Clustering and Distributed Computing in Abstract Argumentation

Mickael Lafages
Sylvie Doutre
Marie-Christine Lagasquie-Schiex

Université de Toulouse, IRIT
{mickael.lafages,sylvie.doutre,lagasq}@irit.fr

Tech. Report
IRIT/RR – 2018 – 11 – FR

Abstract

Computing acceptability semantics of abstract argumentation frameworks (AFs), in large-scale instances, is an area that has recently received a lot of attention.

In this report is presented the first algorithm, named *AFDivider*, that uses spectral clustering methods in order to tackle this issue. This algorithm computes the semantic labellings of an AF by first removing trivial parts of the AF, then cutting the AF into clusters and computing simultaneously in each cluster labelling parts, before finally reunifying compatible parts to get the whole AF labellings.

This algorithm is proven to be sound and complete for the stable, complete and preferred semantics. Experiments show that this cutting process and distributed computing significantly decrease the solving time of some hard AF instances.

This efficiency of *AFDivider* can be explained by the fact that it limits the solving hardness to the clusters, contrarily to the other existing clustering-based algorithms that propagate the combinatorial effect due to the number of labellings to the whole AF. *AFDivider* is thus particularly well suited for non dense AFs with a clustered structure.

Contents

Introduction

Abstract argumentation, the area we are interested in in this report, is a field of research in Artificial Intelligence that proposes methods to represent and deal with contentious information, and to draw conclusions or take decision from it. It is called "abstract" because it does not focus on how to construct arguments but rather on how arguments affect each other. Arguments are seen as generic entities that interact positively (support relation) or negatively (attack relation) with each other. This abstraction level allows to propose generic reasoning processes that could be applied to any precise definition or formalism for arguments.

There exist several approaches and formalisms to express argumentation problems. They differ about which "argumentation frameworks" and which "semantics" they use to determine the argumentation solutions. These are key notions in this research area:

- Considering the first key notion, here are some questions that have to be answered in order to "choose an argumentation framework" that fits with our need. Do we allow positive relations? If so, of which kind? Do we allow negative relations? If so, of which kind? Is there any notion of strength in arguments or in relations? The aim of making more complex argumentation frameworks is to be able to better capture human argumentation subtleties.

- Given an argumentation framework, the second key notion, "semantics", corresponds to a formal way to say how the solution of the argumentation should be decided. It is really related with the notion of "acceptability". How to define an *acceptable* argumentation problem solution?

A lot of propositions have been made to enhance the expressivity in abstract argumentation (*e.g.* [21, 6, 11, 3]). In this report, we focus on solving more efficiently argumentation problems that are expressed in the basic, seminal argumentation framework and semantics defined by Dung [12]. This is a necessary first step before considering studying the extension of this work to other, enriched argumentation frameworks and semantics.

In Dung's setting, solutions of an argumentation problem are sets of arguments which, when considered together, win the argumentation. Finding all the possible solutions of an argumentation problem, *i.e.* all its winning sets of arguments, can be very time consuming. Many argumentation problem instances, particularly large[1], are too hard to be solved in an acceptable time, as shown by the results of the ICCMA argumentation solver competition[2]. This hardness is not relative to the current state of the art but rather to the intrinsic theoretical complexity of the argumentation semantics that are tackled [13].

Moreover, there exists a recent research field in Artificial Intelligence called "Argument mining" whose object of study is how to extract arguments from natural language speeches, oral or written (see [22] for more information). When major advances in this area will make available a lot of data for

[1] This notion of largeness of an argumentation framework is not so simple to define. It is related to the fact that the computation of the solutions is complex either because of the number of arguments, or of the number of interactions, or because of the structure of the argumentation framework.

[2] `http://argumentationcompetition.org`

argumentation, this issue of solving time will become increasingly critical. There is a need for heuristics, methods and algorithms efficient enough to tackle such issues and make possible the use of automated argumentation models in the large-scale.

Enhancing the computational time of enumerating the solutions of an argumentation framework has been the object of study of many works, resulting in the elaboration of several recent algorithms such as [1, 9, 17, 2] (see [10] for an overview).

The *AFDivider* algorithm that we propose in this report has for main purpose to find all the possible solutions of an argumentation problem, using methods that have not yet been considered for this purpose, namely spectral clustering methods, originally combined to techniques that have already been applied in other existing algorithms. The solutions are defined in terms of semantic labellings [7, 5]. In a word, *AFDivider* computes the semantic labellings of an AF by first removing trivial parts of the AF, then cutting the AF into clusters and computing simultaneously in each cluster labelling parts, before finally reunifying compatible parts to get the whole AF labellings.

The main aim of this report is to present the *AFDivider* algorithm, to show how it differs from other algorithms and to give experimental results on how it behaves on hard argumentation problem instances.

We start with giving the required background knowledge on abstract argumentation (Chapter 1) and on other formal additional underlying notions (Chapter 2). We then present our contribution (Chapter 3), and we give experimental results (Chapter 4), before comparing our algorithm with other existing ones (Chapter 5). Finally, we conclude on perspectives linked with the *AFDivider* algorithm.

Chapter 1

Abstract argumentation

In this section is presented the basic framework of abstract argumentation and some semantics.

1.1 Framework

According to Dung [12], an abstract argumentation framework consists of a set of arguments and of a binary attack relation between them.

Definition 1 *(Argumentation framework). An* argumentation framework *(AF) is a pair* $\Gamma = \langle A, R \rangle$ *where* A *is a finite 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*. The set of all possible argumentation frameworks is denoted as* $\mathscr{AF}$*.*

Hence, an argumentation framework can be represented by a directed graph with arguments as vertices and attacks as edges. Figure 1.1 shows an example of an AF.

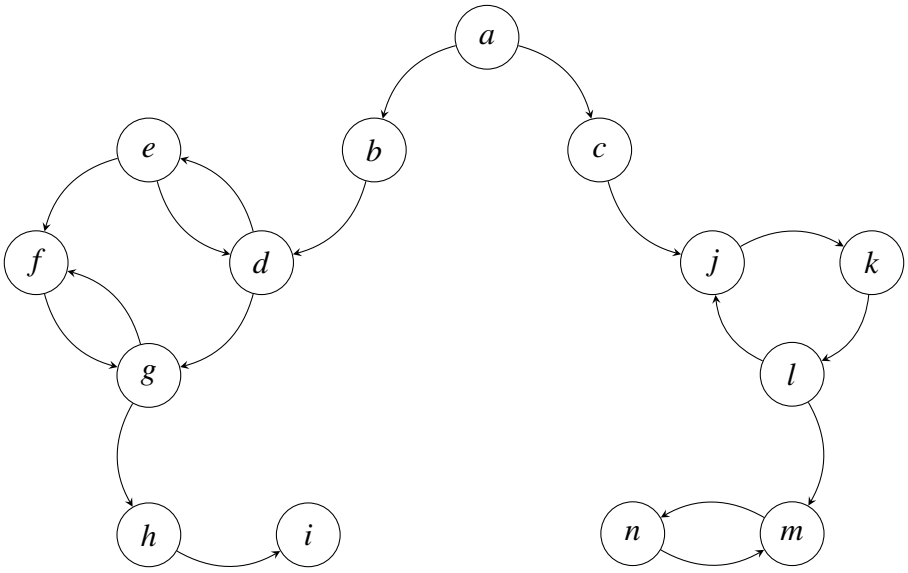


Figure 1.1: Example of an argumentation framework

1.2 Labelling and semantics

Acceptability semantics can be defined in terms of labellings [7, 5].

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

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

Definition 3 *(Legally labelled argument). Let $\Gamma = \langle A, R \rangle$ be an AF, and $\ell \in \mathscr{L}(\Gamma)$ be a labelling.*

- *An $\texttt{in}$-labelled argument is said to be legally $\texttt{in}$ iff all its attackers are labelled $\texttt{out}$.*
- *An $\texttt{out}$-labelled argument is said to be legally $\texttt{out}$ iff at least one of its attackers is labelled $\texttt{in}$.*
- *An $\texttt{und}$-labelled argument is said to be legally $\texttt{und}$ iff it does not have an attacker that is labelled $\texttt{in}$ and one of its attackers is not labelled $\texttt{out}$.*

Definition 4 *(Admissible labelling). Let $\Gamma = \langle A, R \rangle$ be an AF, and $\ell \in \mathscr{L}(\Gamma)$ be a labelling. ℓ is an* admissible labelling *of Γ iff it satisfies the following conditions for any $a \in A$:*

- *For each $a \in \texttt{in}(\ell)$, a is legally $\texttt{in}$.*
- *For each $a \in \texttt{out}(\ell)$, a is legally $\texttt{out}$.*

Definition 5 *(Complete labelling). Let $\Gamma = \langle A, R \rangle$ be an AF, and $\ell \in \mathscr{L}(\Gamma)$ be a labelling. ℓ is a* complete labelling *of Γ iff it satisfies the following conditions for any $a \in A$:*

- *For each $a \in \texttt{in}(\ell)$, a is legally $\texttt{in}$.*
- *For each $a \in \texttt{out}(\ell)$, a is legally $\texttt{out}$.*
- *For each $a \in \texttt{und}(\ell)$, a is legally $\texttt{und}$.*

Definition 6 *(Grounded, preferred and stable labelling). Let $\Gamma = \langle A, R \rangle$ be an AF, and $\ell \in \mathscr{L}(\Gamma)$ be a labelling.*

- *ℓ is the* grounded labelling *of Γ iff it is the complete labelling of Γ that minimizes (w.r.t $\subseteq$) the set of $\texttt{in}$-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 $\texttt{in}$-labelled arguments.*
- *ℓ is a* stable labelling *of Γ iff it is a complete labelling of Γ which has no $\texttt{und}$-labelled argument.*

It can be noticed that all complete labellings include the grounded labelling, and, as stable and preferred labellings are a type of complete labellings, they, also, include the grounded labelling.

Definition 7 *(Semantic). A semantic σ is a total function $\sigma : \mathscr{AF} \to 2^{\mathscr{L}(\Gamma)}$ that associates to an AF $\Gamma = \langle A, R \rangle$ a subset of $\mathscr{L}(\Gamma)$.*

Given an AF $\Gamma = \langle A, R \rangle$, 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 $\mathscr{L}_\sigma(\Gamma)$.

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6
a	*in*	*in*	*in*	*in*	*in*	*in*
b	*out*	*out*	*out*	*out*	*out*	*out*
c	*out*	*out*	*out*	*out*	*out*	*out*
d	*out*	*out*	*in*	*in*	*und*	*und*
e	*in*	*in*	*out*	*out*	*und*	*und*
f	*out*	*out*	*in*	*in*	*und*	*und*
g	*in*	*in*	*out*	*out*	*und*	*und*
h	*out*	*out*	*in*	*in*	*und*	*und*
i	*in*	*in*	*out*	*out*	*und*	*und*
j	*und*	*und*	*und*	*und*	*und*	*und*
k	*und*	*und*	*und*	*und*	*und*	*und*
l	*und*	*und*	*und*	*und*	*und*	*und*
m	*und*	*out*	*und*	*out*	*und*	*out*
n	*und*	*in*	*und*	*in*	*und*	*in*
grounded					×	
complete	×	×	×	×	×	×
preferred		×		×		
stable						

Table 1.1: Semantic labellings

Example 1 *Let us consider the AF of Figure 1.1. Table 1.1 shows the labellings corresponding to the different semantics (the other possible labellings are not given).*

As you can see this AF has no stable labellings.
Its unique grounded labelling ($\ell_{gr} \equiv \ell_5$) is such that $\ell_{gr}(a) =$ in, $\ell_{gr}(b) = \ell_{gr}(c) =$ out and $\forall x \in A \setminus \{a,b,c\}$, $\ell_{gr}(x) =$ und (see Figure 1.2).

1.3 Problem types and complexities

Given an AF instance, a semantic σ and a, an argument of the instance, several problem types are of interest. Here is a non-exhaustive list of them:

- Give a labelling.
- Give all labellings.
- Is a labelled *in* in one labelling?
- Is a labelled *in* in all labellings?
- Give all labellings in which a is labelled *in*.
- Give one labelling in which a is labelled *out*.

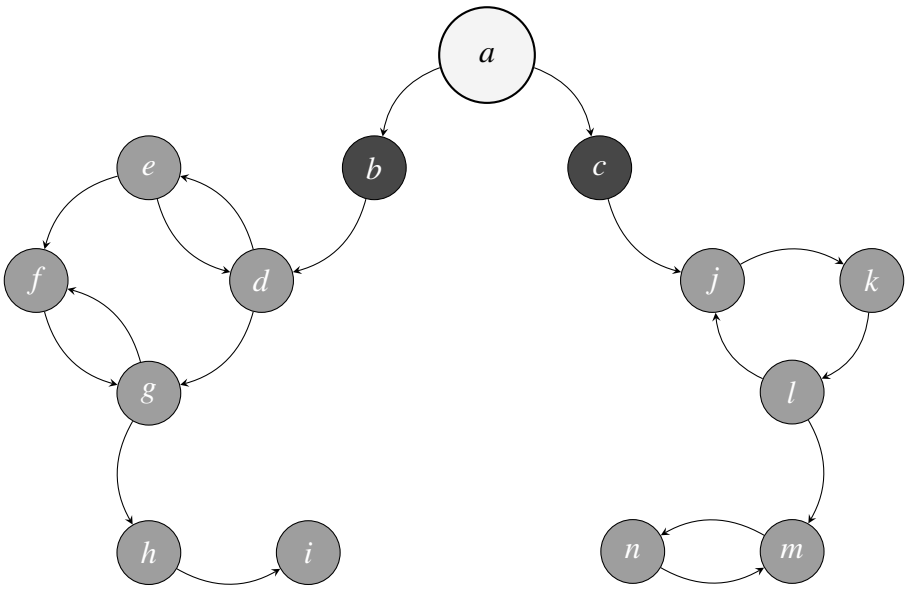


Figure 1.2: Grounded labelling

- Give all labellings in which a is labelled `out`.

Each of these problems admits a particular complexity, which depends of the chosen semantic. They belong to the complexity classes between L and Σ_2^P. In [14] are listed various problems and their complexities according to the studied semantic.

Of course, the complexity of the studied problem combined with the "largeness" of the given AF instance has an important impact on the efficiency of the problem resolution particularly in terms of execution time.

`Note:` *It is difficult to define formally what is a "large-scale argumentation framework". We have seen "small" ones in terms of number of arguments that were pretty hard to solve given the relation structure between the arguments. As a consequence, many labellings were possible and it was very time consuming to compute them. We have also find out "large" ones in terms of number of arguments that were easy to solve. The hardness of finding all solutions is probably due to a certain range of relation density (not too high because in this case it would be easy to find that there is not so many solutions). This analysis is left for future works.*

Chapter 2

Mathematical background

In this chapter are presented the basic notions under which the proposed algorithm is based: notions related to set theory, graph theory, matrices and constraints satisfaction problem (CSP) modelling.

2.1 Mathematical notions

2.1.1 Set theory

Given that our aim is to cut argumentation problems into smaller pieces, the notion of *partition* will be useful.

Definition 8 *(Partition). A partition* $\Omega = \{\omega_1, ..., \omega_n\}$ *of a set* E *is a set of subsets of* E *such that:*

- $\forall i, j \in \{1, ..., n\}$ *s.t.* $i \neq j, \omega_i \cap \omega_j = \emptyset$
- $\bigcup_{i \in \{1,...,n\}} \omega_i = \mathrm{E}$

A partition of two elements is called a bipartition.

2.1.2 Graph Theory

In this section are given the notions of graph theory used in the algorithm. Different types of graph are presented, and also notions related to nodes, relations, paths, subgraphs and topology.

Graph types

Definition 9 *(Non-directed and directed graph). A non-directed (respectively directed) graph is an ordered pair* $G = (V, E)$ *where:*

- V *is a set whose elements are called nodes or vertices;*
- E *is a set of unordered (respectively ordered) pairs of vertices called non directed edges (respectively directed edges).*

Definition 10 *(Weighted graph). A weighted directed (respectively non-directed) graph is an ordered pair* $G = (V, E, W)$ *where:*

- (V, E) *is a directed (respectively non-directed) graph.*

- *$W : E \to \mathbb{R}$ is a total function that associates a weight to each directed (respectively non-directed) edge in E.*

In the following we will implicitly consider that a non-weighted directed (respectively non-directed) graph $G = (V,E)$ is a weighted directed (respectively non-directed) graph whose edges are weighed 1.

Node and edge relations

Definition 11 *(Incident). Let $G = (V,E)$ be a graph (directed or not), and $\mathrm{e} = (v_i, v_j) \in E$ be an edge. We say that* e *is incident to v_i and v_j, or joins v_i and v_j. Similarly, v_i and v_j are incident to* e.

Definition 12 *(Adjacent). Let $G = (V,E)$ be a graph (directed or not), and $v_i \in V$, $v_j \in V$ be two nodes of G. We say that v_i and v_j are adjacent if $(v_i, v_j) \in E$ or $(v_j, v_i) \in E$.*

Definition 13 *(Degree). Let $G = (V,E)$ be a graph and $v \in V$ be a vertex. The degree of v in G, noted $deg(G,v)$, is its number of incident edges.*

Definition 14 *(Weighted degree). Let $G = (V,E,W)$ be a weighted graph (directed or not). Let $v \in V$ be a vertex and $I = \{e | e = (v,v') \in E\}$ the set of its incident edges. We define the weighted degree of v, noted $deg_w(G,v)$, as the weight sum of its incident edges:*

$$deg_w(G,v) = \sum_{e \in I} W(e)$$

`Note:` *In order to simplify the notation, $deg(G,v)$ (resp. $deg_w(G,v)$) will be noted $deg(v)$ (resp. $deg_w(v)$) when there is no ambiguity about the graph in which the degree is measured.*

Connectivity

Definition 15 *(Path). Let $G = (V,E)$ be a directed graph (respectively non-directed). Let* $\mathrm{p} = (v_1, v_2, \ldots, v_{k-1}, v_k)$ *be a sequence of vertices of G.* p *is a path if all the vertices (except perhaps the first and last ones) are distinct and $\forall i \in \{1,...,k-1\}$, (v_i, v_{i+1}) belongs to E (respectively $(v_i, v_{i+1}) or (v_{i+1}, v_i)$ belong to E).*

The length of p *is the number of vertices of* p *minus 1 (i.e. $k-1$).*

Definition 16 *(Connected graph). Let $G = (V,E)$ be a graph (directed or not). G is a connected graph if, for all distinct vertices $v_i \in V$ and $v_j \in V$, there exists a non-directed path* p *in G s.t. v_i is the first vertex of* p *and v_j is the last vertex of* p*. Otherwise the graph is called a disconnected graph.*

Definition 17 *(Subgraph). Let $G = (V,E)$ be a directed graph (respectively non-directed graph). A subgraph* $\mathrm{S} = (V',E')$ *of G is a directed graph (respectively non-directed graph) such that:*

- *$V' \subseteq V$.*
- *$E' \subseteq E$.*
- *$\forall (v_i, v_j) \in E', v_i \in V'$ and $v_j \in V'$.*

Definition 18 *(Graph restriction $\downarrow$). Let $G = (V,E)$ be a graph and $S \subseteq V$ be a set of vertices. The* restriction *of G to S is the subgraph of G defined as $G\downarrow_S \equiv (S, E \cap (S \times S))$.*

Note: Notice that when restricting a graph G to a set S of arguments, any edge of G whose endpoints are both in S must be kept. It is not the case for the more general subgraph definition. Indeed, a subgraph of G whose set of arguments coincides with S may not keep all these edges.

