *AFDivider* algorithm (Algorithm 1) starts with computing the grounded labelling of Γ (line 1). Indeed in each of the semantics σ we are interested in, the arguments labelled ***in*** or ***out*** in the grounded labelling are labelled in the same way in all the σ-labellings. It is a fixed part. Note that the function $ComputeGroundedLabelling(\Gamma)$ returns a partial labelling of Γ in which the arguments are labelled ***in*** or ***out***. The ***und***-labelled arguments according to the grounded semantics do not belong to ℓ_{gr}.

Γ is then split into disjoint sub-AFs obtained after removing the arguments labelled ***in*** or ***out*** in the grounded labelling (line 2). The $CCSet$ variable is the set of connected components computed.

Given that there is no relation between them, the labelling computation of those connected components can be made in a simultaneous way (line 3) according to the chosen semantics.

For each of these connected components, a clustering is made (line 4) using the spectral clustering method presented in Sect. 2.2. The similarity matrix on which the spectral analysis relies is a kind of adjacency matrix where the directionality of edges is omitted and where the matrix values are the number of edges between two arguments. Basically, the more an argument will be related to another, the more similar the two arguments will be considered.

This similarity criterion is particularly relevant for non-dense graphs with a clustered structure. Indeed, it produces sparse matrices and as a consequence the eigenvector equation system to solve will be simplified as there will be many zero values. This is what motivated our choice for the spectral clustering method.

After this clustering process, *ComputeCompLabs* (Algorithm 2) is called to compute in a distributed way all the labellings of the connected component according to σ (line 5).

Finally, given that ℓ_{gr} is a fixed part of all σ-labellings of Γ and that all the connected components are completely independent, to construct the σ-labellings of the whole AF, a simple Cartesian product is made (line 7) between the labellings of all the components and the grounded one.

If one of the components has no labelling then the whole AF has no labelling (so $\mathscr{L}_\sigma = \varnothing$).

Consider now Algorithm 2 that computes the component labellings in a distributed way, relying on the clustering made. The σ-labellings of each cluster are computed simultaneously (line 1). Unlike the case of connected components used in Algorithm 1, there exist attacks between clusters. In order to compute all the possible σ-labellings of a given cluster, every case concerning its inward attacks (attacks whose target is in the current cluster but the source is from another cluster) have to be considered. Given that the sources of an inward attack could be labelled ***in***, ***out*** or ***und*** in their own cluster, the σ-labellings of the current cluster have to be computed for all the labelling combinations of inward attack sources.

Algorithm 2: *ComputeCompLabs* algorithm.

```
Data: Let $ClustSet$ be a set of cluster structures for a component $\gamma$, $\sigma$ be a
      semantics
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\gamma)}$: the set of the $\sigma$-labellings of $\gamma$
1 for all $\kappa_j \in ClustSet$ do in parallel $\mathscr{L}_\sigma^{\kappa_j} \leftarrow ComputeClustLabs(\sigma, \kappa_j)$
2 $\mathscr{L}_\sigma \leftarrow ReunifyCompLabs(\bigcup_{\kappa_j \in ClustSet} \mathscr{L}_\sigma^{\kappa_j}, ClustSet)$
3 if $\sigma = pr$ then $\mathscr{L}_\sigma \leftarrow \{\ell | \ell \in \mathscr{L}_\sigma \text{ s.t. } \nexists \ell' \in \mathscr{L}_\sigma \text{ s.t. } \mathit{in}(\ell) \subset \mathit{in}(\ell')\}$
4 return $\mathscr{L}_\sigma$
```

Note that having "well shaped" clusters (*i.e.* clusters with few inter cluster attacks) reduces considerably the number of cases to compute, as there are few edges cut. Thus this algorithm is well suited for clustered non-dense graphs.

Once that, for all clusters, the *ComputeClustLabs* function has computed the σ-labellings for all the possible cases (this is done by calling any sound and complete procedure computing the semantics labellings), the *ReunifyCompLabs* function is called in order to reunify compatible labelling parts. Labelling parts are said to be compatible together when all the targets of the inter cluster attacks are legally labelled in the resulting reunified labelling.

A special step has to be done for the *preferred* semantics as this reunifying process does not ensure the maximality (w.r.t $\subseteq$) of the set of in-labelled arguments (so not all of the labellings produced in line 2 are *preferred* ones). A maximality check is done (line 3) in order to keep only the wanted labellings.

Note that, when computing the *stable* semantics, the set of labellings $\mathscr{L}_\sigma$ returned by the function *ReunifyCompLabs* may be empty. It happens when one of the component clusters has no *stable* labelling.

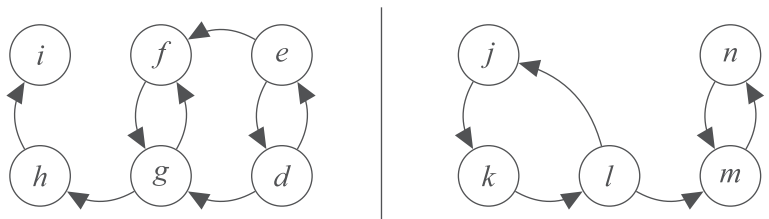
3.2 An Illustrating Example

In this section, the behaviour of our algorithms is illustrated on the AF given in Fig. 1 for the *preferred* semantics, as it is the most complex semantics of the three targeted ones.