Definition 19 *(Connected component). Let G be a graph (directed or not). Let H be a subgraph of G such that:*

- *H is connected.*
- *H is not contained in any connected subgraph of G which has more vertices or edges than H has.*

Then H is a connected component of G.

In the following by "component" we mean "connected component".

Topology

Formally, *"clusters"* can be defined as following:

Definition 20 *(Cluster). Let G be a graph. A cluster c of G is a connected subgraph of G.*

In order to define these clusters, we can use the notion of relation density:

Definition 21 *(Relation density). Let $G = (V,E)$ be a graph, S be a subset of V. The relation density $\mathscr{R}_d(G)$ of the graph G is defined by:*

$$\mathscr{R}_d(G) = \frac{|E|}{|V|}$$

In practice, given an initial graph, we will be interested by some of its connected subgraphs which have similar sizes (number of nodes) and such that their inside relation density is greater than their neighbouring relation density. Note that some graphs can be defined in which it would be difficult to identify clusters:

Definition 22 *(Random graph). A random graph $G_{rand} = (V,E)$ of relation density $\mathscr{R}_d(G_{rand})$ is a graph constructed by creating edges randomly between the nodes of V until reaching the relation density $\mathscr{R}_d(G_{rand})$ wanted.*
This random construction process prevents the creation of a clustered structure.

Definition 23 *(Random graph of degree sequence). Let $G = (V,W,E)$ be a graph. Let $G^{\tau}_{rand} = (V,W',E')$ be a random graph of relation density $\mathscr{R}_d(G)$. G^{τ}_{rand} is a random graph of same degree sequence as G if:*

$$\forall v \in V,\ deg_w(G,v) = deg_w(G^{\tau}_{rand},v)$$

2.1.3 Matrices

In this section are presented the matrix notions needed for the AF cutting process, the key notions being the eigenvectors and values, and the laplacian matrix.

Definition 24 *(Matrix). A matrix* M *with* m *lines and* n *columns, or a* $m \times n$ *matrix, with values in some field of scalars* K *is an application of* $\{1,2,...,m-1,m\} \times \{1,2,...,n-1,n\}$ *in* K. $M_{i,j} \in K$ *is the image of the couple* (i,j). *i is called the line index and j, the column index.*

Definition 25 *(Eigenvector and eigenvalue). Let* E *be a vector space over some field* K *of scalars, let* u *be a linear transformation mapping* E *into* E *(i.e.* $u : E \to E$*), and let* $v \in E$ *be a non-zero vector.*

v is an eigenvector of u *if and only if there exists a scalar* $\lambda \in K$ *such that:*

$$u(v) = \lambda \cdot v$$

In this case λ *is called eigenvalue (associated with the eigenvector v).*

For more details on eigenvectors and eigenvalues see [20], Chapter 6.

Definition 26 *(Adjacency matrix). Let* $G = (V,E,W)$ *be a weighted non-directed graph. The adjacency matrix* M_a *of* G *is an* $n \times n$ *matrix (with* $n = |V|$*) defined as:*

$$(M_a)_{i,j} = \begin{cases} W((v_i,v_j)) \text{ if } (v_i,v_j) \in E \\ 0 \text{ otherwise} \end{cases}$$

If the weights of a graph *G* represent similarity measures then adjacency matrix is called the similarity matrix of *G*.

Definition 27 *(Degree matrix). Given a non-directed graph* $G = (V,E,W)$*, the degree matrix* M_d *for* G *is an* $n \times n$ *matrix (with* $n = |V|$*) defined as:*

$$(M_d)_{i,j} = \begin{cases} deg_w(v_i) \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

`Note:` M_d *is a diagonal matrix.*

Definition 28 *(Laplacian matrix). Given a non-directed graph* $G = (V,E,W)$*, the laplacian matrix* M_l *for* G *is an* $n \times n$ *matrix (with* $n = |V|$*) defined as:*

$$M_l = M_d - M_a$$

2.2 Clustering algorithms

Finding clusters in graph is a subject that has been widely studied. In this section we will present two approaches that seem to be appropriate for argumentation framework clustering. For an overview of non directed graph clustering algorithms see [24] and for directed ones see [19].

The first approach relies mainly on a coarsening process of the initial AF to maximize a certain criterion. The second, the one we have chosen to implement in our algorithm, is based on a spectral analysis of the laplacian matrix of an AF.

2.2.1 ConClus

Before presenting the *ConClus* algorithm, proposed in [23], we will define the modularity maximization criterion on which several clustering algorithms are based.

The modularity is a measure that represents how much a given graph has a clustered structure. When this value is around 0 (or lower than 0) it means that the graph has a random structure and the more this value tends toward 1 the more the graph has well defined clusters in itself.

Formally, modularity is defined as following:

Definition 29 *(Modularity criterion). Let $G = (V,E)$ a graph and $\Omega = \{\omega_1, ..., \omega_n\}$ be a partition of V such that:*

$$\forall i \in \{1, ..., n\},\ G\downarrow_{\omega_i} \text{ is a connected subgraph}$$

The modularity $\mathcal{Q}$ of G is defined by:

$$\mathcal{Q} = \sum_{i=1}^{n} \left[\tau_{clust}^{\omega_i} - \tau_{rand}^{\omega_i} \right]$$

Where:

- $\tau_{clust}^{\omega_i}$ *is the fraction of all edges that lie within $G\downarrow_{\omega_i}$. Formally:*

$$\tau_{clust}^{\omega_i} = \frac{|E \cap (\omega_i \times \omega_i)|}{|E|}$$

- *Let $G_{rand}^{\tau} = (V,E')$ be a random graph of same degree sequence as G.*

  $\tau_{rand}^{\omega_i}$ *is the fraction of all edges that lie within $G_{rand}^{\tau}\downarrow_{\omega_i}$. Formally:*

$$\tau_{rand}^{\omega_i} = \frac{|E' \cap (\omega_i \times \omega_i)|}{|E|}$$

  Notice that by Definition 23, we have $|E| = |E'|$.

Given a graph $G = (V, E)$, algorithms based on modularity maximization criterion are interested in finding the partition $\Omega = \{\omega_1, ..., \omega_n\}$ of V that maximizes the modularity $\mathscr{Q}$.

The inconvenient of seeking clusters in a graph using such a maximization is that, in presence of relative large communities, relative small ones may be undetected and considered as belonging to a larger group (see [16] for more details).

The *ConClus* algorithm uses a similar modularity measure that takes into account the directionality of edges and also a parameter that to prevent the relative small cluster blindness problem. In the interest of brevity, we will not present this modularity measure in details but we give the intuition of the *ConClus* algorithm that uses it (for more details see [23]):

1. Given a graph and a partition corresponding to the singletons of each node, a list of the edges is determined such that, if these edges are contracted in order to form a "super node", the resulting graphs would have the best modularities.

2. A set of possible graphs is thus created.

3. The edge contractions that are recurrent in the process of creating those graphs (above 50% of them) are considered as permanent and a consensus graph is determined.

4. If is not possible to increase significatively the modularity the algorithm stops. Else, it goes back to the first step with the consensus graph as input graph.

Finally, in the graph returned by *ConClus* algorithm, the super nodes (contracted ones) correspond to a partition of the initial nodes and to a clustering of the initial graph.

2.2.2 Spectral clustering

The spectral clustering is a clustering method which is based on the spectral analysis of a similarity laplacian matrix.

A similarity matrix is a square matrix in which the lines and the columns describe the same set of elements. The matrix coefficients (*i.e.* the cell values) represent how much a element is similar to another, according to given similarity measure.

In short, here is how the spectral clustering works:

- Given a similarity matrix, the laplacian of this matrix is computed.
  - The lines of the laplacian matrix correspond to the coordinates of the elements in a certain similarity space.
- The eigenvectors of the laplacian matrix with their associated eigenvalues are computed.
- The eigenvalues computed are sorted increasing order. A number n of them is kept with their associated eigenvectors.
  - This solving and sorting process is done in order to project the datapoints in a new space which maximizes the closeness of similar elements. This space basis is formed by the computed eigenvectors. The eigenvalue of an eigenvector represents how much the datapoints are scattered on the eigenvector corresponding axis. Given that we are interested in the dimensions that maximize the best similarity (axes on which the datapoints are closed to each other), we keep the n smallest eigenvalues and their eigenvectors.

  - If there are clusters in a data set, it is reasonable to think that the number of small eigenvalues are the number of groups identified (the datapoints being rather homogeneous following that axis). An heuristic to find the appropriate number of dimensions to keep is to detect the jump in the eigenvalues sequence (sorted in increasing order).

- A matrix whose columns are the remaining eigenvectors is constructed. The lines of it represent the new elements coordinates.

- Once this data treatment is done, 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 elements (see [18] for more information about *KMeans* algorithm).

In Section 3.3, this algorithm is illustrated on a concrete AF example.

`Note:` *For more information on spectral clustering see [26].*

Although the *ConClus* algorithm is one of the best algorithms to find clusters in directed graphs, generating all these graphs could be time consuming. For this reason, in our algorithm presented in Chapter 3, we have chosen to use firstly the clustering algorithm based on spectral analysis. Experiments with *ConClus* algorithm are left for future works.

2.3 Constraint Satisfaction Problem

In this section is presented the formal definition of a constraint satisfaction problem (CSP). A CSP modelling will be used for reunifying the solutions parts computed by our algorithm.

Given a set of changeable state objects, a Constraint Satisfaction Problem (CSP) is a mathematical problem in which we look for a configuration of object states (*i.e.* a mapping where each object has a particular state) that satisfies a certain number of constraints.

Definition 30 *(Constraint Satisfaction Problem). A CSP is defined by a triplet* $\Psi = \langle X, D, C \rangle$ *where:*

- $X = \{X_1, ..., X_n\}$ *is a set of variables.*
- $D = \{D(X_1), ..., D(X_n)\}$ *is a set of domains, where* $D(X_i) \subset \mathbb{Z}$ *is the finite set of values that variable* X_i *can take (i.e.* $D(X_i)$ *is the domain of* X_i*).*
- $C = \{c_1, ..., c_e\}$ *is a set of constraints.*

Definition 31 *(Constraint). A constraint* c_i *is a boolean function involving a sequence of variables* $X(c_i) = (X_{i_1}, ..., X_{i_q})$ *called its scheme. The function is defined on* $\mathbb{Z}^q$*. A combination of values (or tuple)* $\tau \in \mathbb{Z}^q$ *satisfies* c_i *if* $c_i(\tau) = 1$ *(also noted* $\tau \in c_i$*). If* $c_i(\tau) = 0$ *(or* $\tau \notin c_i$*),* τ *violates* c_i*.*

Definition 32 *(Instantiation). An instantiation of the X variables is a mapping where each* X_i *takes a value in its domain* $D(X_i)$*.*

Definition 33 *(CSP Solution). A solution of a CSP is an instantiation of the X variables that violates no constraint.*

`Note:` *For more details on CSP see [25].*

Chapter 3

Distributed Algorithms

The distributed way of computing semantics we are going to present relies on the idea that argumentation frameworks constructed from real data have a particular structure. Indeed, given that people have thematics and goals while arguing, it is a reasonable conjecture to say that the AFs obtained from real argumentation are not random, that they have globally a low density of relations between arguments and finally that their topology reveals clusters, *i.e.* areas with higher relation densities and others with lower ones.

The idea behind clusters is that we can see them as subtopics of the main argumentation object. Although there may have connections between these subtopics it seems judicious to cut the main problem into small pieces and to apply a reasoning process on them, taking into account the possible interactions between them. Searching a solution in one of these subtopics may be seen as an assumption-based reasoning: solutions will be sought for all configurations of attack status from neighbour subtopics.

Finally, the "compatible" subtopic solutions will be merged to form a solution of the main argumentation framework.

The algorithm that we propose is based on the ideas mentioned above. Given an argumentation framework Γ and a semantics σ, four major steps apply:

1. A pretreatment on Γ removes "trivial" parts of it.

2. Clusters (areas with high relation density) in Γ are identified.

3. The labellings under semantics σ in each of these clusters are computed **in parallel**.

4. The results of each cluster are reunified to get the labellings of Γ.

The problem we are interested in is to find all the labellings of Γ under the **complete**, the **stable** and **preferred** semantics, as quickly as possible. Given that the grounded semantics can be computed in linear time and that it gives only one labelling, the approach of this paper is unappropriated.

In the following sections we will present the different steps mentioned above, then the algorithms themselves. Afterwards, we will highlight the relation between AFs with input, introduced by Baroni et al., and the cluster structures introduced in this paper. Finally, we will prove that the algorithms are sound and complete for the *stable*, *complete* and *preferred* semantics.

3.1 Definitions

For practical reasons we define a transformation that gives a non-directed graph from a directed graph.

Definition 34 *(Undirection transformation) Let $G = (V, E, W)$ be a directed weighted graph. The non-directed graph $G' = (V', E', W')$ obtained by the undirection transformation of G (noted $\mathscr{U}(G)$) is defined as following:*

- $V' = V$.
- $E' = \{(v_i, v_j) | (v_i, v_j) \in E \text{ or } (v_j, v_i) \in E\}$.
- $W' : E' \to \mathbb{R}$ *is defined as following:*

$$W' : (v_i, v_j) \mapsto \begin{cases} W((v_i, v_j)) \text{ if } (v_i, v_j) \in E \text{ and } (v_j, v_i) \notin E, \\ W((v_j, v_i)) \text{ if } (v_i, v_j) \notin E \text{ and } (v_j, v_i) \in E, \\ W((v_i, v_j)) + W((v_j, v_i)) \text{ if } (v_i, v_j) \in E \text{ and } (v_j, v_i) \in E \end{cases}$$

By convention, if G is a non-directed graph we say that $G = \mathscr{U}(G)$.

3.2 Pretreatement: removing AF trivial parts

What we call the "trivial part" (or "fixed part") of an AF is simply a part of it that has a unique and fixed labelling that can be computed in linear time. As it will be seen in Section 3.4, for each attack between clusters, several cases have to be considered and this can be very time consuming. In order to avoid this cost for attacks that are in the "trivial part", we simply cut that part from the AF and, only after that, look for clusters.

Given that we are interested in the complete, stable and preferred semantics, a good way to remove that "trivial part" is to compute the grounded labelling of the AF. Indeed, all complete, stable and preferred labellings include the grounded one. Furthermore, the grounded labelling is computable in linear time. This idea of preprocessing has been exploited in [8].

Once the grounded labelling ℓ_{gr} is computed for a given Γ, we consider a restriction Γ_{hard} of Γ to those arguments that are labelled `und` in the grounded labelling:

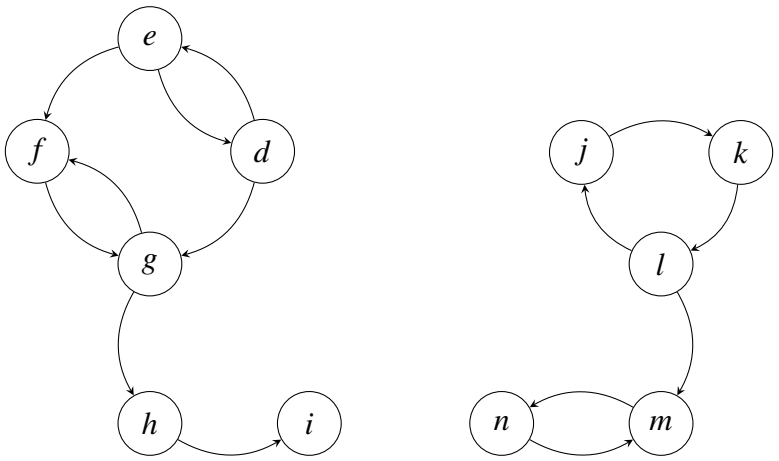
$$\Gamma_{hard} = \Gamma\downarrow_{\{a | a \in A, \ell_{gr}(a) = \texttt{und}\}}$$

Γ_{hard} may possibly be a disconnected graph. We take advantage of that potential property in order to enhance the parallel computing as it will be explained in Sections 3.4 and 3.5.

Example 2 *As an illustration, let Γ be the AF represented in Figure 1.1. In the grounded labelling ℓ_{gr}, argument a is labelled* `in`*, arguments b and c are labelled* `out` *and all the other ones are labelled* `und` *(see Figure 1.2). The arguments in the "trivial part" of Γ are then a, b and c. For all possible complete, stable and preferred labellings, they will be labelled like in ℓ_{gr}.*

Γ_{hard} is obtained by removing a, b and c from Γ. Γ_{hard} is represented on Figure 3.1.

Given that Γ_{hard} is a disconnected graph, we split it into two independent AFs γ_1 and γ_2 as shown in Figure 3.1. The same treatment described in Section 3.3 will be applied simultaneously to both of them.



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

Figure 3.1: Γ_{hard} and its components

3.3 Identifying Clusters

Finding clusters in the AF structure is tantamount to resolving a *Sparsest Cut Problem*. Such a problem, in the context of argumentation frameworks, consists in finding a partition of the arguments that minimizes the number of attacks across the cuts while balancing the parts size (*i.e.* number of arguments). As it is an NP-Hard problem, we look for an algorithm with a lower complexity using a heuristic that gives us an approximation of the sparsest cut partition.

This problem has been widely studied in graph theory and a lot of algorithms have been proposed. Some of them work on the graph structure itself. They merge nodes together while maximizing a certain criterion. At the end the super-nodes created represent the clusters. See Section 2.2.1 for an algorithm example.

The algorithm that we have chosen to use here is based on a completely different approach, used in *data mining clustering*. The idea is to do a spectral analysis of the similarity matrix of a data set in order to find the most homogeneous partition of it. See Section 2.2.2 for information about this kind of clustering.

We have begun our work by this approach because we thought that merging nodes could be more time consuming. Indeed our cutting algorithm consists mainly in solving a system of equations corresponding to a particular matrix, as explained in the following. This system of equations can be considerably simplified if the corresponding matrix is sparse (*i.e.* with many zero values). It seems reasonable to think that, because of human argumentation structure,[1] this matrix will most of the time be a sparse one. The comparison with other clustering algorithms will be made in future works.

Given that the approach is intended for catching similarities between data, we had to adapt it for arguments and AFs. In our adaptation, the similarity between arguments represents how much an argument is connected or related to another one. The initial similarity matrix is here a kind of adjacency matrix of the AF nodes. As similarity is a symmetric relation, there is no notion of directionality in that approach. Nevertheless, we needed to express the fact that two nodes between which there are two attacks (one in each direction) are more related to one another than two nodes between which there is only one attack. That is why it was not possible to simply use the adjacency matrix of the AF as initial similarity matrix.

[1] We made the conjecture that argumentation frameworks obtained from real data has globally a low relation density. We thought that simply because arguments have precise goals. It is very unlikely to have an argument attacking most of what have been said. On the contrary when arguing we attack key points that change the outcome of the argumentation.

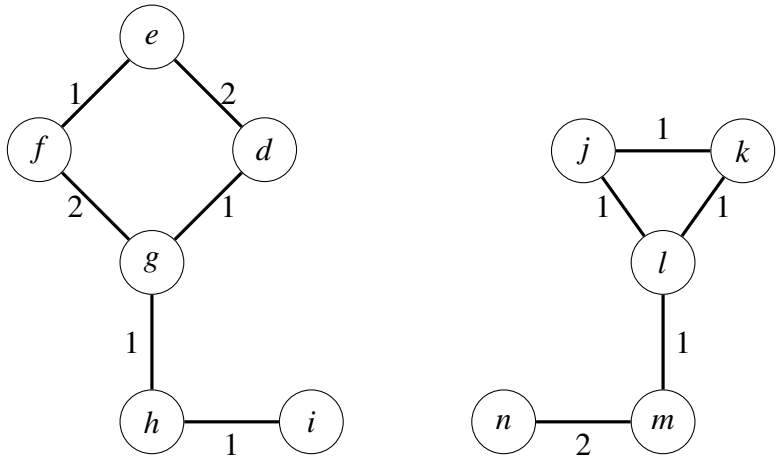
Instead of it, we use the adjacency matrix of the weighted non-directed graph obtained by the undirection transformation of the initial AF.

Example 3 *Let consider the components of Γ_{hard} shown in Figure 3.1. The weighted non-directed graphs obtained by $\mathscr{U}(\gamma_1)$ and $\mathscr{U}(\gamma_2)$ are shown in Figure 3.2. The similarity matrices of $\mathscr{U}(\gamma_1)$ and $\mathscr{U}(\gamma_2)$ are:*

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

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

As you can see, given that the AF relation density is low the matrices are rather sparse.



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

Figure 3.2: $\mathscr{U}(\gamma_1)$ and $\mathscr{U}(\gamma_2)$

.

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 appear some agglomerates corresponding to the node clusters.

To do that, we compute the n smallest eigenvalues[2] of the laplacian matrix obtained from the similarity matrix and the vectors associated with them (this n is an arbitrary parameter).

[2]There exist algorithms, such as *Krylov-Schur* method, able to compute eigenvectors from smallest to greatest eigenvalue and stop at any wanted step (*e.g.* 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.

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).

Example 4 *Let take as example the case of γ_2. The degree matrix $M_d^{\gamma_2}$ of $\mathscr{U}(\gamma_2)$ is:*

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

and then, its laplacian matrix $M_l^{\gamma_2}$ is:

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

The eigenvectors of $M_l^{\gamma_2}$ are:

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

and their eigenvalues are:

$$\begin{array}{c} \begin{array}{ccccc} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{array} \\ \begin{bmatrix} 2.476651 \times 10^{-16} & 5.857864 \times 10^{-1} & 3.000000 & 3.414214 & 5.000000 \end{bmatrix} \end{array}$$

In this example we have chosen to keep all the vectors (i.e. $n = 5$).

Now that the similarity space is found, another important step is to find how many groups we have in that space. Intuitively, the number of eigenvectors with small eigenvalues, and so, the number of axes with small variance is the number of clusters. However, within the n smallest eigenvalues determined, it is difficult to formally say what is a small eigenvalue, and so, what is the number of clusters to chose.

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 you will have to compute cases as explained in Section 3.4 (which is very time consuming). We have then to find a compromise between number of clusters and homogeneity.

As eigenvalues say, in the end, how much the corresponding clusters will be homogeneous, the heuristic we have chosen to consider is to look for the "best elbow" in that ascending order sequence. We look for the number of dimensions to keep just before the quick growth of the variance. If the topology of the AF let appear some clusters then we will indeed have elbows. We can see in Figure 3.3 that this "best elbow" in the eigenvalues sequence (blue line with squares) is in second position. In that case the number of clusters determined by that heuristic is so 2.

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).

As you can see the first point of the second derivative, corresponding to the concavity formed by the first three eigenvalues, is the first value above the threshold and then we determine that the "best elbow" is in position 2.

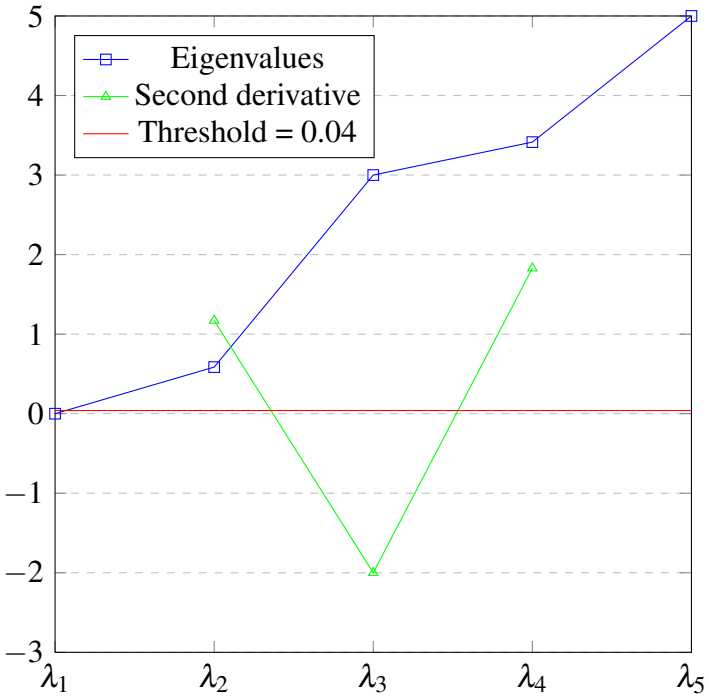


Figure 3.3: Eigenvalues sorted by ascending order

Once the number of clusters is chosen, we remove from the similarity matrix the columns that are after this number (*i.e.* we remove the dimensions we are not interested in for the clustering). The lines of the resulting matrix, which columns are the kept eigenvectors, correspond to the coordinates of the nodes in that new compressed similarity space.

Finally, we just have to apply a *KMeans* type algorithm [18] to find the groups of datapoint in that space and so have the partition of arguments we wanted.

Example 5 *Given that the chosen number of clusters is* 2*, we keep only the vectors* v_1 *and* v_2 *and when binded by column the lines they form correspond to the coordinates of the arguments in a new space that maximizes similarity.*

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

As you can see the v_1 dimension is useless. In practice it is removed.

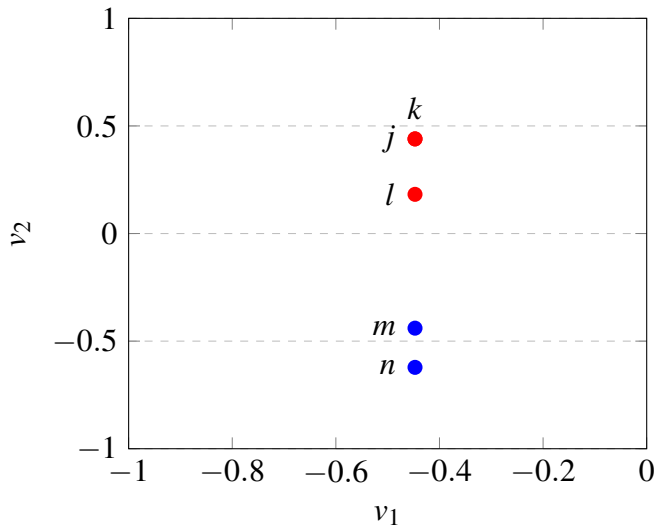


Figure 3.4: Arguments datapoints projected in similarity space

Figure 3.4 clearly shows two clusters.

Example 6 *The partition determined for γ_1 and γ_2 is shown in Figure 3.5.*

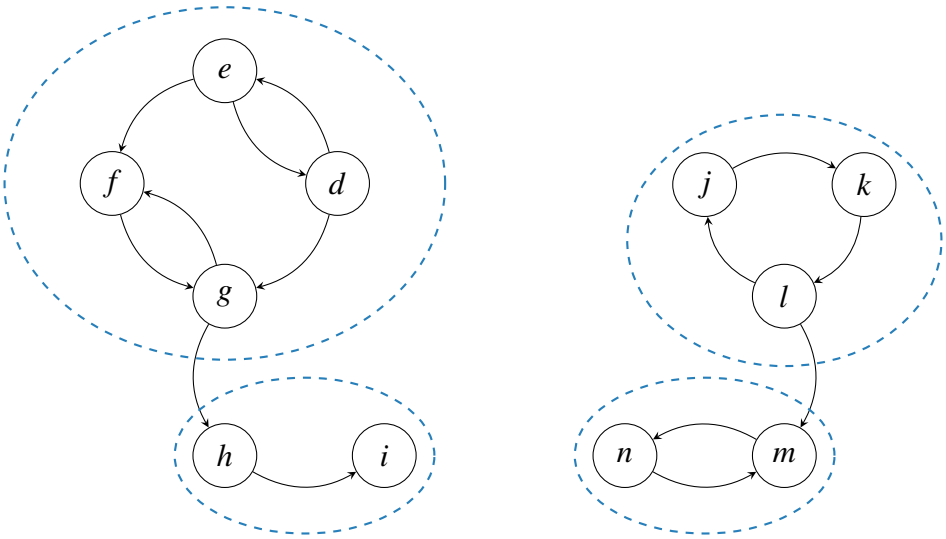
We define a cluster structure at this step that will be useful in the following.

Definition 35 *(Cluster structure). Let $\Gamma = \langle A, R \rangle$ be an AF, Ω be the partition of A, ω be an element of Ω (i.e. a set of arguments) and $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster structure. κ is defined as follows:*

$$\begin{aligned} \gamma &= \Gamma{\downarrow_\omega} \\ I &= \{(a,b) | (a,b) \in R, b \in \omega \text{ and } a \notin \omega\} \\ O &= \{(a,b) | (a,b) \in R, b \notin \omega \text{ and } a \in \omega\} \\ B &= \{a | (a,b) \in O \text{ or } (b,a) \in I\} \end{aligned}$$

Note: "I" means "inward attacks", "O" means "outward attacks" and "B" means "border arguments".

Example 7 *Figure 3.6 represents the cluster structures in γ_1 and γ_2. As an example the cluster structure κ_1 is defined as $\langle \gamma_1, \emptyset, \{(g,h)\}, \{g\} \rangle$.*



(a) Clusters of γ_1 (b) Clusters of γ_2

Figure 3.5: Cluster partition

3.4 Computing the labellings

Once the cluster structures are created, we compute the labellings for the given semantics for all possible cases. The same treatment described below will be applied simultaneously for the different clusters.

The notion of AF clusters as defined in this paper is very similar to the notion of I/O Argumentation Framework introduced by [4]. Following the status of the arguments that, from outside, attack the cluster, a set of labellings will be computed and, as a consequence, these labellings will imply a status for the sources of attacks going outside of the cluster.

We call "context" a labelling of the cluster inward attack sources.

Definition 36 *(Context). Let $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster structure, and $S_I = \{a | (a,b) \in I\}$ be the inward attack sources of κ. A context μ of κ is a labelling of S_I.*

Given that an argument can be labelled in, out or und, in the worst case there will be $3^{|I|}$ contexts. The exact number of contexts is $3^{|\{a|(a,b)\in I\}|}$.

Each context induces an AF denoted by $\gamma_i{}'$ from the original AF cluster γ_i. Let $\kappa = \langle \gamma_i, I, O, B \rangle$ be a cluster structure, S be the set of I-sources, T be the set of I-targets and μ be a context of κ. Here is how $\gamma_i{}'$ is induced for a particular context μ:

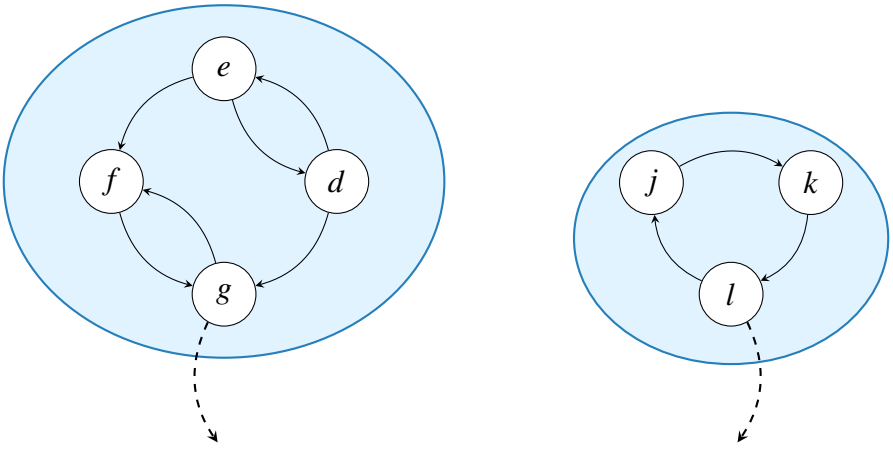
1. $\gamma_i{}'$ receives a copy of γ_i
2. $\forall s \in S$ s.t. $\mu(s) = \mathit{in}, \forall t \in \{t | (s,t) \in T\}$, t is removed from $\gamma_i{}'$ with all the attacks that have t as endpoint.
3. $\forall s \in S$ s.t. $\mu(s) = \mathit{und}, \forall t \in \{t | (s,t) \in T, \nexists (s',t)$ s.t. $\mu(s') = \mathit{in}\}$, the attack (t,t) is added to $\gamma_i{}'$.

If $\mu(s) = \mathit{out}$ there is nothing to do as the attack would have no effect.

Formally here is how induced AF are defined.

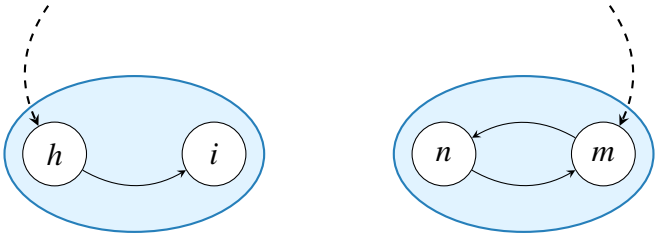
Definition 37 *(Induced AF). Let $\gamma = \langle A, R \rangle$ be an AF, $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster structure, and μ be a context of κ. The induced AF γ' of κ (or induced from γ) under the context μ is defined as following:*

$$\gamma' = \langle A', R' \rangle$$



(a) κ_1: first cluster of γ_1

(b) κ_3: first cluster of γ_2



(c) κ_2: second cluster of γ_1

(d) κ_4: second cluster of γ_2

Figure 3.6: Clusters structures of the components

With:

- $A' = A \setminus D$
- $D = \{a | a \in A \text{ and } (s,a) \in I \text{ and } s \in \mathit{in}(\mu)\}$
- $R' = (R \setminus \{(s,t) | s \in D \text{ or } t \in D\}) \cup \{(a,a) | (s,a) \in I \text{ and } s \in \mathit{und}(\mu)\}$

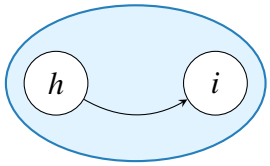
Example 8 *Figure 3.7 represents the three AFs induced from κ_2.*

For all those induced AFs, we compute the labellings corresponding to the given semantics.

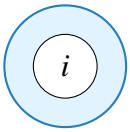
Definition 38 *(Induced labellings). Let $\gamma = \langle A, R \rangle$ be an AF, $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster structure, μ be a context of κ. Let γ' be the induced AF of κ under the context μ, D be the set of arguments such that $D = \{a | a \in A \text{ and } (s,a) \in I \text{ and } s \in \mathit{in}(\mu)\}$ and ℓ^D is the labelling defined as $\{(a, \mathtt{out}) | a \in D\}$.*

The set of induced labellings $\mathscr{L}_\sigma^{\mu(\kappa)}$ of γ under the context μ is defined as following:

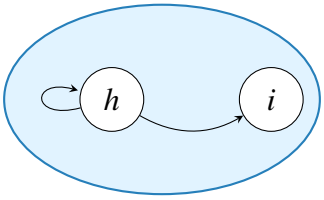
$$\mathscr{L}_\sigma^{\mu(\kappa)} = \{\ell \cup \ell^D | \ell \in \mathscr{L}(\gamma')\}$$



(a) g is labelled $\texttt{out}$, (g,h) is inactive



(b) g is labelled $\texttt{in}$, (g,h) is active



(c) g is labelled $\texttt{und}$, (g,h) is undecided

Figure 3.7: AFs induced from κ_2

At this step, the computation of the labellings can be done in parallel for each induced AF γ'.

Note: In our experiments, the computation of the labellings of an induced AF will be realized using an existing AF solver.

Then the labellings of the initial AF are obtained by reunifying the induced labellings of each cluster. In order to do that, we must to associate a "configuration" ξ with each induced labelling ℓ. This configuration expresses under which conditions an induced labelling, from a given cluster, can be reunified with another one from a neighbour cluster. This configuration is a 5-value labelling on the cluster border arguments (*i.e.* $\forall a \in B$).

Definition 39 *(Configuration ξ). Let $\gamma = \langle A, R\rangle$ be an AF, $\kappa = \langle \gamma, I, O, B\rangle$, μ be a context of κ, and $\ell \in \mathscr{L}_\sigma^{\mu(\kappa)}$ be a computed labelling of κ under μ.*

Given ℓ, a configuration is a total function $\xi : B \to \{\texttt{in}, \texttt{out}, \texttt{iout}, \texttt{und}, \texttt{iund}\}$ such that:

$$\xi : a \in B \mapsto \begin{cases} \texttt{in} & \text{if } \ell(a) = \texttt{in} \\ \texttt{out} & \text{if } \ell(a) = \texttt{out} \text{ and } \exists (b,a) \in R \text{ s.t. } \ell(b) = \texttt{in} \\ \texttt{iout} & \text{if } \ell(a) = \texttt{out} \text{ and } \nexists (b,a) \in R \text{ s.t. } \ell(b) = \texttt{in} \\ \texttt{und} & \text{if } \ell(a) = \texttt{und} \text{ and } \nexists (b,a) \in I \text{ s.t. } \mu(b) = \texttt{und} \\ \texttt{iund} & \text{if } \ell(a) = \texttt{und} \text{ and } \exists (b,a) \in I \text{ s.t. } \mu(b) = \texttt{und} \end{cases}$$

In words, for an argument a:

- $\xi(a) = \texttt{in}$ means that a is successfully attacked neither from outside nor from inside the cluster.
- $\xi(a) = \texttt{out}$ means a is legally $\texttt{out}$ from cluster point of view.
- $\xi(a) = \texttt{iout}$ means that a is illegally $\texttt{out}$ from the cluster point of view.
- $\xi(a) = \texttt{und}$ means that a is is legally $\texttt{und}$ from cluster point of view.

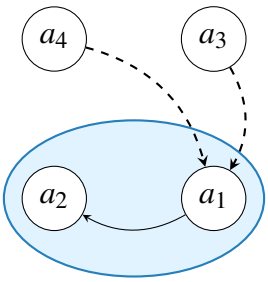


Figure 3.8: Example of the interest of the 5-value labelling.

- $\xi(a) = \texttt{iund}$ means that a is illegally $\texttt{und}$ from cluster point of view.