The first step consists in computing the grounded labelling in order to eventually split the AF into sub-AFs. The grounded labelling of the AF restricted only to the in-labelled and out-labelled arguments is: $\ell_{gr} = \{(a, \mathit{in}), (b, \mathit{out}), (c, \mathit{out})\}$.

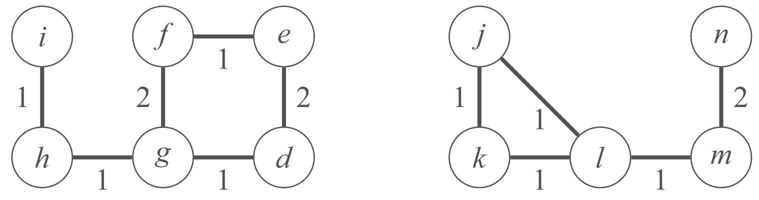
Removing arguments a, b and c from the AF produces two connected components, as illustrated in Fig. 2.

Then simultaneously γ_1 and γ_2 are clustered using the spectral clustering method This is done by several steps. First, we consider the similarity matrices of γ_1 and γ_2 according to our criterion, *i.e.* the number of attacks between arguments. They may also be seen as the adjacency matrices of the weighted non-directed graphs obtained from γ_1 and γ_2 (see Fig. 3). Given that the AF relation density is low, the matrices are rather sparse.



(a) Component 1: γ_1. (b) Component 2: γ_2.

Fig. 2. Connected components resulting from the grounded removal pre-processing.



(a) Component 1: γ_1. (b) Component 2: γ_2.

$$M_a^{\gamma_1} = \begin{array}{c|cccccc|} & d & e & f & g & h & i \\ d & \mathbf{0} & 2 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} \\ e & 2 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ f & \mathbf{0} & 1 & \mathbf{0} & 2 & \mathbf{0} & \mathbf{0} \\ g & 1 & \mathbf{0} & 2 & \mathbf{0} & 1 & \mathbf{0} \\ h & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 1 \\ i & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} \end{array}$$

(c) Similarity matrix of γ_1.

$$M_a^{\gamma_2} = \begin{array}{c|ccccc|} & j & k & l & m & n \\ j & \mathbf{0} & 1 & 1 & \mathbf{0} & \mathbf{0} \\ k & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} \\ l & 1 & 1 & \mathbf{0} & 1 & \mathbf{0} \\ m & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 2 \\ n & \mathbf{0} & \mathbf{0} & \mathbf{0} & 2 & \mathbf{0} \end{array}$$

(d) Similarity matrix of γ_2.

Fig. 3. Step 1 of the spectral clustering.

Once the AF similarity matrix is constructed, data are projected in a new space in which similarity is maximised. If a certain structure exists in the data set, we will see in that space some agglomerates appear, corresponding to the node clusters. To do this projection, we compute the n smallest eigenvalues[5] of the Laplacian matrix obtained from the similarity matrix and the vectors associated with them (this n is an arbitrary parameter; in this example we have chosen to keep all the vectors, *i.e.* $n = 5$). Indeed, the eigenvectors found will correspond to the basis of that similarity space and the eigenvalues to the variance on the corresponding axes. Given that we are looking for homogeneous groups, we will consider only the axis on which the variance is low, and so the eigenvectors that have small eigenvalues. The space whose basis is the n selected eigenvectors (corresponding to the n smallest eigenvalues) is then a compression of similarity space (*i.e.* we keep only the dimension useful for a clustering).

[5] There exist algorithms, such as *Krylov-Schur* method, able to compute eigenvectors from smallest to greatest eigenvalue and to stop at any wanted step (*e.g.* the number of vectors found). With such an algorithm it is not necessary to find all the solutions as we are interested only in the small eigenvalues.

Let us take as an example the case of γ_2. Its degree matrix $M_d^{\gamma_2}$ and its Laplacian matrix $M_l^{\gamma_2}$ are given in Fig. 4.

$$M_d^{\gamma_2} = \begin{array}{c|ccccc|} & j & k & l & m & n \\ j & 2 & 0 & 0 & 0 & 0 \\ k & 0 & 2 & 0 & 0 & 0 \\ l & 0 & 0 & 3 & 0 & 0 \\ m & 0 & 0 & 0 & 3 & 0 \\ n & 0 & 0 & 0 & 0 & 2 \end{array}$$

(a) Degree matrix of γ_2.

$$M_d^{\gamma_2} - M_a^{\gamma_2} = M_l^{\gamma_2} = \begin{array}{c|ccccc|} & j & k & l & m & n \\ j & 2 & -1 & -1 & 0 & 0 \\ k & -1 & 2 & -1 & 0 & 0 \\ l & -1 & -1 & 3 & -1 & 0 \\ m & 0 & 0 & -1 & 3 & -2 \\ n & 0 & 0 & 0 & -2 & 2 \end{array}$$

(b) Laplacian matrix of γ_2.