Note: We do not need a value to represent the fact that an argument is illegaly $\texttt{in}$ *because if a border argument is* $\texttt{in}$ *then all its attackers must be* $\texttt{out}$*. As defined in Section 3.5 a simple constraint on endpoint attack labels is sufficient to ensure only such reunifications.*

That is not the case for the values $\texttt{out}$ *and* $\texttt{und}$*.*

Let illustrate that fact. Consider the cluster structure shown in Figure 3.8. Let say that because of a certain context μ a_1 *is labelled* $\texttt{out}$*. From the cluster point of view* a_3 *could be labelled* $\texttt{in}$*,* $\texttt{out}$ *or* $\texttt{und}$*, and the same for* a_4*. Indeed, all these endpoint attack labels couples are valid. An extra constraint is needed to ensure that at least one between* a_3 *and* a_4 *is labelled* $\texttt{in}$*. That why we need to differentiate* $\texttt{out}$ *and* $\texttt{iout}$*.*

The same reasoning shows that we also need two undecided states ($\texttt{und}$ *and* $\texttt{iund}$*).*

Example 9 *Here is the result according to the* complete *semantics for our running example.*

For κ_1 *we have only one context* $\mu_1^{\kappa_1} = \emptyset$ *that gives the labellings and their induced configurations shown in Figure 3.9.*

	$\ell_1^{\kappa_1}$	$\xi_1^{\kappa_1}$	$\ell_2^{\kappa_1}$	$\xi_2^{\kappa_1}$	$\ell_3^{\kappa_1}$	$\xi_3^{\kappa_1}$
d	$\texttt{out}$		$\texttt{in}$		$\texttt{und}$	
e	$\texttt{in}$		$\texttt{out}$		$\texttt{und}$	
f	$\texttt{out}$		$\texttt{in}$		$\texttt{und}$	
g	$\texttt{in}$	$\texttt{in}$	$\texttt{out}$	$\texttt{out}$	$\texttt{und}$	$\texttt{und}$

Figure 3.9: κ_1 labellings and configurations under $\mu_1^{\kappa_1}$.

For κ_2 *we have three contexts:* $\mu_1^{\kappa_2} = \{g = \texttt{out}\}$*,* $\mu_2^{\kappa_2} = \{g = \texttt{in}\}$ *and* $\mu_1^{\kappa_3} = \{g = \texttt{und}\}$*. Figure 3.10 gives their corresponding labellings and induced configurations.*

For κ_3 *we have only one context* $\mu_1^{\kappa_3} = \emptyset$ *that gives the labellings and their induced configurations shown in Figure 3.11.*

For κ_4 *we have three contexts:* $\mu_1^{\kappa_4} = \{l = \texttt{in}\}$*,* $\mu_2^{\kappa_4} = \{l = \texttt{out}\}$ *and* $\mu_1^{\kappa_4} = \{l = \texttt{und}\}$*. Figure 3.12 gives their corresponding labellings and induced configurations.*

Notice that $\xi_1^{\kappa_4}(m) \neq \texttt{iout}$ *because* $\ell_1^{\kappa_4}(n) = \texttt{in}$ *and the attack* (n,m) *exists in* κ_4*.*

	$\ell_1^{\kappa_2}$	$\xi_1^{\kappa_2}$
h	in	in
i	out	

(a) Under $\mu_1^{\kappa_2}$

	$\ell_2^{\kappa_2}$	$\xi_2^{\kappa_2}$
h	out	$iout$
i	in	

(b) Under $\mu_2^{\kappa_2}$

	$\ell_3^{\kappa_2}$	$\xi_3^{\kappa_2}$
h	und	$iund$
i	und	

(c) Under $\mu_3^{\kappa_2}$

Figure 3.10: κ_2 labellings and configurations under $\mu_1^{\kappa_2}$.

	$\ell_1^{\kappa_3}$	$\xi_1^{\kappa_3}$
j	und	
k	und	
l	und	und

Figure 3.11: κ_3 labellings and configurations under $\mu_1^{\kappa_3}$.

	$\ell_1^{\kappa_4}$	$\xi_1^{\kappa_4}$
m	out	out
n	in	

(a) Under $\mu_1^{\kappa_4}$

	$\ell_2^{\kappa_4}$	$\xi_2^{\kappa_4}$	$\ell_3^{\kappa_4}$	$\xi_3^{\kappa_4}$
m	in	in	out	out
n	out		in	

(b) Under $\mu_2^{\kappa_4}$

	$\ell_4^{\kappa_4}$	$\xi_4^{\kappa_4}$	$\ell_5^{\kappa_4}$	$\xi_5^{\kappa_4}$
m	out	out	und	$iund$
n	in		und	

(c) Under $\mu_3^{\kappa_4}$

Figure 3.12: κ_4 labellings and configurations under $\mu_1^{\kappa_4}$.

After that we have computed the different labellings and their corresponding configuration, we keep only the "distinct labellings" with their "merge configurations".

Definition 40 *(Distinct labelling set). Let $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster, let $\mathscr{L}^{\kappa} = \{\ell_1^{\kappa}, ..., \ell_n^{\kappa}\}$ be the set of labellings computed from κ, and $\mathscr{L}_{\mathscr{D}}^{\kappa}$ be the distinct labelling set of κ.*

$\mathscr{L}_{\mathscr{D}}^{\kappa}$ is defined as following:

$$\mathscr{L}_{\mathscr{D}}^{\kappa} = \{\ell_i^{\kappa} | \ell_i^{\kappa} \in \mathscr{L}^{\kappa} \text{ and } \nexists \ell_j^{\kappa} \in \mathscr{L}^{\kappa} \text{ s.t. } \ell_j^{\kappa} = \ell_i^{\kappa} \text{ and } j < i\}$$

Example 10 *For κ_4, $\mathscr{L}_{\mathscr{D}}^{\kappa_4} = \{\ell_1^{\kappa_4}, \ell_2^{\kappa_4}, \ell_5^{\kappa_4}\}$.*

Notice it is possible for a labelling to have several and different configurations. These configurations can only differ on und and $iund$ labels. As an example, consider the Figure 3.13.

Example 11 *In no case a can be labelled in. Let thus consider only the contexts of the right cluster that can possibly lead to valid reunified labellings, which are: $\{(a, out)\}$ and $\{(a, und)\}$. In both cases, we have a unique labelling $\{(b, und)\}$. Nevertheless, considering the configurations, we obtain two distinct ones. For the first context, we have: $\{(b, und)\}$ using the forth rule of the configuration definition (Definition 39) and $\{(b, iund)\}$ using the last rule.*

Given that it is possible for a labelling to have several and different configurations, we introduce the notion of "merge configuration".

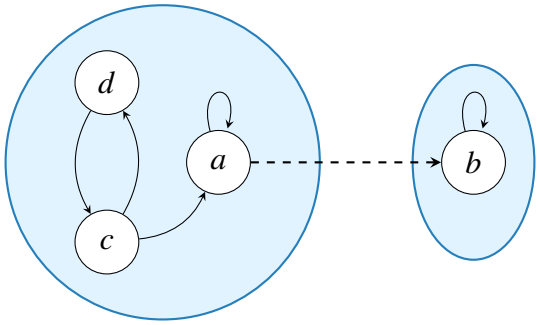


Figure 3.13: Illustration of merge configuration

Definition 41 *(Merge configuration). Let $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster, let $\mathscr{L}^{\kappa} = \{\ell_1^{\kappa}, ..., \ell_n^{\kappa}\}$ be the set of labellings and $\mathscr{C}^{\kappa} = \{\xi_1^{\kappa}, ..., \xi_n^{\kappa}\}$ the set of their corresponding configurations computed from κ, let $\ell_i^{\kappa} \in \mathscr{L}^{\kappa}$ be a labelling and $\mathscr{C}_{\ell_i^{\kappa}} = \{\xi_j^{\kappa} | \xi_j^{\kappa} \in \mathscr{C}^{\kappa} \text{ s.t. } \ell_j^{\kappa} = \ell_i^{\kappa}\}$ be the set of all its possible configurations. Let $\xi \in \mathscr{C}_{\ell_i^{\kappa}}$ be a possible configuration of ℓ_i^{κ}.*

The merge configuration $\xi_{\ell_i^{\kappa}}$ of ℓ_i^{κ} is defined as follows:

$$\forall a \in B, \xi_{\ell_i^{\kappa}}(a) = \begin{cases} in & \text{if } \ell_i^{\kappa}(a) = in \\ out & \text{if } \ell_i^{\kappa}(a) = out \text{ and } \exists (b,a) \in \gamma \text{ s.t. } \ell(b) = in \\ iout & \text{if } \ell_i^{\kappa}(a) = out \text{ and } \exists (b,a) \in \gamma \text{ s.t. } \ell(b) = in \\ und & \text{if } \ell_i^{\kappa}(a) = und \text{ and } \exists \xi \in \mathscr{C}_{\ell_i^{\kappa}} \text{ s.t. } \xi(a) = und \\ iund & otherwise \end{cases}$$

The merge configuration as defined is the most flexible configuration of a given labelling. It ensures all the requirements for a good reunification without adding unwanted restrictions.

Definition 42 *(Distinct labelling/configuration set). Let $\kappa = \langle \gamma, I, O, B \rangle$ be a cluster, let $\mathscr{L}_{\mathscr{D}}^{\kappa}$ be the distinct labelling set of κ, and let $\mathscr{D}_{\ell/c}^{\kappa}$ be its distinct labelling/configuration set.*

$\mathscr{D}_{\ell/c}^{\kappa}$ is defined as following:

$$\mathscr{D}_{\ell/c}^{\kappa} = \{(\ell^{\kappa}, \xi_{\ell^{\kappa}}) | \ell^{\kappa} \in \mathscr{L}_{\mathscr{D}}^{\kappa}\}$$

Example 12 *This step will affect only the cluster κ_4 as $\ell_1^{\kappa_4} = \ell_3^{\kappa_4} = \ell_4^{\kappa_4}$. The new set of labelling/configuration of κ_4 is shown on Figure 3.14.*

	$\ell_1'^{\kappa_4}$	$\xi_1'^{\kappa_4}$	$\ell_2'^{\kappa_4}$	$\xi_2'^{\kappa_4}$	$\ell_3'^{\kappa_4}$	$\xi_3'^{\kappa_4}$
m	*out*	*out*	*in*	*in*	*und*	*iund*
n	*in*		*out*		*und*	

Figure 3.14: κ_4 labellings and configurations.

We can notice after this filtering and merging process that:

- One context can give several labellings.
- From a labelling is induced one and only one merge configuration.
- Several labellings can induce the same merge configuration.

3.5 Reunifying the results

The last step consists in reunifying the labelling parts previously computed. Labelling parts can be reunified if and only if their corresponding configurations are "compatible" together. To express that "compatibily" we transform that reunifying problem into a *constraint satisfaction problem* (CSP).

Here are the four steps of the transformation process :

1. For each cluster κ_i, a variable V_{κ_i} is created. For each of them, the domain is the set of their *distinct* computed labellings (*i.e.* $\forall \ell \in \mathscr{L}_{\mathscr{D}}^{\kappa}$).

2. For each border argument a_j, a variable V_{a_j} is created with a domain corresponding to their possible labels, *i.e.* $\{\texttt{in}, \texttt{out}, \texttt{und}\}$.

3. For each inter-cluster attack (a,b), a constraint is added with the following set of valid tuples:
   $\{(a=\texttt{in}, b=\texttt{out}), (a=\texttt{out}, b=\texttt{in}), (a=\texttt{out}, b=\texttt{out}),$
   $(a=\texttt{out}, b=\texttt{und}), (a=\texttt{und}, b=\texttt{out}), (a=\texttt{und}, b=\texttt{und})\}$

   Let ℓ be a value of the domain of V_{κ_i}, let ξ_ℓ be the corresponding merged configuration and $\kappa_i = \langle \gamma, I, O, B \rangle$ be the corresponding cluster.

4. For each ℓ in V_{κ_i} domain:

   (a) Constraints are added to map the labelling with its corresponding configuration. The constraints are defined as following:

   $$\forall a_j \in B, \quad \begin{array}{l} (V_{\kappa_i} = \ell \wedge \xi_\ell(a_j) = \texttt{in}) \implies V_{a_j} = \texttt{in} \\ (V_{\kappa_i} = \ell \wedge (\xi_\ell(a_j) = \texttt{out} \vee \xi_\ell(a_j) = \texttt{iout})) \implies V_{a_j} = \texttt{out} \\ (V_{\kappa_i} = \ell \wedge (\xi_\ell(a_j) = \texttt{und} \vee \xi_\ell(a_j) = \texttt{iund})) \implies V_{a_j} = \texttt{und} \end{array}$$

   (b) Constraints are added for all arguments labelled $\texttt{iout}$ in ξ:
   $\forall a_j \in \{a | \xi_\ell(a) = \texttt{iout}\}, \quad V_{\kappa_i} = \ell \implies \exists (a_k, a_j) \in I$ s.t. $V_{a_k} = \texttt{in}$

   (c) Constraint are added for all arguments labelled $\texttt{iund}$ in ξ:
   $\forall a_j \in \{a | \xi_\ell(a) = \texttt{iund}\}, \quad V_{\kappa_i} = \ell \implies \exists (a_k, a_j) \in I$ s.t. $V_{a_k} = \texttt{und}$

Note: The constraints have to be seen as declarative rules. For example the rule: $V_{\kappa_i} = \ell \implies \exists (a_k, a_j) \in I$ *s.t.* $V_{a_k} = \texttt{und}$ *as to be understand as "If the variable* V_{κ_i} *has the value* ℓ*, there must be a variable corresponding to one of the attackers of* a_j *that takes the value* $\texttt{und}$*".*

The solutions of that CSP modelling are the set of compatible labellings (corresponding to values of the V_{κ_i} variables).

Example 13 *As example let create the CSP modelisation for the reunification of* γ_1.

Let $\Psi_{\gamma_1} = \langle X, D, C \rangle$ *be that model.* Ψ_{γ_1} *is defined as following:*

- $X = \{V_{\kappa_1}, V_{\kappa_2}, V_g, V_h\}$
- $D = \{$
  $D(V_{\kappa_1}) = \{\ell_1^{\kappa_1}, \ell_2^{\kappa_1}, \ell_3^{\kappa_1}\}$,
  $D(V_{\kappa_2}) = \{\ell_1^{\kappa_2}, \ell_2^{\kappa_2}, \ell_3^{\kappa_2}\}$,
  $D(V_g) = \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$,
  $D(V_h) = \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$
  $\}$
- $C = \{c_1, c_2, c_3, c_4, c_5\}$ *is a set of constraints, with*
  - c_1 *being the constraint that expresses the attack relation from g to h, corresponding to Step 3,*
  - c_2 *being the constraint expressing the fact that the labellings of* κ_1 *impose a label on each of its border arguments (i.e. on g), corresponding to Step 4a,*
  - c_3 *being the constraint expressing the fact that the labellings of* κ_2 *impose a label on each of its border arguments (i.e. on h), corresponding to Step 4a,*
  - c_4 *being the constraint expressing the fact that* $\ell_2^{\kappa_2}$ *can only be reunified with a labelling of* κ_1 *in which g is labelled* `in`*, corresponding to Step 4b,*
  - c_5 *being the constraint expressing the fact that* $\ell_3^{\kappa_2}$ *can only be reunified with a labelling of* κ_1 *in which g is labelled* `und`*, corresponding to Step 4c.*

`Note:` c_4 and c_5 are constraints only for precise labellings of κ_2. c_4 and c_5 must allow g being labelled with any label if the reunification is about another labelling of κ_2.

c_1 accepts only the following tuples:

- $(V_g = \mathtt{in}, V_h = \mathtt{out})$
- $(V_g = \mathtt{out}, V_h = \mathtt{in})$
- $(V_g = \mathtt{out}, V_h = \mathtt{out})$
- $(V_g = \mathtt{out}, V_h = \mathtt{und})$
- $(V_g = \mathtt{und}, V_h = \mathtt{out})$
- $(V_g = \mathtt{und}, V_h = \mathtt{und})$

c_2 accepts only the following tuples (see Figure 3.9):

- $(V_{\kappa_1} = \ell_1^{\kappa_1}, V_g = \mathtt{in})$
- $(V_{\kappa_1} = \ell_2^{\kappa_1}, V_g = \mathtt{out})$
- $(V_{\kappa_1} = \ell_3^{\kappa_1}, V_g = \mathtt{und})$

c_3 accepts only the following tuples (see Figure 3.10):

- $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_h = \mathtt{in})$
- $(V_{\kappa_2} = \ell_2^{\kappa_2}, V_h = \mathtt{iout})$

– $(V_{\kappa_2} = \ell_3^{\kappa_2}, V_h = iund)$

c_4 accepts only the following tuples:

– $(V_{\kappa_2} = \ell_2^{\kappa_2}, V_g = in)$
– $(V_{\kappa_2} = \ell_3^{\kappa_2}, V_g = in)$
– $(V_{\kappa_2} = \ell_3^{\kappa_2}, V_g = out)$
– $(V_{\kappa_2} = \ell_3^{\kappa_2}, V_g = und)$
– $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_g = in)$
– $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_g = out)$
– $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_g = und)$

c_5 accepts only the following tuples:

– $(V_{\kappa_2} = \ell_3^{\kappa_2}, V_g = und)$
– $(V_{\kappa_2} = \ell_2^{\kappa_2}, V_g = in)$
– $(V_{\kappa_2} = \ell_2^{\kappa_2}, V_g = out)$
– $(V_{\kappa_2} = \ell_2^{\kappa_2}, V_g = und)$
– $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_g = in)$
– $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_g = out)$
– $(V_{\kappa_2} = \ell_1^{\kappa_2}, V_g = und)$

`Note:` c_4 and c_5 have both to be respected. As a consequence, the valid tuples concerning V_{κ_2} and V_g are the ones which are both in c_4's valid tuples and c_5's valid tuples. So the second and third tuples accepted by c_4 and the third and fourth tuples accepted by c_5 will be useless.

The solutions are:

- $\{e = in, g = in, f = out, d = out, h = out, i = in\}$
  (corresponding to the configurations $\xi_1^{\kappa_1} = \{g = in\}$ and $\xi_2^{\kappa_2} = \{h = iout\}$)
- $\{e = out, g = out, f = in, d = in, h = in, i = out\}$
  (corresponding to the configurations $\xi_2^{\kappa_1} = \{g = out\}$ and $\xi_1^{\kappa_2} = \{h = in\}$)
- $\{e = und, g = und, f = und, d = und, h = und, i = und\}$
  (corresponding to the configurations $\xi_3^{\kappa_1} = \{g = und\}$ and $\xi_3^{\kappa_2} = \{h = iund\}$)

In the same way for γ_2 we obtain the following results:

- $\{j = und, k = und, l = und, m = und, n = und\}$
  (corresponding to the configurations $\xi_1^{\kappa_3} = \{l = und\}$ and $\xi_3'^{\kappa_4} = \{m = iund\}$)

- $\{j = und, k = und, l = und, m = out, n = in\}$
  (corresponding to the configurations $\xi_1^{\kappa_3} = \{l = und\}$ and $\xi_1'^{\kappa_4} = \{m = out\}$)

To finish, the labellings of the whole AF are reunified. All combinations of these labellings, combined with the one concerning the "trivial part" of the AF, form a global labelling.

Example 14 *In Figure 3.15 you can see the complete labellings thus obtained.*

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6
a	*in*	*in*	*in*	*in*	*in*	*in*
b	*out*	*out*	*out*	*out*	*out*	*out*
c	*out*	*out*	*out*	*out*	*out*	*out*
d	*out*	*out*	*in*	*in*	*und*	*und*
e	*in*	*in*	*out*	*out*	*und*	*und*
f	*out*	*out*	*in*	*in*	*und*	*und*
g	*in*	*in*	*out*	*out*	*und*	*und*
h	*out*	*out*	*in*	*in*	*und*	*und*
i	*in*	*in*	*out*	*out*	*und*	*und*
j	*und*	*und*	*und*	*und*	*und*	*und*
k	*und*	*und*	*und*	*und*	*und*	*und*
l	*und*	*und*	*und*	*und*	*und*	*und*
m	*und*	*out*	*und*	*out*	*und*	*out*
n	*und*	*in*	*und*	*in*	*und*	*in*

Figure 3.15: Complete labellings

3.6 Algorithms

The general form of the distributed algorithm proposed in this paper is given in Algorithm 1 that uses Algorithm 2 or Algorithm 3 following the computed semantics.

3.7 Completeness and soundness

In this section we will prove that the algorithms work well. More precisely, we will prove that the algorithms give all the expected labellings for the complete, stable and preferred semantics; this is the notion of completeness, and will also prove that the algorithms give only good labellings for the semantics complete, stable and preferred; this is the notion of soundness.

In a first step we will give underlying notions of semantics in order to make the proofs, then we will prove the soundness and completeness of the algorithms.

3.7.1 Prerequisite notions

In [4], Baroni et al. introduce several notions and proved semantics properties that are useful to prove that our proposed algorithms are sound and complete. The most important about this section is to understand what is a fully-decomposable semantics and what is a top-down decomposable semantics. Before going in the formal definitions of them, we will give the intuition of what they represent.

A semantics will be a fully-decomposable or top-down decomposable semantics if for any AF and any partition of a given AF, it is possible to reconstruct all the labellings of the whole AF by combining the labellings (under the same semantic) of the partition parts.

To be more precise, the difference between a top-down decomposable semantics and a fully decomposable one is that for a top-down decomposable one, when doing this process of labelling part reunification all the semantics labellings will be found but it is also possible to obtain non correct labellings, whereas, for a fully-decomposable all and only the correct semantics labellings will be obtained.

Algorithm 1 AFDivider algorithm

Input: Let $\Gamma = \langle A, R \rangle$ be an AF and σ be a semantics

Output: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\Gamma)}$: the set of labellings of Γ under the semantics σ

Local variables:

CCSet: the set of connected components (disjoints sub-AFs after removing the grounded influence set)

ClustSet: the set of cluster structures for each connected component γ_i

ℓ_{gr}: the grounded labelling of Γ

$\mathscr{L}_\sigma^{\gamma_i}$: the set of labellings of the component γ_i under the semantics σ

1: $\ell_{gr} \leftarrow ComputeGroundedLabelling(\Gamma)$
2: $CCSet \leftarrow SplitConnectedComponents(\Gamma, \ell_{gr})$
3: **for all** $\gamma_i \in CCSet$ **do in parallel**
4: $\quad ClustSet \leftarrow ComputeClusters(\gamma_i)$
5: $\quad \mathscr{L}_\sigma^{\gamma_i} \leftarrow ComputeComponentLabellings_\sigma(ClustSet)$ `// See Algorithms 2 and 3`
6: **end for**
7: $\mathscr{L}_\sigma \leftarrow \{\ell_{gr}\} \times \prod_{\gamma_i \in CCSet} \mathscr{L}_\sigma^{\gamma_i}$
8: **return** $\mathscr{L}_\sigma$

Algorithm 2 $ComputeComponentLabellings_\sigma$: Component labelling algorithm for stable and complete semantics.

Input: Let *ClustSet* be a set of cluster structures corresponding to a component γ and σ be the stable or the complete semantics.

Output: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\gamma)}$: the set of labellings of γ under the semantics σ

Local variables:

$\mathscr{L}_\sigma^{\kappa_j}$: the set of labellings of the cluster structure κ_j under the semantics σ

1: **for all** $\kappa_j \in ClustSet$ **do in parallel**
2: $\quad \mathscr{L}_\sigma^{\kappa_j} \leftarrow ComputeClusterLabellings_\sigma(\kappa_j)$
3: **end for**
4: $\mathscr{L}_\sigma \leftarrow ReunifyComponentLabellings(\bigcup_{\kappa_j \in ClustSet} \mathscr{L}_\sigma^{\kappa_j}, ClustSet)$
5: **return** $\mathscr{L}_\sigma$

Algorithm 3 *ComputeComponentLabellings*$_{pr}$: Component labelling algorithm for the preferred semantics

Input: Let *ClustSet* be a set of cluster structures corresponding to a component γ
Output: $\mathscr{L}_{pr} \in 2^{\mathscr{L}(\gamma)}$: the set of labellings of γ under the preferred semantics

Local variables:

$\mathscr{L}_{pr}^{\kappa_j}$: the set of labellings of the cluster structure κ_j under the preferred semantics
$\mathscr{L}$: the set of the all labellings reunified from the different clusters

1: **for all** $\kappa_j \in ClustSet$ **do in parallel**
2: $\quad \mathscr{L}_{pr}^{\kappa_j} \leftarrow ComputeClusterLabellings_{pr}(\kappa_j)$
3: **end for**
4: $\mathscr{L} \leftarrow ReunifyComponentLabellings(\bigcup_{\kappa_j \in ClustSet} \mathscr{L}_{pr}^{\kappa_j}, ClustSet)$
5: $\mathscr{L}_{pr} \leftarrow \{\ell | \ell \in \mathscr{L} \text{ s.t. } \nexists \ell' \in \mathscr{L} \text{ s.t. } \mathtt{in}(\ell) \subset \mathtt{in}(\ell')\}$
6: **return** $\mathscr{L}_{pr}$

All the formal definitions and propositions that follow lead to these properties and come from [4].

Definition 43 *(Labelling restriction $\downarrow$). Let ℓ be a labelling. Let S be a set of arguments. The* restriction *of ℓ to S denoted as $\ell \downarrow_S$ is defined as $\ell \cap (S \times \{\mathtt{in}, \mathtt{out}, \mathtt{und}\})$.*

Example 15 *The restriction of ℓ_6 (shown in Figure 3.15) to $\{a,b,c\}$ is $\ell_6 \downarrow_{\{a,b,c\}} = \{a = \mathtt{in}, b = \mathtt{out}, c = \mathtt{out}\}$.*

Definition 44 *(Input arguments and conditioning relation). Let $\Gamma = \langle A, R \rangle$ be an AF and $S \subseteq A$ be a set. The input of S, denoted as S^{inp}, is the set $\{b \in A \setminus S | \exists a \in S, (b,a) \in R\}$.*

The conditioning relation of S, denoted as S^R, is defined as $R \cap (S^{inp} \times S)$.

The notion of AF with input defined below is very similar to the notion of cluster structure we defined with the difference that the AF with input has a fixed labelling on the input arguments (*i.e.* a fixed context).

Definition 45 *(AF with input and local function). An argumentation framework with input is a tuple $\langle \Gamma, \mathscr{I}, \ell^{\mathscr{I}}, R_{\mathscr{I}} \rangle$, including an argumentation framework $\Gamma = \langle A, R \rangle$, a set of arguments $\mathscr{I}$ such that $\mathscr{I} \cap A = \emptyset$, a labelling $\ell^{\mathscr{I}}$ of the elements of $\mathscr{I}$ and a relation $R_{\mathscr{I}} \subseteq \mathscr{I} \times A$.*

A local function $\mathscr{F}$ assigns to any argumentation framework with input a (possibly empty) set of labellings of Γ, i.e. $\mathscr{F}(\Gamma, \mathscr{I}, \ell^{\mathscr{I}}, R_{\mathscr{I}}) \in 2^{\mathscr{L}(\Gamma)}$.

Given an AF with input denoted by Γ, its standard argumentation framework denoted by Γ' is an AF that simulates the conditioning labelling of the input arguments of Γ. Computing the labelling of this standard argumentation framework gives thus, indirectly, the labellings of the sub-AF we are interested in, and under a certain conditioning due to the input arguments labelling. So this standard AF Γ' corresponds to the AF Γ in which some fictive arguments and interactions are added in order to justify the labellings.

Following is the formal definition:

Definition 46 *(Standard argumentation framework). Given an argumentation framework with input* $\langle \Gamma, \mathscr{I}, \ell^{\mathscr{I}}, R_{\mathscr{I}} \rangle$*, the standard argumentation framework w.r.t.* $\langle \Gamma, \mathscr{I}, \ell^{\mathscr{I}}, R_{\mathscr{I}} \rangle$ *is defined as* $\Gamma' = \langle A \cup \mathscr{I}', R \cup R'_{\mathscr{I}} \rangle$*, where* $\mathscr{I}' = \mathscr{I} \cup \{a' | a \in \mathscr{I} \cap \texttt{out}(\ell^{\mathscr{I}})\}$ *and* $R'_{\mathscr{I}} = R_{\mathscr{I}} \cup \{(a', a) | a \in \mathscr{I} \cap \texttt{out}(\ell^{\mathscr{I}})\} \cup \{(a,a) | a \in \mathscr{I} \cap \texttt{und}(\ell^{\mathscr{I}})\}$.[3]

Note: By definition, the labellings of the standard AF Γ' *restricted to the inputs of the AF* Γ *are exactly the labellings* $\ell^{\mathscr{I}}$ *given in* Γ.

Given an AF with input, the canonical local function is simply a function that gives the set of labellings under a certain semantics of the sub-AF we are interested in (*i.e.* the input arguments and the other fictive arguments created are not in these labellings).

Definition 47 *(Canonical local function). Let* $\Gamma = \langle A, R \rangle$ *be an AF,* σ *be a semantics,* $\langle \Gamma, \mathscr{I}, \ell^{\mathscr{I}}, R_{\mathscr{I}} \rangle$ *be an AF with input, and* Γ' *be its standard argumentation framework. The canonical local function* $\mathscr{F}_\sigma$ *is the local function such that* $\mathscr{F}_\sigma(\Gamma, \mathscr{I}, \ell^{\mathscr{I}}, R_{\mathscr{I}}) = \{\ell \downarrow_A | \ell \in \mathscr{L}_\sigma(\Gamma')\}$.

Definition 48 *(Semantics fully decomposability). A semantics* σ *is* fully decomposable *(or simply* decomposable*) if and only if there is a local function* $\mathscr{F}$ *such that for every AF* $\Gamma = \langle A, R \rangle$ *and every partition* $\Omega = \{\omega_1, ..., \omega_n\}$ *of A,*

$$\mathscr{L}_\sigma(\Gamma) = \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} \mid \forall i,\ \ell^{\omega_i} \in \mathscr{F}(\Gamma \downarrow_{\omega_i},\ \omega_i^{inp},\ (\bigcup_{j \in \{1,...,n\}\ s.t.\ j \neq i} \ell^{\omega_j}) \downarrow_{\omega_i^{inp}},\ \omega_i^R)\}.$$

Definition 49 *(Initial argument). Let* $\Gamma = \langle A, R \rangle$ *be an AF, and* $b \in A$ *be an argument.* b *is an initial argument of* Γ *if there is no argument in* Γ *attacking* b*. In graph theory, it is called source node.*

Definition 50 *(Complete-compatibility). A semantics* σ *is complete-compatible if and only if the following conditions hold:*

1. *For any AF* $\Gamma = \langle A, R \rangle$*, every labelling* $\ell \in \mathscr{L}_\sigma(\Gamma)$ *satisfies the following conditions:*
   - *if* $a \in A$ *is initial, then* $\ell(a) = \texttt{in}$
   - *if* $b \in A$ *and there is an initial argument in A which attacks b, then* $\ell(b) = \texttt{out}$
   - *if* $c \in A$ *is self-attacking, and there is no attacker of c besides c itself, then* $\ell(c) = \texttt{und}$
2. *For any set of arguments* $\mathscr{I}$ *and any labelling* $\ell^{\mathscr{I}}$ *of* $\mathscr{I}$*, the AF* $\Gamma' = \langle \mathscr{I}', att' \rangle$*, where* $\mathscr{I}' = \mathscr{I} \cup \{a' | a \in \mathscr{I} \cap \texttt{out}(\ell^{\mathscr{I}})\}$ *and* $att' = \{(a', a) | a \in \mathscr{I} \cap \texttt{out}(\ell^{\mathscr{I}})\} \cup \{(a,a) | a \in \mathscr{I} \cap \texttt{und}(\ell^{\mathscr{I}})\}$*, admits a unique labelling, i.e.* $|\mathscr{L}_\sigma(\Gamma')| = 1$.

Proposition 1 *The complete, stable, preferred and grounded semantics are complete-compatible.*

Proposition 2 *Given a complete-compatible semantics* σ*, if* σ *is fully decomposable then there is a unique local function satisfying the conditions of Definition 48, coinciding with the canonical local function* $\mathscr{F}_\sigma$.

Proposition 3 *The complete and stable semantics are fully decomposable.*

[3] The fictive arguments are denoted by a' in the definition of $\mathscr{I}'$ and the fictive interactions are the pairs (a', a) or (a, a) appearing in the definition of $R'_{\mathscr{I}}$.

Definition 51 *(Top-down decomposability). Let σ be a complete-compatible semantics and $\mathscr{F}_\sigma$ be the canonical local function corresponding to σ.*

σ is top-down decomposable if and only if for any AF $\Gamma = \langle A, R\rangle$ and any partition $\Omega = \{\omega_1, ..., \omega_n\}$ of A, it holds that:

$$\mathscr{L}_\sigma(\Gamma) \subseteq \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_\sigma(\Gamma \downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{\omega_j}) \downarrow_{\omega_i^{inp}}, \omega_i^R)\}$$

Proposition 4 *The complete, stable, preferred and grounded semantics are top-down decomposable.*

Definition 52 *(Partition selector). A partition selector $\mathscr{S}$ is a function receiving as input an AF $\Gamma = \langle A, R\rangle$ and returning a set of partitions of A.*

Definition 53 *(Top-down, bottom-up and fully decomposability w.r.t. a partition selector $\mathscr{S}$)*

Let $\mathscr{S}$ be a partition selector. A complete-compatible semantics σ is top-down decomposable w.r.t. $\mathscr{S}$ iff for any AF Γ and any partition $\Omega = \{\omega_1, ..., \omega_n\} \in \mathscr{S}(\Gamma)$, it holds that:

$$\mathscr{L}_\sigma(\Gamma) \subseteq \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_\sigma(\Gamma \downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{\omega_j}) \downarrow_{\omega_i^{inp}}, \omega_i^R)\}$$

A complete-compatible semantics σ is bottom-up decomposable w.r.t. $\mathscr{S}$ iff for any argumentation framework AF and any partition $\Omega = \{\omega_1, ..., \omega_n\} \in \mathscr{S}(\Gamma)$, it holds that:

$$\mathscr{L}_\sigma(\Gamma) \supseteq \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_\sigma(\Gamma \downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{\omega_j}) \downarrow_{\omega_i^{inp}}, \omega_i^R)\}$$

A complete-compatible semantics is fully decomposable (or simply decomposable) w.r.t. a partition selector $\mathscr{S}$ iff it is both top-down and bottom-up decomposable w.r.t. $\mathscr{S}$.

Definition 54 *(Path-equivalence relation). Let $G = (V, E)$ be a directed graph. The binary relation of path-equivalence between nodes, denoted as $PE_G \subseteq (V \times V)$, is defined as follows:*

- *$\forall v_i \in V, (v_i, v_i) \in PE_G$.*
- *given two distinct nodes $v_i, v_j \in V, (v_i, v_j) \in PE_G$ if and only if there is a path from v_i to v_j and a path from v_j to v_i.*

Definition 55 *(SCC-component). The strongly connected components of a directed graph G are the equivalence classes of nodes under the relation of path-equivalence. Basically, an SCC-component is a (directed) subgraph in which there is a path between each pair of its vertices.*

Let Γ be an AF. We denote by $SCCS(\Gamma)$ the set of all SCC-components of Γ.

Definition 56 *(USCC partition selector). The USCC partition selector (denoted $\mathscr{S}_{USCC}$) is the partition selector such as for any AF $\Gamma = \langle A, R\rangle$:*

$$\mathscr{S}_{USCC}(\Gamma) = \{\Omega |\ \Omega \text{ is a partition of } A \text{ and } \forall S \in SCCS(\Gamma), \exists \omega_i \in \Omega \text{ s.t. } \omega_i \cap S \neq \emptyset \implies S \subseteq \omega_i\}$$

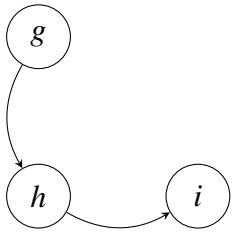
Proposition 5 *The preferred semantics is fully decomposable w.r.t. $\mathscr{S}_{USCC}$.*

3.7.2 Relation between AFs with input and cluster structures

In this section are highlighted the relation between AFs with input, introduced by Baroni et al. [4], and the cluster structures we introduced in this report. This relation identified will allow us to use decomposability properties for cluster structures.

The following example illustrates the differences between the two approaches.

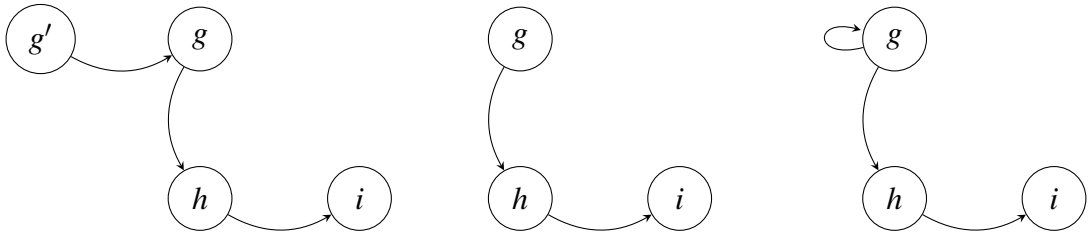
Example 16 *Consider the following AF denoted* Γ*:*



Given $\omega = \{h,i\}$*,* $\gamma = \Gamma\downarrow_\omega$ *is represented as follows:*

Considering our approach, the cluster structure for ω *is* $\kappa = \langle \gamma, I = \{(g,h)\}, O = \emptyset, B = \{h\}\rangle$*.*
Then three contexts exist: $\mu_1 = \{(g, \texttt{out})\}$*,* $\mu_2 = \{(g, \texttt{in})\}$*,* $\mu_3 = \{(g, \texttt{und})\}$*.*
And so three induced AFs can be defined (for respectively μ_1*,* μ_2*, and* μ_3*):*

Considering the approach proposed by Baroni and co., the AF with input corresponding to ω *is defined by* $\langle \gamma, \{g\}, \mu, \{(g,h)\}\rangle$ *with* μ *being either* μ_1*, or* μ_2*, or* μ_3*.*
So three standard AFs can be defined (for respectively μ_1*,* μ_2*, and* μ_3*):*



The following proposition give the correspondence between our induced AFs and the standard AFs.

Proposition 6 *Let* σ *be a complete-compatible semantics.*

Let $\Gamma = \langle A, R\rangle$ *be an AF and* $\omega \subseteq A$ *be a set of arguments. Let* $\gamma = \langle \omega, R_\gamma\rangle$ *be the restricted AF corresponding to* $\Gamma\downarrow_\omega$*.*

Let $\kappa = \langle \gamma, I = \omega^R, O = R \cap (\omega \times A \setminus \omega), B = \{a | (a,b) \in O \text{ or } (b,a) \in I\}\rangle$ *be the cluster structure corresponding to* ω*.*

Let μ *be a context of* κ*.*

The following equation holds:

$$\mathscr{L}_\sigma^{\mu(\kappa)} = \mathscr{F}_\sigma(\gamma, \omega^{inp}, \mu, \omega^R)$$

PROOF.