Fig. 4. Step 2 of the spectral clustering for γ_2.

The eigenvalues of $M_l^{\gamma_2}$ sorted in ascending order are:

$$\begin{array}{ccccc} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \\ \left[2.476651 \times 10^{-16}\right. & 5.857864 \times 10^{-1} & 3.000000 & 3.414214 & \left.5.000000\right] \end{array}$$

and their associated eigenvectors are:

$$\begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \\ -0.4472136 & 0.4397326 & 7.071068 \times 10^{-1} & 0.3038906 & 0.1195229 \\ -0.4472136 & 0.4397326 & -7.071068 \times 10^{-1} & 0.3038906 & 0.1195229 \\ -0.4472136 & 0.1821432 & -5.551115 \times 10^{-17} & -0.7336569 & -0.4780914 \\ -0.4472136 & -0.4397326 & -2.775558 \times 10^{-16} & -0.3038906 & 0.7171372 \\ -0.4472136 & -0.6218758 & -1.665335 \times 10^{-16} & 0.4297663 & -0.4780914 \end{array}$$

Now that the similarity space is found, the following step is to find how many groups we have in that space. This number can be founded using the eigenvalue sequence sorted in ascending order and identifying in this sequence the "best elbow" (*i.e.* the point that corresponds to a quick growth of the variance, see Fig. 5). In our example, the number of clusters determined by that heuristic is 2.

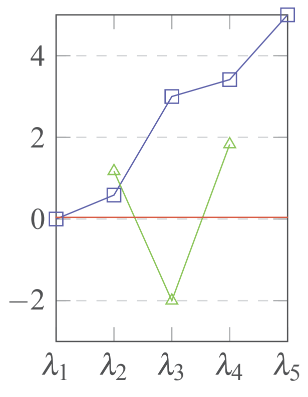


Fig. 5. Step 3 of the spectral clustering. (Color figure online)

To compute that "best elbow" we consider the second derivative (green line with triangles) of the ascending order sequence. As the second derivative represents the concavity of the eigenvalue sequence, we can take the first value of the second derivative above a certain threshold (red line without symbol) determined experimentally (*i.e.* the first position where the eigenvalue sequence is enough convex).

The first point of the second derivative, corresponding to the concavity formed by the first three eigenvalues, is the first value above the threshold; so we determine that the "best elbow" is in position 2.

Once the number of clusters is chosen, we must to find the partition of the set of arguments. This is done using a *KMeans* type algorithm [30][6] applied on the kept eigenvectors following the chosen number of clusters (see Fig. 6).

The matrix composed by the kept eigenvectors (the *two* first eigenvectors, 2 being the number of clusters):

$$\begin{array}{c|c} & \begin{array}{cc} v_1 & v_2 \end{array} \\ \begin{array}{c} j \\ k \\ l \\ m \\ n \end{array} & \begin{bmatrix} -0.4472136 & 0.4397326 \\ -0.4472136 & 0.4397326 \\ -0.4472136 & 0.1821432 \\ -0.4472136 & -0.4397326 \\ -0.4472136 & -0.6218758 \end{bmatrix} \end{array}$$

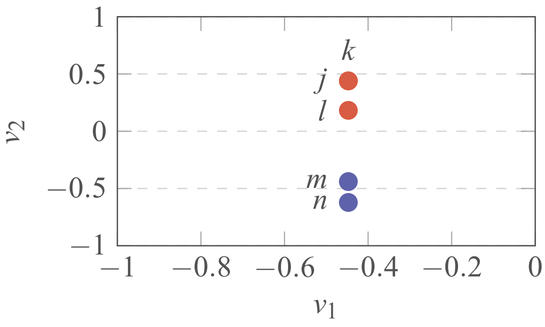


Fig. 6. Step 4 of the spectral clustering.

The lines of this matrix correspond to the coordinates of the nodes in the compressed similarity space. With a *KMeans* algorithm we can find groups of datapoint in that space and so have the partition of arguments we wanted (here $\{j, k, l\}$ and $\{m, n\}$).

The complete result given by the spectral clustering is shown in Fig. 7. κ_1 and κ_2 are the clusters determined from γ_1, and κ_3 and κ_4 are the ones from γ_2.

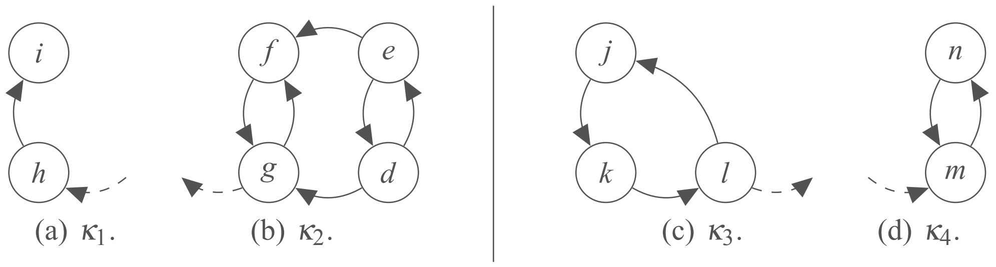


Fig. 7. Identified clusters.