Let γ' be the induced AF of κ under context μ.

Let $\langle \gamma, \omega^{inp}, \mu, \omega^R \rangle$ be an AF with input and χ be its standard argumentation framework.

Let prove that:

$$\mathscr{L}_{\sigma}^{\mu(\kappa)} = \mathscr{F}_{\sigma}(\gamma, \omega^{inp}, \mu, \omega^R)$$

By definition of the induced AF (Definition 37), we have:

$$\gamma' = \langle \omega', R_{\gamma'} \rangle$$

Where:

- $D = \{a | a \in \omega \text{ and } (s,a) \in \omega^R \text{ and } s \in \mathtt{in}(\mu)\}$ *being the set of arguments attacked by an* $\mathtt{in}$*-labelled argument in* μ.
- $\omega' = \omega \setminus D$
- $R_{\gamma'} = (R_{\gamma} \cap (\omega' \times \omega')) \cup \{(a,a) | (s,a) \in \omega^R \text{ and } s \in \mathtt{und}(\mu)\}$

γ' is so the AF obtained from γ after the removal of the arguments attacked by an $\mathtt{in}$*-labelled argument of the context and after the adding of self-attacks on each argument attacked by an* $\mathtt{und}$*-labelled argument of the context.*

By definition of the standard argumentation framework (Definition 46), we have:

$$\chi = \langle \omega \cup \mathscr{I}', R_{\gamma} \cup R'_{\mathscr{I}} \rangle$$

Where:

- $\mathscr{I}' = \omega^{inp} \cup \{a' | a \in \omega^{inp} \cap \mathtt{out}(\mu)\}$
- $R'_{\mathscr{I}} = \omega^R \cup \{(a',a) | a \in \omega^{inp} \cap \mathtt{out}(\mu)\} \cup \{(a,a) | a \in \omega^{inp} \cap \mathtt{und}(\mu)\}$

Let χ_1 be the AF corresponding to $\chi \downarrow_{\omega \cup \{a | a \in \mathtt{in}(\mu)\} \cup \{a | a \in \mathtt{und}(\mu)\}}$.

Given that to obtain χ_1 from χ we just have to remove the arguments labelled $\mathtt{out}$ *in μ and those attacking them,*[4] *we have then:*

$$\{\ell \downarrow_{\omega} | \ell \in \mathscr{L}_{\sigma}(\chi_1)\} = \{\ell \downarrow_{\omega} | \ell \in \mathscr{L}_{\sigma}(\chi)\} \tag{3.1}$$

Let ω' be the set of arguments such that $\omega' = \omega \setminus D$ (as defined above).

Let χ_2 be the AF corresponding to $\chi_1 \downarrow_{\omega' \cup \{a | a \in \mathtt{und}(\mu)\}}$.

Given that to obtain χ_2 from χ_1 we just have to remove the arguments labelled $\mathtt{in}$ *in μ and those they attack,*[5] *we have then:*

$$\{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_{\sigma}(\chi_2)\} = \{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_{\sigma}(\chi_1)\} \tag{3.2}$$

Notice that to obtain $\gamma' = \langle \omega', \omega^R \cup \{(a',a) | a \in \mathtt{out}(\mu)\} \cup \{(a,a) | a \in \mathtt{und}(\mu)\} \rangle$ from χ_2 we just have to remove the arguments labelled $\mathtt{und}$ *in μ and add a self attack to each of the arguments they attack.*

[4] All these arguments are not in ω.

[5] All these arguments are not in ω'.

Considering the AF χ_2, let $U = \{a | a \in \omega' \text{ and } (b,a) \in \omega^R \text{ and } b \in \omega^{inp} \cap \texttt{und}(\mu)\}$ be the set of arguments of ω' attacked by an argument labelled `und` *in μ. Let $u \in U$ be one of these arguments.*

Given that u is attacked by an `und`*-labelled argument, u must be labelled* `und` *or* `out`*.*

Notice that having an argument labelled `und` *cannot have as consequence an argument labelled* `in` *or* `out`*. And so, if u is labelled* `out` *in some labelling of χ_2, it is not due to the set of arguments labelled* `und` *in μ.*

Knowing this, we have:

$$\mathscr{L}_\sigma(\gamma') = \{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_\sigma(\chi_2)\} \tag{3.3}$$

From Equation 3.3 and Equation 3.2, we have:

$$\mathscr{L}_\sigma(\gamma') = \{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_\sigma(\chi_1)\} \tag{3.4}$$

Let ℓ^D be the labelling of the set of arguments D defined as following: $\ell^D = \{(a, \texttt{out}) | a \in D\}$.
From Equation 3.4 and Equation 3.1, we have:

$$\{\ell \cup \ell^D | \ell \in \mathscr{L}(\gamma')\} = \{\ell \downarrow_\omega | \ell \in \mathscr{L}_\sigma(\chi)\} \tag{3.5}$$

By definition of an induced labelling set (Definition 38), we have:

$$\mathscr{L}_\sigma^{\mu(\kappa)} = \{\ell \cup \ell^D | \ell \in \mathscr{L}(\gamma')\} \tag{3.6}$$

By definition of a canonical local function (Definition 47), we have:

$$\mathscr{F}_\sigma(\gamma, \omega^{inp}, \mu, \omega^R) = \{\ell \downarrow_\omega | \ell \in \mathscr{L}_\sigma(\chi)\} \tag{3.7}$$

From Equations 3.6 and 3.7 and 3.5, we prove thus that:

$$\mathscr{L}_\sigma^{\mu(\kappa)} = \mathscr{F}_\sigma(\gamma, \omega^{inp}, \mu, \omega^R)$$

■

3.7.3 Proofs of soundness and completeness

In all the following proofs, by $\mathscr{L}_\sigma()$ we mean "the set of labellings under the semantics σ according to the mathematical definition of σ" whereas by $\mathscr{L}^*_\sigma()$ we mean "the set of labellings under the semantics σ computed with our algorithm".

Thus, proving completeness is proving that $\mathscr{L}_\sigma() \subseteq \mathscr{L}^*_\sigma()$ and proving soundness is proving $\mathscr{L}^*_\sigma() \subseteq \mathscr{L}_\sigma()$.

We assume, in the following proofs, that the external existing solver used to compute the labellings of the induced AFs from the different cluster structures is sound and complete for the grounded, complete, stable and preferred semantics.

Proposition 7 *(Soundness of Algorithm 2). Algorithm 2 is sound for the* stable *and* complete *semantics.*

PROOF. *(Soundness of Algorithm 2). Let $\gamma = \langle A, R\rangle$ be an AF and $\Omega = \{\omega_1, ..., \omega_n\}$ be a partition of A corresponding to the clustering of γ. Let σ be a fully decomposable and complete-compatible semantics and let ℓ^* be a labelling of γ according to σ obtained by Algorithm 2.*

Let suppose that $\ell^* \notin \mathscr{L}_\sigma(\gamma)$. We will prove that it is impossible with a reductio ad absurdum.

As σ is a complete-compatible and fully decomposable semantics we can say that (Definition 48):

$$\ell^* \notin \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_\sigma(\gamma{\downarrow_{\omega_i}}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\}\ s.t.\ j\neq i} \ell^{\omega_j}){\downarrow_{\omega_i^{inp}}}, \omega_i^R)\} \tag{3.8}$$

And so:

$$\exists \omega_i \in \Omega\ s.t.\ \ell^*{\downarrow_{\omega_i}} \notin \mathscr{F}_\sigma(\gamma{\downarrow_{\omega_i}}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\}\ s.t.\ j\neq i} \ell^{\omega_j}){\downarrow_{\omega_i^{inp}}}, \omega_i^R) \tag{3.9}$$

In the following we denote by ω the particular ω_i for which Formula 3.9 holds in order to simplify the notation.

Let $\kappa = \langle \gamma{\downarrow_\omega}, I = \omega^R, O = R \cap (\omega \times A \setminus \omega), B = \{a | (a,b) \in O \text{ or } (b,a) \in I\}\rangle$ be the cluster structure corresponding to ω.

Let μ be a context of κ such that $\mu = (\bigcup_{j\in\{1,...,n\}\ s.t.\ \omega_j\neq\omega} \ell^{\omega_j}){\downarrow_{\omega^{inp}}}$.

Let $\mathscr{L}_\sigma^{*\mu(\kappa)}$ be the set of labellings of κ under the context μ produced by Algorithm 2.

Let $\ell'^* \in \mathscr{L}_\sigma^{*\mu(\kappa)}$ be the labelling coinciding with $\ell^*{\downarrow_\omega}$ (i.e. $\ell'^* = \ell^*{\downarrow_\omega}$).

We have so:

$$\ell'^* \in \mathscr{L}_\sigma^{*\mu(\kappa)} \tag{3.10}$$

Whereas:

$$\ell'^* \notin \mathscr{F}_\sigma(\gamma{\downarrow_\omega}, \omega^{inp}, \mu, \omega^R) \tag{3.11}$$

And so:

$$\mathscr{L}_\sigma^{*\mu(\kappa)} \neq \mathscr{F}_\sigma(\gamma{\downarrow_\omega}, \omega^{inp}, \mu, \omega^R) \tag{3.12}$$

Nevertheless, according to Proposition 6 we must have:

$$\mathscr{L}_\sigma^{*\mu(\kappa)} = \mathscr{F}_\sigma(\gamma{\downarrow_\omega}, \omega^{inp}, \mu, \omega^R) \tag{3.13}$$

Thus, there is a contradiction between Equation 3.12 and Equation 3.13.

From this contradiction we can conclude that:

$$\mathscr{L}_\sigma^*(\gamma) \subseteq \mathscr{L}_\sigma(\gamma) \tag{3.14}$$

We prove so that for any fully decomposable semantics σ our algorithm is sound, and so for the complete and stable semantics following Proposition 3. ■

Proposition 8 *(Completeness of Algorithm 2). Algorithm 2 is complete for the* stable, complete *and* preferred *semantics.*[6]

[6]Indeed, even if Algorithm 2 is only defined for stable and complete semantics, it can be proven that its completeness property also holds when it is used with a semantic σ that is the preferred one.

PROOF. *(Completeness of Algorithm 2). Let $\gamma = \langle A, R\rangle$ be an AF, $\Omega = \{\omega_1, ..., \omega_n\}$ be a partition of A and $\{\kappa_1, ..., \kappa_n\}$ be the set of cluster structures corresponding to Ω, with each κ_i being defined as:*

$$\kappa_i = \langle \gamma{\downarrow_{\omega_i}}, I = \omega_i^R, O = R \cap (\omega_i \times A \setminus \omega_i), B = \{a | (a,b) \in O \text{ or } (b,a) \in I\}\rangle$$

Let $\mathscr{L}_{\mathscr{D}}^{\kappa_i}$ be the set of distinct labellings of κ_i according to the semantics σ.
Let $\mathscr{L}_{\sigma}^{*}(\gamma)$ be the set of labellings of γ according to σ obtained by Algorithm 2.
Let $\mathscr{L}_{\sigma}^{*\mu(\kappa_i)}$ be the set of labellings of κ_i under the context μ.
Let σ be a top-down decomposable semantics.
By definition we have (Definition 51):

$$\mathscr{L}_{\sigma}(\gamma) \subseteq \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_{\sigma}(\gamma{\downarrow_{\omega_i}}, \omega_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^{\omega_j}){\downarrow_{\omega_i^{inp}}}, \omega_i^R)\} \tag{3.15}$$

Given that the labellings of all cluster structures are computed for every possible context, we have, by definition of the context and of the input arguments:

$$\forall i, \forall \ell^{inp} = (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^{\omega_j}){\downarrow_{\omega_i^{inp}}}, \exists \mu^{\kappa_i} \text{ s.t. } \mu^{\kappa_i} = \ell^{inp} \tag{3.16}$$

Given that the external solver that computes the labellings of $\gamma{\downarrow_{\omega_i}}$ according to the semantics σ is sound and complete, and considering χ being the standard AF w.r.t to the AF with input $\langle \gamma{\downarrow_{\omega_i}}, \omega_i^{inp}, \mu^{\kappa_i}, \omega_i^R\rangle$, we have:

$$\forall i, \forall \mu^{\kappa_i}, \forall \ell^{\chi} \in \mathscr{L}_{\sigma}(\chi), \exists \ell \in \mathscr{L}_{\sigma}^{*\mu(\kappa_i)} \text{ s.t. } \ell = \ell^{\chi}{\downarrow_{\omega_i}} \tag{3.17}$$

So we have:

$$\forall i, \forall \mu^{\kappa_i}, \forall \ell^{\chi} \in \mathscr{L}_{\sigma}(\chi), \ell^{\chi}{\downarrow_{\omega_i}} \in \mathscr{L}_{\mathscr{D}}^{\kappa_i} \tag{3.18}$$

And so (following Def. 47):

$$\forall \omega_i, \mathscr{F}_{\sigma}(\gamma{\downarrow_{\omega_i}}, \omega_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^{\omega_j}){\downarrow_{\omega_i^{inp}}}, \omega_i^R) \subseteq \mathscr{L}_{\mathscr{D}}^{\kappa_i} \tag{3.19}$$

As a consequence and because of Equation 3.15 we have ($\prod$ denoting the cartesian product):

$$\mathscr{L}_{\sigma}(\gamma) \subseteq \prod_{\kappa_i} \mathscr{L}_{\mathscr{D}}^{\kappa_i} \tag{3.20}$$

Let $\psi = \{\ell | \ell \in \prod_{\kappa_i} \mathscr{L}_{\mathscr{D}}^{\kappa_i}$ and $\exists a \in A$ s.t. a is illegally labelled in $\ell\}$ be the set of all possible incorrect labellings (i.e. the set of labellings in which there exists an argument that is not legally labelled).
We have, by definition of σ:

$$\mathscr{L}_{\sigma}(\gamma) \subseteq (\prod_{\kappa_i} \mathscr{L}_{\mathscr{D}}^{\kappa_i}) \setminus \psi \tag{3.21}$$

Given that, for all computed labellings, we keep only the merged configuration, that is the most flexible possible configuration, our CSP modelisation does not add extra constraints.
The proposed reunification removes, thus, only the labellings belonging to ψ.
As a consequence, we have:

$$\mathscr{L}_\sigma(\gamma) \subseteq \mathscr{L}^*_\sigma(\gamma) \tag{3.22}$$

We prove so that for any top-down decomposable semantics σ our algorithm is complete, and so for the complete, stable and preferred semantics following Proposition 4. ■

Proposition 9 *(Soundness and completeness of Algorithm 3). Algorithm 3 is sound and complete for the* preferred *semantics.*

PROOF. *(Soundness and completeness of Algorithm 3). Let $\gamma = \langle A, R\rangle$ be an AF. Given that steps 1-4 of Algorithm 3 exactly correspond to Algorithm 2 and given that Algorithm 2 is* complete *for the* preferred *semantics (see Proposition 8), $\mathscr{L}$, the set of all labellings reunified from the different clusters obtained in the step 4 of Algorithm 3, contains all the preferred labellings of γ.*

Moreover, the difference between Algorithm 2 and Algorithm 3 is that, in the step 5 of Algorithm 3, we keep from $\mathscr{L}$ only the maximal (w.r.t $\subseteq$ of $\mathtt{in}$-labelled arguments) labellings, that are by definition the preferred labellings. As a consequence, $\mathscr{L}_{pr}$ contains only and all the preferred *labellings of γ.*

Algorithm 3 is, thus, sound and complete for the preferred *semantics.* ■

Let us prove now that the entire algorithm is sound and complete for the *stable* and *complete* semantics when using Algorithm 1 and Algorithm 2, and sound and complete for the *preferred* semantics when using Algorithm 1 and Algorithm 3.

Proposition 10 *(Soundness of Algorithm 1 + Algorithm 2). Algorithm 1 is sound for the* stable *and* complete *semantics when using Algorithm 2 to compute the component labellings.*

PROOF. *(Soundness of Algorithm 1 + Algorithm 2). Let $\Gamma = \langle A, R\rangle$ be an AF, ℓ_{gr} be its grounded labelling, $\Gamma_{hard} = \Gamma\downarrow_{\{a|a\in A, \ell_{gr}(a)=\mathtt{und}\}}$ be the hard part of Γ and $\{\gamma_1 = \langle A_1, R_1\rangle, ..., \gamma_n = \langle A_n, R_n\rangle\}$ be the set of AFs obtained from Γ_{hard} components.*

Let σ be the complete *or* stable *semantics.*

Let $\mathscr{L}^*_\sigma(\Gamma)$ be the set of labellings of Γ obtained from Algorithm 1.

Let $\mathscr{L}_\sigma(\Gamma)$ be the set of labellings of Γ.

Let $\ell^* \in \mathscr{L}^*_\sigma(\Gamma)$ be a labelling of Γ computed by Algorithm 1.

Let $\mathscr{L}^*_\sigma(\gamma_i)$ be the set of labellings of γ_i obtained from Algorithm 2.

Following Algorithm 1, we have:

$$\ell^* = \ell_{gr} \cup \bigcup \ell^*_i, \text{ with } \ell^*_i \in \mathscr{L}^*_\sigma(\gamma_i) \tag{3.23}$$

Let $\Omega = \{\omega_{gr}, A_1, ..., A_n\}$ be a partition of A with $\omega_{gr} = \{a | a \in \mathtt{in}(\ell_{gr}) \text{ or } a \in \mathtt{out}(\ell_{gr})\}$.