After the clustering, the next step of our algorithm is the computation of *preferred* labellings. This computation is made simultaneously in the different clusters using an external solver (one of the best solvers identified in the ICCMA competition, see [34]). Recall that, for each cluster, every case concerning its inward attacks (attacks whose target is in the current cluster but the source is from another cluster) have to be considered. Given that the sources of an inward attack could be labelled ***in***, ***out*** or ***und*** in their own cluster, the σ-labellings of the current cluster have to be computed for all the labelling combinations of inward attack sources. For instance, for κ_1 (resp. κ_4), three cases for h and so for i (resp. for m and so for n) must be considered. Whereas for κ_2 and κ_3, there is no inward attack, the computed labellings only depend on the content of the cluster. The tables in Fig. 8(a) show the computed labelling parts for each cluster.

[6] Given n observations, a *KMeans* algorithm aims to partition the n observations into k subsets such that the distance between the elements inside each subset is minimized. Here we have $n = 5$ and $k = 2$.

Notice that although three cases are computed for κ_4, only two labellings are obtained. This is due to the maximality of the *preferred* semantics. Indeed, even though m is attacked by an ***und***-labelled argument, n may be labelled ***in*** as it defends itself against m. As a consequence, m would be labelled ***out***.

The last step of our algorithm is the reunifying phase (line 2, Algorithm 2). In this step, the constructed labellings are those in which all the target arguments are legally labelled. As an example, $\ell_1^{\kappa_1}$ cannot be reunified with $\ell_2^{\kappa_2}$ as h would be illegally ***out***-labelled. Figure 8(b) shows the valid reunified labellings for each component.

κ_1

	$\ell_1^{\kappa_1}$	$\ell_2^{\kappa_1}$	$\ell_3^{\kappa_1}$
h	out	in	und
i	in	out	und

κ_2

	$\ell_1^{\kappa_2}$	$\ell_2^{\kappa_2}$
d	out	in
e	in	out
f	out	in
g	in	out

κ_3

	$\ell_1^{\kappa_3}$
j	und
k	und
l	und

κ_4

	$\ell_1^{\kappa_4}$	$\ell_2^{\kappa_4}$
m	out	in
n	in	out

(a) Cluster labellings.

γ_1

	$\ell_1^{\gamma_1}$	$\ell_2^{\gamma_1}$
d	out	in
e	in	out
f	out	in
g	in	out
h	out	in
i	in	out

γ_2

	$\ell_1^{\gamma_2}$
j	und
k	und
l	und
m	out
n	in

(b) Component labellings.

Fig. 8. Labellings computed using our algorithm.

In that particular example all the reunified labellings are maximal w.r.t $\subseteq$ of the set of ***in***-labelled arguments, so the maximality check (line 3, Algorithm 2) does not change the set of labellings.[7]

Finally, the *preferred* labellings of the whole AF are constructed by performing a Cartesian product of the component labellings and of the grounded one. See the final computed *preferred* labellings in Table 1, Sect. 2.1 (labellings ℓ_1 and ℓ_2).

4 Soundness and Completeness

This section presents formal properties of *AFDivider*: soundness and completeness for the *complete*, the *stable* and the *preferred* semantics. Let σ be one of

[7] To highlight the necessity of the maximality check, let us take as minimal example the AF defined by $\langle\{a, b\}, \{(a, b), (b, a)\}\rangle$ and a partition of it in which each argument is in a different cluster. For each cluster, we will have three possible labellings as the inward attack source may be labelled ***in***, ***out*** or ***und*** in the other cluster. The reunifying phase will thus admit the labelling $\{(a, \textit{\textbf{und}}), (b, \textit{\textbf{und}})\}$ which is not a *preferred* labelling.

these three semantics. To be *sound* for σ means that the algorithm produces only σ-labellings. To be *complete* for σ means that the algorithm produces all the σ-labellings. In other words, given σ, *AFDivider* produces only and all the σ-labellings.

In order to prove these properties, we rely on the notions of *top-down* and *bottom-up* semantics decomposability introduced in [5] and then developed in [9]. In a few words, a semantics σ is said to be *top-down* decomposable if, for all AF Γ and for all its partitions into sub-AFs, the set of σ-labellings of Γ is included in the set of valid labellings obtained by reunifying the σ-labellings of its sub-AFs. A semantics σ is said to be *bottom-up* decomposable if, for all AF Γ and for all its partitions into sub-AFs, the set of valid labellings obtained by reunifying the σ-labellings of its sub-AFs is included in the set of σ-labellings of Γ. A semantics is said to be *fully* decomposable if it is *top-down* and *bottom-up* decomposable. These notions of *top-down* and *bottom-up* semantics decomposability can also be defined w.r.t. a specific type of partition. For instance, the partition selector denoted by $\mathscr{S}_{USCC}$ only produces partitions in which SCCs (Strongly Connected Components) are not split into different parts. In [9] it has been proven that:

- The *stable* and *complete* semantics are *fully* decomposable.
- The *preferred* semantics is *top-down* decomposable.
- The *preferred* semantics is *fully* decomposable w.r.t to $\mathscr{S}_{USCC}$.

Proposition 1. AFDivider *is sound and complete for the complete and the stable semantics.*

Sketch of proof. *Let σ be a fully decomposable semantics. Let $\Gamma = \langle A, R\rangle$ be an AF. Let ℓ_{gr} be the grounded labelling of Γ restricted to the* **in**-*labelled and* **out**-*labelled arguments. Let $\Omega = \{\omega_{gr}, \omega_1^1, \dots, \omega_{n_1}^1, \dots, \omega_1^k, \dots, \omega_{n_k}^k\}$ be a partition of A such that ω_{gr} is the set of arguments labelled in ℓ_{gr} and such that for all i and j, ω_i^j is the set of arguments corresponding to the cluster j of the component i determined by the component clustering performed by* AFDivider.

Given that for all clusters, the labellings are computed for all possible labellings of the cluster inward attack sources, and given that σ is fully decomposable, the set of valid reunified labellings produced by AFDivider *is equal to $\mathscr{L}_\sigma(\Gamma)$.*

And so AFDivider *is sound and complete for the* complete *and the* stable *semantics.* ■

Proposition 2. AFDivider *is sound and complete for the preferred semantics.*

Sketch of proof. *Let σ be the* preferred *semantics. Let $\Gamma = \langle A, R\rangle$ be an AF. Let ℓ_{gr} be the grounded labelling of Γ restricted to the* **in**-*labelled and* **out**-*labelled arguments. Let $\{\gamma_1, \dots, \gamma_k\}$ be the set of all connected components obtained by* AFDivider *after removing ℓ_{gr}. Let $\Omega = \{\omega_{gr}, \omega_1^1, \dots, \omega_{n_1}^1, \dots, \omega_1^k, \dots, \omega_{n_k}^k\}$ be a partition of A such that ω_{gr} is the set of arguments labelled in*

ℓ_{gr} *and such that for all* i *and* j, ω_i^j *is the set of arguments corresponding to the cluster* j *of the component* γ_i *determined by the component clustering performed by* AFDivider.

Given that the preferred *semantics is* top-down *decomposable, and given that for all clusters, the labellings are computed for all possible labellings of the cluster inward attack sources, then for each component* γ_i, $\mathscr{L}_{pr}(\gamma_i)$ *is included in the set of valid reunified labellings produced by the function* $ReunifyCompLabs$ *(Algorithm 2, line 2). The maximality check (line 3) makes Algorithm 2 sound and complete for the* preferred *semantics.*

Let $\Omega' = \{\omega_{gr}, \omega^1, \ldots, \omega^k\}$ *be a partition of* A *such that* ω_{gr} *is the set of arguments labelled in* ℓ_{gr} *and such that for all* i: $\omega^i = \bigcup_{j=1}^{n_i}(\omega_j^i)$. *Let* $S = \{(a,b) | \exists i \text{ s.t. } (a,b) \in (\omega_{gr} \times \omega^i) \cap R\}$ *be the set of all attacks going from an argument labelled in* ℓ_{gr} *to an argument non present in* ℓ_{gr}. *Note that all the sources of these attacks are* **`out`**-*labelled in* ℓ_{gr}. *Let* $\Gamma' = \langle A, R' \rangle$ *with* $R' = R \setminus S$, *be the AF obtained from* Γ *when removing the attacks in* S. *Given that the sources of attacks removed to obtained* Γ' *from* Γ *are all* **`out`**-*labelled arguments, we have* $\mathscr{L}_\sigma(\Gamma') = \mathscr{L}_\sigma(\Gamma)$. *Note that* $\Omega' \in \mathscr{S}_{USCC}(\Gamma')$. *Indeed, for all* i, $(\omega_{gr} \times \omega^i) \cap R' = \varnothing$ *and for all* $j \neq i$, $(\omega^j \times \omega^i) \cap R' = \varnothing$.

Given that the preferred *semantics is* fully *decomposable w.r.t.* $\mathscr{S}_{USCC}$ *then the set of valid labellings obtained by reunifying the* σ-*labellings of the sub-AFs corresponding to* Ω' *equal to* $\mathscr{L}_\sigma(\Gamma')$. *Given that Algorithm 2 is sound and complete for the* preferred *semantics, the Cartesian product made in Algorithm 1 (line 7) computes exactly* $\mathscr{L}_\sigma(\Gamma')$. *As a consequence, Algorithm 1 computes exactly* $\mathscr{L}_\sigma(\Gamma)$. AFDivider *is thus sound and complete for the* preferred *semantics.* ■

5 Experimental Results

In this section we present some experimental results conducted with the *AFDivider* algorithm. The experiments have been made on some hard instances of the ICCMA competition, which are mostly of Barabási–Albert (BA) type. They all are non-dense and have a clustered structure.