We have (following Def. 47):

$$\mathscr{F}_\sigma(\gamma\downarrow_{\omega_{gr}}, \omega^{inp}_{gr}, (\bigcup_{i\in\{1,...,n\}} \ell^{A_i})\downarrow_{\omega^{inp}_{gr}}, \omega^R_{gr}) = \{\ell_{gr}\} \tag{3.24}$$

Because σ is a fully decomposable semantics we have so (Definition 48):

$$\mathscr{L}_\sigma(\Gamma) = \{\ell_{gr} \cup \bigcup_{A_i} \ell^{A_i}\} \text{ with } \ell^{A_i} \in \mathscr{F}_\sigma(\gamma\downarrow_{A_i}, A_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{A_j})\downarrow_{A_i^{inp}}, A_i^R) \tag{3.25}$$

Given that Equation 3.25 holds and that Algorithm 2 is sound for fully decomposable semantics (i.e. $\forall p \in \Omega$, $\mathscr{L}^*_\sigma(\Gamma\downarrow_p) \subseteq \mathscr{L}_\sigma(\Gamma\downarrow_p)$*), we have:*

$$\ell^* \in \mathscr{L}_\sigma(\Gamma) \tag{3.26}$$

And thus:

$$\mathscr{L}^*_\sigma(\Gamma) \subseteq \mathscr{L}_\sigma(\Gamma) \tag{3.27}$$

We prove so that for the complete *and* stable *semantics our algorithm is sound.*

■

Proposition 11 *(Completeness of Algorithm 1 + Algorithm 2). Algorithm 1 is complete for the* stable*,* complete *and* preferred *semantics when using Algorithm 2 to compute the component labellings.*[7]

PROOF. *(Completeness of Algorithm 1 + Algorithm 2). Let* $\Gamma = \langle A, R\rangle$ *be an AF,* ℓ_{gr} *be its grounded labelling,* $\Gamma_{hard} = \Gamma\downarrow_{\{a|a\in A, \ell_{gr}(a)=\texttt{und}\}}$ *be the hard part of* Γ *and* $\{\gamma_1 = \langle A_1, R_1\rangle, ..., \gamma_n = \langle A_n, R_n\rangle\}$ *be the set of AFs obtained from* Γ_{hard} *components.*

Let σ *be the* complete*,* stable *or* preferred *semantics.*

Let $\mathscr{L}^*_\sigma(\Gamma)$ *be the set of labellings obtained from Algorithm 1.*

Let $\mathscr{L}^*_\sigma(\gamma_i)$ *be the set of labellings obtained from Algorithm 2 for the component* γ_i*.*

Let $\mathscr{L}_\sigma(\Gamma)$ *be the set of labellings of* Γ*.*

Let $\Omega = \{\omega_{gr}, A_1, ..., A_n\}$ *be a partition of A with* $\omega_{gr} = \{a | a \in \texttt{in}(\ell_{gr}) \text{ or } a \in \texttt{out}(\ell_{gr})\}$*.*

Let $\ell \in \mathscr{L}_\sigma(\Gamma)$ *be a labelling of* Γ *according to* σ*.*

Given that (following Def. 47):

$$\mathscr{F}_\sigma(\gamma\downarrow_{\omega_{gr}}, \omega_{gr}^{inp}, (\bigcup_{i\in\{1,...,n\}} \ell^{A_i})\downarrow_{\omega_{gr}^{inp}}, \omega_{gr}^R) = \{\ell_{gr}\} \tag{3.28}$$

We have by definition of top-down decomposable semantics (following Def. 51):

$$\mathscr{L}_\sigma(\Gamma) \subseteq \{\ell_{gr} \cup \bigcup_{A_i} \ell^{A_i}\} \text{ with } \ell^{A_i} \in \mathscr{F}_\sigma(\gamma\downarrow_{A_i}, A_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{A_j})\downarrow_{A_i^{inp}}, A_i^R) \tag{3.29}$$

Given that Algorithm 2 is complete for top-down decomposable semantics (i.e. $\forall p \in \Omega, \mathscr{L}_\sigma(\Gamma\downarrow_p) \subseteq \mathscr{L}^*_\sigma(\Gamma\downarrow_p)$*),*

$$\forall A_i, \ell^{A_i} \in \mathscr{L}^*_\sigma(\gamma_i) \tag{3.30}$$

Furthermore:

$$\forall \ell^* \in \mathscr{L}^*_\sigma(\Gamma), \ell^* = \ell_{gr} \cup \bigcup \ell_i^*, \text{ with } \ell_i^* \in \mathscr{L}^*_\sigma(\gamma_i) \tag{3.31}$$

[7] Same remark as the one given for Prop. 8 about the completeness in the case of preferred semantics.

We have so:

$$\{\ell_{gr} \cup \bigcup_{A_i} \ell^{A_i}\} = \mathscr{L}^*_\sigma(\Gamma) \tag{3.32}$$

Finally, we have:

$$\mathscr{L}_\sigma(\Gamma) \subseteq \mathscr{L}^*_\sigma(\Gamma) \tag{3.33}$$

We prove so that our algorithm is complete for the complete, stable *and* preferred *semantics.* ■

Proposition 12 *(Soundness and completeness of Algorithm 1 + Algorithm 3).*
Algorithm 1 is sound and complete for the preferred *semantics when using Algorithm 3 to compute the component labellings.*

PROOF. *(Soundness and completeness of Algorithm 1 + Algorithm 3). Let $\Gamma = \langle A, R\rangle$ be an AF, ℓ_{gr} be its grounded labelling, $\Gamma_{hard} = \Gamma\downarrow_{\{a|a\in A, \ell_{gr}(a)=\texttt{und}\}}$ be the hard part of Γ and $\{\gamma_1 = \langle A_1, R_1\rangle, ..., \gamma_n = \langle A_n, R_n\rangle\}$ be the set of AFs obtained from Γ_{hard} components.*

Let $\mathscr{L}^*_{pr}(\Gamma)$ be the set of labellings of Γ obtained from Algorithm 1.

Let $\mathscr{L}_{pr}(\Gamma)$ be the set of labellings of Γ.

Let $\mathscr{L}^*_{pr}(\gamma_i)$ be the set of labellings of γ_i obtained from Algorithm 3.

Following Algorithm 1, we have:

$$\mathscr{L}^*_{pr}(\Gamma) = \{\ell_{gr} \cup \ell^{A_1} \cup ... \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{L}^*_{pr}(\gamma_i)\} \tag{3.34}$$

Let $A_0 = \{a | a \in \texttt{in}(\ell_{gr})$ or $a \in \texttt{out}(\ell_{gr})\}$ be the fixed part of Γ. The set of argument set $\Omega = \{A_0, A_1, ..., A_n\}$ is then a partition of A.

By definition of the grounded labelling, we have:

$$\exists a \in A \text{ s.t. } \ell_{gr}(a) = \texttt{und} \implies (\forall a' \in A \text{ s.t. } (a', a) \in R, \ell_{gr}(a) \neq \texttt{in}) \tag{3.35}$$

Given that:

$$\texttt{und}(\ell_{gr}) \cap A_0 = \emptyset \tag{3.36}$$

And that by construction of A_0:

$$\forall i \in \{1, ..., n\}, \forall a \in A_i, \ell_{gr}(a) = \texttt{und} \tag{3.37}$$

The consequence of Equation 3.35 is:

$$\forall i \in \{1, ..., n\}, \forall (a', a) \in R \text{ s.t. } a' \in A_0 \text{ and } a \in A_i, \ell_{gr}(a') = \texttt{out} \tag{3.38}$$

Let $\Gamma' = \langle A, R \setminus \{(a', a) | a' \in A_0 \text{ and } a \notin A_0\}\rangle$ be the AF constructed by removing from Γ the attacks between its fixed part and its non fixed part. As all arguments in the fixed part attacking arguments outside the fixed part is labelled `out` *(Equation 3.38) their attacks have no effect. The consequence is the following:*

$$\mathscr{L}_{pr}(\Gamma') = \mathscr{L}_{pr}(\Gamma) \tag{3.39}$$

Notice that Γ' has $n+1$ connected components corresponding to the partition Ω. Given that there is no connection (attack) between those connected components we can say that:

$$\Omega \in \mathscr{S}_{USCC}(\Gamma') \tag{3.40}$$

As the preferred semantics is fully decomposable w.r.t. $\mathscr{S}_{USCC}$ (Proposition 5), we have (following Definition 53):

$$\mathscr{L}_{\sigma}(\Gamma') = \{\ell^{A_0} \cup ... \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{F}_{pr}(\Gamma \downarrow_{A_i}, A_i^{inp}, (\bigcup_{j \in \{0,...,n\} \text{ s.t. } j \neq i} \ell^{A_j}) \downarrow_{A_i^{inp}}, A_i^R)\} \tag{3.41}$$

Notice that:

$$\mathscr{F}_{pr}(\Gamma' \downarrow_{A_0}, A_0^{inp}, (\bigcup_{i \in \{1,...,n\}} \ell^{A_i}) \downarrow_{A_0^{inp}}, A_0^R) = \{\ell_{gr}\} \tag{3.42}$$

Notice also that, given Algorithm 3 is sound and complete for the preferred *semantics (Proposition 12), we have:*

$$\forall i \in \{1,...,n\}, \mathscr{F}_{pr}(\Gamma' \downarrow_{A_i}, A_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^{A_j}) \downarrow_{A_i^{inp}}, A_i^R) = \mathscr{L}^*_{pr}(\gamma_i) \tag{3.43}$$

From the equations 3.41, 3.42, 3.43, we have:

$$\mathscr{L}_{pr}(\Gamma') = \{\ell_{gr} \cup \ell^{A_1} \cup ... \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{L}^*_{pr}(\gamma_i)\} \tag{3.44}$$

From Equation 3.39 and Equation 3.44, we have:

$$\mathscr{L}_{pr}(\Gamma) = \{\ell_{gr} \cup \ell^{A_1} \cup ... \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{L}^*_{pr}(\gamma_i)\} \tag{3.45}$$

Finally, from Equation 3.45 and Equation 3.34 we have:

$$\mathscr{L}^*_{pr}(\Gamma) = \mathscr{L}_{pr}(\Gamma) \tag{3.46}$$

We prove so that Algorithm 1, when using Algorithm 3 to compute the component labellings, is sound and complete for the preferred *semantics.*

■

Chapter 4

Experimental results

In this chapter we will present experiment results conducted with *AFDivider*, our algorithm.

4.1 The experimental setting

Experiments presented in this paper were carried out using the OSIRIM platform that is managed by IRIT and supported by CNRS, the Region Midi-Pyrénées, the French Government, and ERDF (see `http://osirim.irit.fr/site/en` *).*

There exists an argumentation solver competition in abstract argumentation, the International Competition on Computational Models of Argumentation [1] (ICCMA), in which solvers are compared on several types of argumentation problems. One of these problems is to enumerate all the labellings of an AF given a semantic. As some instances were too hard to be solved in reasonable time they have been removed from the competition. Since we are interested in solving "large-scale" AF, we have tried our algorithm on some of these instances.

To compute the labellings of an induced AF after the clustering process, we have used an already existing solver called "Pyglaf", one of the best solver at the last ICCMA session, which transforms the AF labelling problem into a SAT problem (for more details see [2]). Given that our results were dependent of Pyglaf performances, we have only compared our algorithm (using Pyglaf) with Pyglaf itself on some hard instances and for the preferred semantic.

For each experiment we used 6 cores of a AMD Opteron 6262HE processor, each core having a frequency of 1.6 GHz. The RAM size was 45GB. The timeout had been set to 1 hour.

As the solvers are multithreaded we have chosen to compare them using real time.

4.2 The results

Figure 4.1 compares our algorithm (using Pyglaf) with Pyglaf itself. In that table, by *MemOv* we mean "memory overflow" and by *Timeout* we mean that the experiment did not end in one hour. The time result format is "minutes:secondes.centiseconds".

[1] `http://argumentationcompetition.org/`

		Pyglaf		AFDivider	
	nb labellings	end state	time	end state	time
BA_160_50_1.apx	24 576	✓	0:03.37	✓	0:2.60
BA_160_20_4.apx	24 576	✓	0:10.93	✓	0:2.48
bw2.pfile-3-05.pddl.3.cnf.apx	29 928	✓	1:17.25	✓	6:12.23
bw2.pfile-3-05.pddl.1.cnf.apx	39 976	✓	0:41.53	✓	0:46.99
BA_120_70_1.apx	288 000	✓	2:18.83	✓	0:15.70
BA_100_60_2.apx	1 078 272	✓	20:59.39	✓	0:37.35
BA_120_80_2.apx	1 285 632	*Timeout*		✓	0:58.68
BA_180_60_4.apx	1 376 256	✓	30:04.05	✓	1:34.47
basin-or-us.gml.20.apx	1 963 008	*Timeout*		✓	1:15.89
BA_100_80_3.apx	4 478 976	*Timeout*		✓	2:42.48
amador-transit_20151216_1706.gml.80.apx	11 751 480	*Timeout*		✓	36:45.63
BA_180_70_1.apx	323 592 192	*Timeout*		*MemOv*	
BA_120_90_5.apx	394 243 200	*Timeout*		*MemOv*	
BA_200_70_4.apx	10 749 542 400	*Timeout*		*MemOv*	

Figure 4.1: Experiment comparison table: hard instances

We can see in Figure 4.1 that for "small" instances according to the number of labellings (less than 50 000 labellings) our algorithm is not always better. But for harder instances, our algorithm is far better than Pyglaf. Except the *amador-transit_20151216_1706.gml.80.apx* instance (because we cannot really compare as our algorithm time is too closed to the timemout), we can observe a real order of magnitude change: from more than one hour to less than three minutes.

These results confirm that the proposed algorithm is of interest, and that the AF clustering approach is relevant. These results will be deeper analysed, and further experiments will be conducted in future work. The effect of the graph types on the performances should in particular be investigated.

Chapter 5

Related works

In this chapter we will compare the behaviour of our algorithm to other existing ones. In order to illustrate how these other algorithms work, we will consider the AF shown in Figure 5.1 as running example and show how the preferred labellings are computed following the different algorithms. This particular AF has been chosen because its structure let appear clusters in it, it has 4 SCCs and there is an interesting hierarchy between them. Theses two last points are very relevant for the algorithms presented in Sections 5.1 and 5.2.

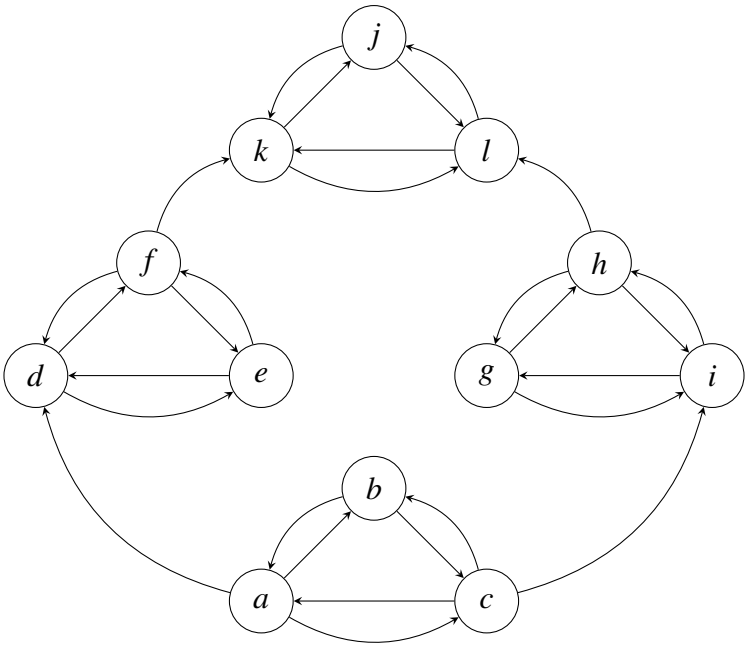


Figure 5.1: AF example Γ

5.1 Dynamic programming algorithm

In [15], Dvořák et al. proposed an algorithm based on a dynamic analysis of an argumentation framework. In the interest of brevity, we will just highlight the main idea of this algorithm (see [15] for a more detailed explanation).

Basically, this algorithm relies on the nice tree decomposition of a graph.

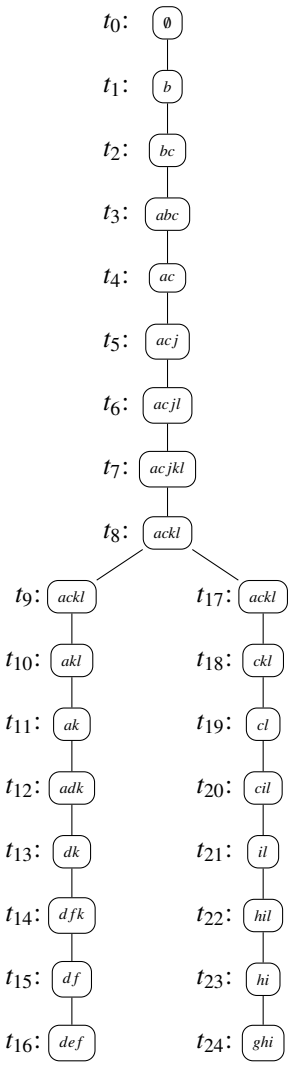


Figure 5.2: Nice tree decomposition of Γ

Definition 57 *(Tree decomposition). Let $G = (V,E)$ be a non directed graph. A tree decomposition of G is a pair $\langle \mathscr{T}, \mathscr{X} \rangle$ where $\mathscr{T} = (V_{\mathscr{T}}, E_{\mathscr{T}})$ is a tree and $\mathscr{X} = (X_t)_{t \in V_{\mathscr{T}}}$ is a set of so-called bags, which has to satisfy the following conditions:*

- $\bigcup_{t \in V_{\mathscr{T}}} X_t = V$, *i.e.* $\mathscr{X}$ *is a set covering of* V.
- *for each* $v \in V$, $\mathscr{T} \downarrow_{\{t | v \in X_t\}}$ *is a connected tree.*
- *for each* $\{v_i, v_j\} \in E$, $\{v_i, v_j\} \subseteq X_t$ *for some* $t \in V_{\mathscr{T}}$.

Definition 58 *(Width of a tree decomposition). Let $\langle \mathscr{T}, \mathscr{X} \rangle$ be a tree decomposition where $\mathscr{T} = (V_{\mathscr{T}}, E_{\mathscr{T}})$ is a tree and $\mathscr{X} = (X_t)_{t \in V_{\mathscr{T}}}$ is a set of so-called bags. The width of such a tree decomposition is given by:*

$$max\{card(X_t) | t \in V_{\mathscr{T}}\} - 1$$

Definition 59 *(Tree width of a graph). Let $G = (V,E)$ be a non directed graph. The tree-width of G is defined by the minimum width over all its tree decompositions.*

Definition 60 *(Nice tree decomposition). A tree decomposition $\langle \mathscr{T}, \mathscr{X} \rangle$ of a graph G is called nice if $\mathscr{T}$ is a rooted tree and if each node $t \in \mathscr{T}$ is one of the following types:*

- *LEAF: t is a leaf of $\mathscr{T}$*
- *FORGET: t has only one child t' and $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$*
- *INSERT: t has only one child t' and $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$*
- *JOIN: t has two children t', t'' and $X_t = X_{t'} = X_{t''}$*

Example 17 *Figure 5.2 shows one nice tree decomposition of the AF Γ, $\langle \mathscr{T}, \mathscr{X} \rangle$ where:*

- *$\mathscr{T} = (V_{\mathscr{T}}, E_{\mathscr{T}})$ with:*
  - *$V_{\mathscr{T}} = \{t_i | i \in [\![0,24]\!]\}$*
  - *$E_{\mathscr{T}} = \{(t_i, t_{i+1}) | i \in [\![0,15]\!] \cup [\![17,23]\!]\} \cup \{(t_8, t_{17})\}$*
- *$\mathscr{X} = \{X_{t_i} | i \in [\![0,24]\!]\}$ with:*

$v \backslash X_t$	X_{t_0}	X_{t_1}	X_{t_2}	X_{t_3}	X_{t_4}	X_{t_5}	X_{t_6}	X_{t_7}	X_{t_8}	X_{t_9}	$X_{t_{10}}$	$X_{t_{11}}$	$X_{t_{12}}$	$X_{t_{13}}$	$X_{t_{14}}$	$X_{t_{15}}$	$X_{t_{16}}$	$X_{t_{17}}$	$X_{t_{18}}$	$X_{t_{19}}$	$X_{t_{20}}$	$X_{t_{21}}$	$X_{t_{22}}$	$X_{t_{23}}$	$X_{t_{24}}$
a				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					✓							
b		✓	✓	✓																					
c			✓	✓	✓	✓	✓	✓	✓	✓								✓	✓	✓	✓				
d													✓	✓	✓	✓	✓								
e																	✓								
f															✓	✓	✓								
g																									✓
h																							✓	✓	✓
i																					✓	✓	✓	✓	✓
j						✓	✓	✓																	
k								✓	✓	✓	✓	✓	✓	✓	✓			✓	✓						
l							✓	✓	✓	✓	✓							✓	✓	✓	✓	✓	✓		

Table 5.1: Bags of $\mathscr{X}$
(✓ means that the vertex v corresponding to the current line belongs to the bag X_t corresponding to the current column)

As node type examples, according to Definition 60:

- *t_{24} is a LEAF type node*
- *t_2 is a FORGET type node*
- *t_{10} is a INSERT type node*
- *t_8 is a JOIN type node*

The nice tree decomposition shown in Figure 5.2 is one among all tree decompositions of Γ with the minimal width, which is 4. In other words, the tree-width of Γ is 4.