To compute the labellings of a cluster given a particular labelling of its inward attack sources, we have used an already existing solver called "*Pyglaf*", one of the best solvers at the ICCMA 2017 session, which transforms the AF labelling problem into a SAT problem [3]. In this paper, we compare our algorithm (using *Pyglaf*) with *Pyglaf* itself, and with *ArgSemSAT* [20], for the preferred, the complete and the stable semantics.

For each experiment, we used 6 cores of a Intel Xeon Gold 6136 processor, each core having a frequency of 3 GHz. The RAM size was 45GB. As at least two of the three used solvers are multithreaded (*Pyglaf* and *AFDivider*), we have chosen to compare them using both CPU and real time (the CPU time includes the user and the system times). Note that, for our algorithm, the durations cover both the clustering time and the computation of labellings time. The timeout has been set to 1 h for the real time.

Table 2. Experimental results (PR: preferred, CO: complete, ST: stable, MO : “Memory Overflow”, TO: “stop with TimeOut”, “−”: “missing data”). The time result format is “minutes: seconds. centiseconds”.

			i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
PR	Nb lab. (≈)		0.28×10^6	1.07×10^6	1.28×10^6	1.37×10^6	1.96×10^6	4.47×10^6	11.75×10^6	10.74×10^9
	AFDivider	end state	✓	✓	✓	✓	✓	✓	✓	MO
		CPU time	0:07.35	0:14.05	0:19.89	0:31.39	0:28.01	0:46.27	12:15.16	
		real time	0:05.84	0:27.98	0:20.42	0:35.05	0:31.31	1:09.10	12:39.21	
	Pyglaf	end state	✓	✓	✓	✓	✓	TO	TO	TO
		CPU time	0:45.33	6:18.60	11:06.51	15:21.07	54:49.31			
		real time	0:39.00	6:04.37	10:12.22	14:51.09	54:20.72			
	ArgSemSAT	end state	TO	TO	TO	TO	TO	TO	TO	TO
ST	Nb lab. (≈)		Idem preferred case							
	AFDivider	end state	✓	✓	✓	✓	✓	✓	✓	MO
		CPU time	0:06.52	0:13.29	0:18.01	0:28.50	0:26.66	0:45.56	1:39.14	
		real time	0:06.26	0:13.20	0:18.78	0:31.02	0:29.46	0:50.79	1:48.30	
	Pyglaf	end state	✓	✓	✓	✓	✓	✓	✓	TO
		CPU time	0:05.43	0:17.31	0:24.78	0:31,50	0:41.69	1:13.10	3:35.76	
		real time	0:03.02	0:09.22	0:14.76	0:18.43	0:21.15	0:42.57	1:53.95	
	ArgSemSAT	end state	TO	TO	TO	TO	TO	TO	TO	TO
CO	Nb lab. (≈)		0.80×10^9	5.22×10^9	9.31×10^9	11.93×10^9	16.18×10^9	49.58×10^9	-	22×10^{15}
	AFDivider	end state	MO	MO	MO	MO	MO	MO	MO	MO
	Pyglaf	end state	TO	TO	TO	TO	TO	TO	TO	TO
	ArgSemSAT	end state	TO	TO	TO	TO	TO	TO	TO	TO

Table 2 gives the obtained results on 8 significant instances:[8] i_1 to i_8 for respectively BA_120_70_1.apx, BA_100_60_2.apx, BA_120_80_2.apx, BA_180_60_4.apx, basin-or-us.gml.20.apx, BA_100_80_3.apx, amador-transit_20151216_1706.gml.80.apx and BA_-200_70_4.apx. Note that these instances have a number of labellings under the *preferred* and *stable* semantics that is particularly large (more than a hundred thousand), and even larger for the complete semantics.

In Table 2, it is worth noting that, first, none of the chosen instances is solved by *ArgSemSAT*; second, that none of the three solvers can provide results for the complete semantics; third, that our algorithm is far better than *Pyglaf* on those instances for the preferred semantics.[9] Actually, we can observe a real order of magnitude change which increases with the hardness of the instances: from 39 s to 5 s for i_1 and from almost one hour to 31 s for i_5 (i_6 to i_8 being unsolved by *Pyglaf* in less than one hour). The last chosen instance (i_8), with its more than ten billion *preferred* labellings, presents a memory representation challenge; a compressed representation of the labellings is to be found to tackle such instances. This is also the case for the complete semantics. Finally, concerning the stable semantics, *Pyglaf* and *AFDivider* give similar results: in term of real time, *Pyglaf* is slightly better except on i_7. Nevertheless, it is worth noting that, in term of CPU time, *AFDivider* is generally better than *Pyglaf*; this last point needs further studies.

Overall, these preliminary experimental results show that the AF clustering approach brings a real added value in terms of resolution time in the case of the preferred semantics, and that an additional analysis will be necessary for identifying how to improve the results for the other semantics.

6 Related Work

There exist many approaches for enumerating semantics labellings, but most of them are non-direct, in the sense that they reduce the semantics computation to other problems (most of the time to the SAT problem). Such non-direct approaches may use some kind of cutting process and even distributed computation (it is the case of *Pyglaf* [3]). Direct approaches, such as *AFDivider*, are less common. It is with the existing direct approaches that we compare in this section the *AFDivider* algorithm.

Here are some direct approach algorithms which use some kind of cutting techniques:[10] [24], that presents an algorithm based on a dynamic analysis of an argumentation framework; [27], where the algorithm computes the labellings of an AF following its SCC decomposition; [18], where the *R-PREF*

[8] *amador-transit_20151216_1706.gml.80.apx* and *basin-or-us.gml.20.apx* are instances which come from real data of the traffic domain.

[9] Note that *Pyglaf* is also multi-core. Moreover, when we compare *Pyglaf* and *AFDivider*, we use a computer with the same number of cores. So the fact that there is a more important parallelization in *AFDivider* (so more threads) is not what explains the difference in runtime for the preferred semantics.

[10] For an overview on the different AF splitting possibilities see [8].

algorithm is based on [27]'s approach, with the addition of applying the decomposition process recursively when the labellings under construction break the SCCs; [19], where the *P-SCC-REC* algorithm, inspired by notions introduced in [5, 10, 28, 29], is the parallelized version of *R-PREF*; [11], where the algorithm splits the AF in two parts (without breaking SCCs), and computes their labelling before reunifying them. Let us compare *AFDivider* with these approaches in two respects.

- First, on their ability to break SCCs: [27] and [11] do not do so; [18] and [19] can do so, given a current SCC and an ancestor labelling, but only when the ancestor labelling has some particular effects on the current SCC; [24] always breaks SCCs as at each step at most one argument is added or removed from the considered sub-AF. Nevertheless, this way of updating argument after argument in [24] generates a lot of computations and uses a lot of memory. *AFDivider*, and this is one of its advantages, breaks SCCs whenever it is well suited to have well shaped clusters.
- Second, on their ability to compute the labellings in a distributed way: [11, 18, 24, 27] are fully sequential. *AFDivider* and [19] use distributed computation, but in [19], the computation of one labelling is mainly sequential (it is very unlikely that the greedy phase suffices to generate a labelling). Furthermore parallelizing following labellings could overload the CPUs as the number of solutions in hard AF problems may be huge.

To conclude, what distinguishes best *AFDivider* from the other ones is that cutting the AF into clusters limits the combinatorial effect due to the number of labellings, to the cluster. The other approaches propagate this effect to the whole AF. This property makes *AFDivider* well suited for non-dense AF with a clustered structure. Indeed, in such a structure, the reunifying phase will be less expensive than exploring the whole AF to construct each of the labellings.

An incremental algorithm that computes labellings has been proposed in [2] but it does not concern the enumeration problem. Other works such as [14, 22, 37] might be related to our approach as they analyze some kind of AF matrices; however, it is not done in order to cluster the AF.

7 Conclusion

AFDivider is the first algorithm that uses spectral clustering methods to compute semantics labellings. After removing the trivial part of the AF (grounded labelling), the algorithm cuts the AF into small pieces (the identified clusters), then it computes simultaneously (in each cluster) labelling parts of the AF, before reunifying compatible parts to get the whole AF labellings. Soundness and completeness of this algorithm are proven for the *stable*, the *complete* and the *preferred* semantics.

We compared the behaviour of our algorithm with other ones that also use some kind of clustering. Among the various advantages of our method (its ability to break SCCs and to compute the labellings in a distributed way), we highlighted the fact that cutting the AF into clusters has the great advantage of limiting the solving hardness to the clusters. This algorithm is particularly well suited for non-dense AFs with a clustered structure, such as the ones which are among the hardest instances of the ICCMA competition.

An empirical analysis of *AFDivider* on the benchmarks of the competition is underway and some preliminary results are presented in this paper. Nevertheless, more exhaustive experiments are planned, in particular:

- an analysis of the impact of the partition on the solving time, from a random one to a clustered one; different clustering methods may also be compared;
- a complete comparison with the other existing solvers used in ICCMA competition including the 2019 edition (for instance, CoquiAAS [26], or μ-toksia [32], which is the winner of the 2019 edition);
- and finally the use of *AFDivider* for the other tasks, on the other semantics, of the competition (see [13]).

Another interesting question to answer is how to know in a reasonable time if an AF is well suited for the *AFDivider* algorithm. In fact, this is a double question: "what is a theoretical characterization of such an AF?" and "given an AF, what is the computational cost for checking whether it respects this characterization?".

Moreover, among future works, this approach may be extended to enriched argumentation frameworks (*e.g.* with a support relation or with higher-order interactions), and to other acceptability semantics.

References

1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Rev. Mod. Phys. **74**(1), 47 (2002)
2. Alfano, G., Greco, S., Parisi, F.: Efficient computation of extensions for dynamic abstract argumentation frameworks: an incremental approach. In: IJCAI, pp. 49–55 (2017)
3. Alviano, M.: The pyglaf argumentation reasoner. In: OASIcs-OpenAccess Series in Informatics, vol. 58 (2018)
4. Barabási, A.L., et al.: Network Science. Cambridge University Press, Cambridge (2016)
5. Baroni, P., Boella, G., Cerutti, F., Giacomin, M., van der Torre, L.W.N., Villata, S.: On input/output argumentation frameworks. In: COMMA, pp. 358–365 (2012)
6. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowl. Eng. Rev. **26**(4), 365–410 (2011)
7. Baroni, P., Cerutti, F., Giacomin, M., Guida, G.: AFRA: argumentation framework with recursive attacks. Int. J. Approximate Reasoning **52**(1), 19–37 (2011)
8. Baroni, P., Giacomin, M., Liao, B.: Locality and modularity in abstract argumentation. In: Handbook of Formal Argumentation, pp. 937–979. College Publication (2018)
9. Baroni, P., Boella, G., Cerutti, F., Giacomin, M., Van Der Torre, L., Villata, S.: On the input/output behavior of argumentation frameworks. Artif. Intell. **217**, 144–197 (2014)

10. Baroni, P., Giacomin, M., Liao, B.: On topology-related properties of abstract argumentation semantics. a correction and extension to dynamics of argumentation systems: a division-based method. Artif. Intell. **212**, 104–115 (2014)
11. Baumann, R., Brewka, G., Wong, R.: Splitting argumentation frameworks: an empirical evaluation. In: Modgil, S., Oren, N., Toni, F. (eds.) TAFA 2011. LNCS (LNAI), vol. 7132, pp. 17–31. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29184-5_2
12. Bistarelli, S., Rossi, F., Santini, F.: Not only size, but also shape counts: abstract argumentation solvers are benchmark-sensitive. J. Log. Comput. **28**(1), 85–117 (2018)
13. Bistarelli, S., Santini, F., Kotthoff, L., Mantadelis, T., Taticchi, C.: Int. Competition on Computational Models of Argumentation (2019). https://www.iccma2019.dmi.unipg.it/
14. Butterworth, J., Dunne, P.: Spectral techniques in argumentation framework analysis. COMMA **287**, 167 (2016)
15. Caminada, M.: On the issue of reinstatement in argumentation. In: JELIA, pp. 111–123 (2006)
16. Carrera, Á., Iglesias, C.A.: A systematic review of argumentation techniques for multi-agent systems research. Artif. Intell. Rev. **44**(4), 509–535 (2015)
17. Cayrol, C., Lagasquie-Schiex, M.C.: On the acceptability of arguments in bipolar argumentation frameworks. In: Godo, L. (ed.) ECSQARU, pp. 378–389 (2005)
18. Cerutti, F., Giacomin, M., Vallati, M., Zanella, M.: An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation. In: KR (2014)
19. Cerutti, F., Tachmazidis, I., Vallati, M., Batsakis, S., Giacomin, M., Antoniou, G.: Exploiting parallelism for hard problems in abstract argumentation. In: AAAI, pp. 1475–1481 (2015)
20. Cerutti, F., Vallati, M., Giacomin, M., Zanetti, T.: ArgSemSAT-2017 (2017)
21. Charwat, G., Dvořák, W., Gaggl, S.A., Wallner, J.P., Woltran, S.: Methods for solving reasoning problems in abstract argumentation-a survey. Artif. Intell. **220**, 28–63 (2015)
22. Corea, C., Thimm, M.: Using matrix exponentials for abstract argumentation. In: SAFA Workshop, pp. 10–21 (2016)
23. Dung, P.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**, 321–357 (1995)
24. Dvořák, W., Pichler, R., Woltran, S.: Towards fixed-parameter tractable algorithms for abstract argumentation. Artif. Intell. **186**, 1–37 (2012)
25. Kröll, M., Pichler, R., Woltran, S.: On the complexity of enumerating the extensions of abstract argumentation frameworks. In: IJCAI, pp. 1145–1152 (2017)
26. Lagniez, J.M., Lonca, E., Mailly, J.G.: CoQuiAAS v3.0. ICCMA 2019 Solver Description (2019)
27. Liao, B.: Toward incremental computation of argumentation semantics: a decomposition-based approach. Ann. Math. Artif. Intell. **67**(3–4), 319–358 (2013)
28. Liao, B., Huang, H.: Partial semantics of argumentation: basic properties and empirical. J. Logic Comput. **23**(3), 541–562 (2013)
29. Liao, B., Jin, L., Koons, R.C.: Dynamics of argumentation systems: a division-based method. Artif. Intell. **175**(11), 1790–1814 (2011)
30. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)

31. Malliaros, F.D., Vazirgiannis, M.: Clustering and community detection in directed networks: a survey. Phys. Rep. **533**(4), 95–142 (2013)
32. Niskanen, A., Järvisal, M.: μ-toksia. Participating in ICCMA 2019 (2019)
33. Robert, M.K.: Elementary linear algebra. University of Queenland (2013)
34. Rodrigues, O., Black, E., Luck, M., Murphy, J.: On structural properties of argumentation frameworks: lessons from ICCMA. In: SAFA Workshop, pp. 22–35 (2018)
35. Schaeffer, S.E.: Graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007)
36. Von Luxburg, U.: A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
37. Xu, Y., Cayrol, C.: Initial sets in abstract argumentation frameworks. J. Appl. Non-Classical Logics **28**(2–3), 260–279 (2018)