There exist other possible nice tree decompositions of Γ with non minimal width. As an example, the one shown in Figure 5.3 has a width of 11.

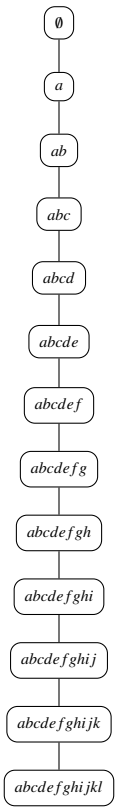


Figure 5.3: Nice tree decomposition of Γ

To each nice tree node is associated a sub AF defined as following:

Definition 61 *(Tree node associated AF). Let Γ be an AF and $\langle \mathscr{T}, \mathscr{X} \rangle$ be its tree decomposition where $\mathscr{T} = (V_{\mathscr{T}}, E_{\mathscr{T}})$ is a tree and $\mathscr{X} = (X_t)_{t \in V_{\mathscr{T}}}$ is a set of so-called bags. We denote by $X_{\geqslant t}$ the union of all bags $X_s \in \mathscr{X}$ such that s occurs in the subtree of $\mathscr{T}$ rooted at t.*

Let $t \in V_{\mathscr{T}}$ be a tree node. The AF γ associated with t is defined as following:

$$\gamma = \Gamma \downarrow_{X_{\geqslant t}}$$

Example 18 *Let take as example the node t_{12} in Figure 5.2. According to Definition 61, we have:*

$$X_{\geqslant t_{12}} = X_{t_{12}} \cup X_{t_{13}} \cup X_{t_{14}} \cup X_{t_{15}} \cup X_{t_{16}} = \{a, d, e, f, k\}$$

We have so:

$$\gamma = \Gamma \downarrow_{X_{\geqslant t_{12}}} = \Gamma \downarrow_{\{a,d,e,f,k\}}$$

Figure 5.4 shows the AF γ associated with the node t_{12}.

Once the AF nice tree determined and the sub AFs associated to each tree node, the tree is explored from the bottom up. On each tree node, the labellings of its associated AF are computed. The node type (LEAF, INSERT, FORGET or JOIN) indicates which operations to do in order to update the computed set of labellings.

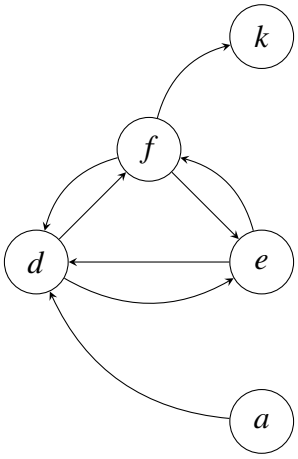


Figure 5.4: γ, AF associated with the node t_{12}

Notice that the sub AF associated with the tree root is the whole AF. So, at the tree root, all the labellings of the AF are found.

This is basically the general idea of this algorithm.

This algorithm is dynamic in the sense that we are interested in the labellings of sub AF that evolve dynamically following the nice tree decomposition. To each leaf is associated an initial AF that will be transformed forgetting and inserting argument nodes in it. This approach has the advantage of breaking the SCC and eventually the hardness of the AF problem. Nevertheless it has also some disadvantages.

Indeed, each step adds or removes at most one argument. The consequence is that a lot of updates are useless and a lot of space is used for potential correct labellings.

Example 19 *Let take as example the AF in Figure 5.5 and its nice tree decomposition in Figure 5.6.*

Figure 5.5: AF example

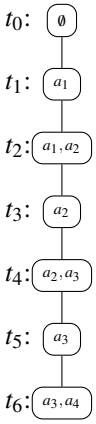


Figure 5.6: Nice tree decomposition

Although this AF admits only one preferred labelling which is $\{(a_1, \texttt{in}), (a_2, \texttt{out}), (a_3, \texttt{in}), (a_4, \texttt{out})\}$,

as we go from the leaf to the top the set of partial labellings will be updated 6 times and, at each tree node, we will have to consider all the potential partial labellings.

Actually, this algorithm does not work directly with labellings but with "colorings" which is a 4-state argument mapping from which are determined the semantic extensions we are interested in. Without going too deep into the details of how this coloring works, we will just highlight the fact that for each argument attacked by an argument outside the current associated AF, four colorings have to be consider, according to the four possible status of that argument. As a consequence, a lot of space is used in order to ensure that all possibilities have been explored.

This algorithm and the *AFDivider* algorithm have both the ability to break the SCC and hopefully the hardness the AF. However, they differ on other points and the main one is how the combinatorial effect of potential labelling number is tackled. Although the *AFDivider* algorithm computes all cases for a given cluster, this combinatorial effect is limited to that particular cluster and is not propagated on the whole AF. As a consequence, space and computational time are spared.

5.2 SCC Decomposition based algorithm

In [17], Beishui Liao proposed an algorithm that computes the labellings of an AF following its SCC decomposition.

Notice that if each SCC component of a graph is considered as a super node, the resulting super graph will be acyclic.

We can thus have a hierarchical representation of this super graph: in the first layer are SCC components with no parents, in the second layer are contained all the SCC components whose parents are in the previous layers, and so on.

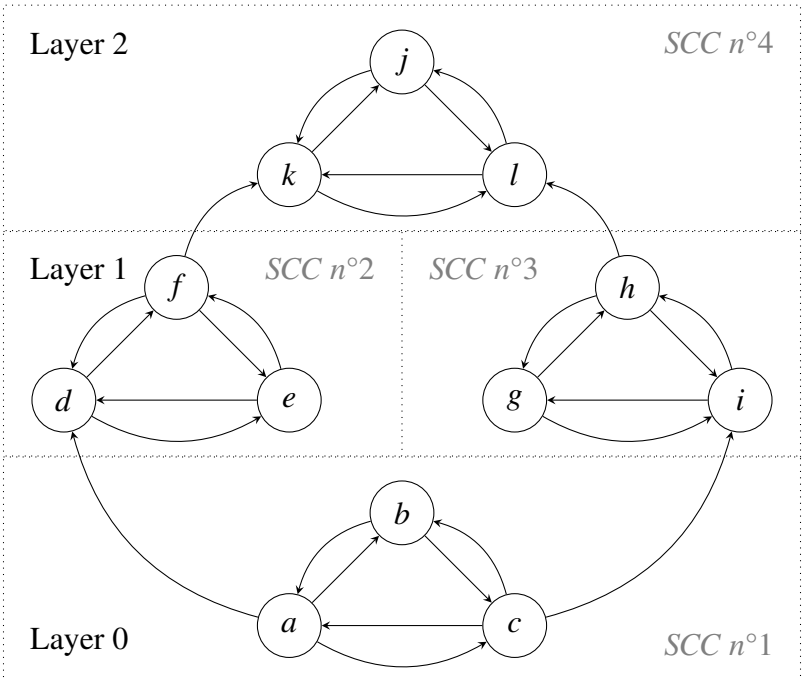


Figure 5.7: AF example Γ (the first layer being Layer 0)

Example 20 *Figure 5.7 is the SCC hierarchical view of* Γ.

Given that the labellings of each SCC component are influenced only by the ones of its parents, it is possible to guide the research of labellings following the hierarchical representation of the SCC components of the AF. This is the main idea of the algorithm.

Example 21 *In a first step, the labellings of the SCC n°1 are computed. The result is the following set of labellings:*

$$\{\ell_1^{scc1}, \ell_2^{scc1}, \ell_3^{scc1}\} \text{ with } \begin{cases} \ell_1^{scc1} &= \{(a, \mathtt{in})\}, \{(b, \mathtt{out})\}, \{(c, \mathtt{out})\}, \\ \ell_2^{scc1} &= \{(a, \mathtt{out})\}, \{(b, \mathtt{in})\}, \{(c, \mathtt{out})\}, \\ \ell_3^{scc1} &= \{(a, \mathtt{out})\}, \{(b, \mathtt{out})\}, \{(c, \mathtt{in})\} \end{cases}$$

Then, possible labellings of the SCCs n°2 and n°3 are computed considered the labellings of the parents SCCs, in this case SCC n°1. For instance, considering $\ell_1^{scc1} = \{(a, \mathtt{in})\}, \{(b, \mathtt{out})\}, \{(c, \mathtt{out})\}$*:*

- *For SCC n°2 we have:*

$$\{\ell_1^{scc2}, \ell_2^{scc2}\} \text{ with } \begin{cases} \ell_1^{scc2} &= \{(d, \mathtt{out})\}, \{(e, \mathtt{out})\}, \{(f, \mathtt{in})\}, \\ \ell_2^{scc2} &= \{(d, \mathtt{out})\}, \{(e, \mathtt{in})\}, \{(f, \mathtt{out})\} \end{cases}$$

- *For SCC n°3 we have:*

$$\{\ell_1^{scc3}, \ell_2^{scc3}, \ell_3^{scc3}\} \text{ with } \begin{cases} \ell_1^{scc3} &= \{(g, \mathtt{out})\}, \{(h, \mathtt{out})\}, \{(i, \mathtt{in})\}, \\ \ell_2^{scc3} &= \{(g, \mathtt{out})\}, \{(h, \mathtt{in})\}, \{(i, \mathtt{out})\}, \\ \ell_3^{scc3} &= \{(g, \mathtt{in})\}, \{(h, \mathtt{out})\}, \{(i, \mathtt{out})\} \end{cases}$$

The same thing must be done considering ℓ_2^{scc1} *and* ℓ_3^{scc1}*.*

Afterwards, the labellings of SCC n°4 are computed according to the compatible SCC parents labellings. In the interest of brevity we will not give the entire result as this AF has 47 distinct preferred labellings.

The great advantage of this approach is that no useless computation is made. When going from one layer to another, only possible labellings are considered. This reduces considerably the computational time.

Although not proposed in this paper, it is possible to parallelize the computation when there are independent branches in the acyclic super graph. But even though a distributed version of this algorithm had been proposed, it would still be very different from the *AFDivider* algorithm.

Indeed, this algorithm is profitable only if there are several SCC components and if the hardness of solving the AF problem is not inside the SCC components. The major difference is that the *AFDivider* algorithm is able to look inside SCC components and hopefully break the hardness by finding clusters in it. Another difference is that there are no restriction to parallelize the labelling computation. Finally, the used clustering method tries to balance the cluster sizes (in terms of number of arguments) so that hopefully the workload may be also balanced.

5.3 Parallel SCC-recursive based algorithm: *P-SCC-REC*

The algorithm proposed by Cerruti et al. in [9], named *P-SCC-REC*, has several common points with the *AFDivider* algorithm. Indeed, both algorithms are distributed and they are able to look inside SCC components. Nevertheless, the way of distributing and of "cutting" of the AF are completely different.

The *P-SCC-REC* algorithm is rather complex. We are going to highlight its main concepts (see [9] for additional information).

It is a recursive algorithm. In one recursion level, the following steps are performed:

- As in the *AFDivider* algorithm, the grounded labelling is computed and only the hard part of the AF is considered for the next steps.
- As in the Beishui Liao's algorithm, a SCC hierarchical view of the AF is determined.
- For each SCC, a greedy labelling computing is performed, considering that all arguments attacking the given SCC is labelled $\texttt{out}$. This computation is made in a distributed way, parallelized following the SCCs.
- For each layer:
  - The labelling of the SCCs are computed according to the labelling of their SCC ancestors. This computation is made in a distributed way, parallelized both following the SCCs and the SCC ancestors labellings.
    - In some cases when the SCC ancestors labelling does not allow to determined quickly the labellings of the current SCC, *P-SCC-REC* is called recursively on that particular SCC from which is removed its arguments attacked by its SCC ancestors.
  - Following the previous step, the set of SCC ancestors labellings of the next layer is determined.

Example 22 *Applied to* Γ*,* P-SCC-REC *will behave a bit like Beishui Liao's algorithm as there is no argument labelled* $\texttt{in}$ *nor* $\texttt{out}$ *in the grounded labelling of* Γ*.*

Notice that, given the labelling $\ell_1^{scc1} = \{(a, \texttt{in})\}, \{(b, \texttt{out})\}, \{(c, \texttt{out})\}$*, when computing the labellings of the SCC n°*2*,* P-SCC-REC *will be recursively called on the AF shown in Figure 5.8.*

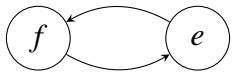


Figure 5.8: *SCC n°*2 under ℓ_1^{scc1}

The *P-SCC-REC* algorithm will look inside an SCC if its SCC ancestor labelling allows it, not according to the size of this SCC and its possible hardness, whereas the *AFDivider* algorithm will try to found clusters similar in size whether it is necessary to break SCCs or not.

There is another aspect of *P-SCC-REC* algorithm that may narrow its performance. If we put aside the greedy phase of the algorithm, the algorithm follows the SCC hierarchical view of the AF and parallelizes following the SCCs in one layer, and following the ancestor labellings. This later parallelization causes two problems:

1. Most of the time, it makes the number of threads explodes and so overloads the CPUs.
2. It leads to redundant computation as the computation cases are not based on the states of input arguments of the current SCC.

Example 23 *Let consider the step to compute the SCC n°*4 *labellings.*

- *As an illustration of point 1:*
  - *We have 21 distinct SCC ancestor labellings and so 21 threads will be created. Although Γ is a small AF, the amount of threads is rather important. On a bigger one, the number of threads could quickly overload the CPUs.*
- *As an example of point 2:*
  - *Even if several distinct SCC ancestor labellings are equal when restricted to the arguments f and h, the labelling computation will be made for each of them, which is highly redundant.*

It is true that some of the cases computed by the *AFDivider* algorithm may be unused in the reunifying phase (bear in mind that it is not possible to know them in advance) but there is no waiting time due to a hierarchical view of the AF, and there is no redundant computation. Furthermore, if the AF is not too dense, the number of threads will not explode, even though the number of labellings is huge.

5.4 To sum up

The advantages of the *AFDivider* algorithm over the compared algorithms are the followings:

- It has the ability to break SCCs whenever it is well suited to have well shaped clusters.
  - Given a current SCC and an ancestor labelling, the *P-SCC-REC* algorithm can break an SCC only when the ancestor labelling have some particular effects on the current SCC (see [9] for more details).
  - Dvořák et al. dynamic algorithm 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 generates a lot of computations and uses a lot of memory.
  - Liao's algorithm does not break SCCs.
- It has the ability to compute the labellings in a distributed way.
  - The *P-SCC-REC* algorithm also uses distributed computation to solve the AF but 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.
  - Dvořák et al. dynamic algorithm is fully sequential.
  - Liao's algorithm is fully sequential.
- It avoids redundant computations.
  - Though more visible in *P-SCC-REC* algorithm as it occurs in different threads, Dvořák et al. dynamic algorithm and Liao's algorithm also make in a certain way redundant computations. Indeed several labellings under construction, restricted to some arguments, may be identical and have the same effect on the non yet labelled part of the AF. Nevertheless these algorithms will not take advantage of that.

    To be fair, the *AFDivider* algorithm may also do such redundant computations but if it is the case it will be restrained to inward part of each cluster and not to the whole AF.

What distinguishes best the *AFDivider* algorithm 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 the *AFDivider* algorithm well suited for non dense AF with a clustered structure. Indeed, in a such structure, the reunifying phase will be less expensive than exploring the whole AF to construct each of the labellings.

Conclusion

The *AFDivider* algorithm presented in this report is the first algorithm that uses spectral clustering methods to compute semantic labellings, and that originally combines them to other existing techniques. 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.

We have proven the soundness and the completeness of this algorithm for the *stable*, *complete* and *preferred* semantics.

Experiments showed that this cutting process and distributed computing allows solving some hard AF instances that could not be solved otherwise, and that it significantly decreases the solving time of others.

We compared the behaviour of our algorithm with other ones that use some kind of clustering. Among the various advantages of our method (its ability to break SCCs, to compute the labellings in a distributed way and to avoid redundant computations) we highlighted the fact that cutting the AF into clusters has the great advantage of limiting the solving hardness to the clusters. Indeed, in those other approaches, the combinatorial effect due to the number of labellings is propagated to the whole AF whereas, in the *AFDivider* algorithm, it is limited to the clusters. This property makes it well suited for non dense AF with a clustered structure.

The idea of clustering an AF and cutting it for parallel computation of labellings is promising, as shown by the experiments. Here are some ideas to go further in this approach:

- A recursive clustering version of this algorithm could be made. Indeed, after the cutting process, an induced AF could still be hard to solve. It may be possible that applying recursively the same clustering process on AF parts (until a certain criterion is satisfied) could enhance the global solving time.

- As shown by the experiments, a work has to be done to tackle the memory limit. A compressed representation of labellings could be very interesting to support scalability.

- We would like to extend this work to more complex AFs, using several types of relation (not only attacks but also supports), with relation and argument strength, recursive relations, and so for more complex semantics.

- We have made the choice to consider non dense AF for which the spectral clustering method is particularly efficient. However, for AF with other graph structures, other clustering methods could be more appropriate. This could be studied in future works.

- An interesting aspect of the *AFDivider* algorithm is that, when several connected components are found after removing the trivial part of an AF, the labellings of these components are all compatible together. This property could be exploited to answer non classical problems such as: "What is the labelling rate in which an argument a is labelled in ?", and that, without explicitly enumerating all the labellings, avoiding the costly cartesian product of component labellings.

Bibliography

[1] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 49–55, 2017.

[2] Mario Alviano. The pyglaf argumentation reasoner. In *OASIcs-OpenAccess Series in Informatics*, volume 58. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[3] Leila Amgoud, Jonathan Ben-Naim, Dragan Doder, and Srdjan Vesic. Acceptability semantics for weighted argumentation frameworks. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, volume 2017, 2017.

[4] Pietro Baroni, Guido Boella, Federico Cerutti, Massimiliano Giacomin, Leendert W. N. van der Torre, and Serena Villata. On input/output argumentation frameworks. In *COMMA*, pages 358–365, 2012.

[5] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[6] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. Afra: Argumentation framework with recursive attacks. *International Journal of Approximate Reasoning*, 52(1):19–37, 2011.

[7] Martin Caminada. On the issue of reinstatement in argumentation. In *JELIA*, pages 111–123, 2006.

[8] Federico Cerutti, Massimiliano Giacomin, Mauro Vallati, and Marina Zanella. An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014*. AAAI Press, 2014.

[9] Federico Cerutti, Ilias Tachmazidis, Mauro Vallati, Sotirios Batsakis, Massimiliano Giacomin, and Grigoris Antoniou. Exploiting parallelism for hard problems in abstract argumentation. In *AAAI*, pages 1475–1481, 2015.

[10] Günther Charwat, Wolfgang Dvořák, Sarah A Gaggl, Johannes P Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation–a survey. *Artificial intelligence*, 220:28–63, 2015.

[11] Sylvie Coste-Marquis, Sébastien Konieczny, Pierre Marquis, and Mohand Akli Ouali. Weighted attacks in argumentation frameworks. In *KR*, 2012.

[12] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and *n*-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[13] Paul E Dunne and Michael Wooldridge. Complexity of abstract argumentation. In *Argumentation in artificial intelligence*, pages 85–104. Springer, 2009.

[14] Wolfgang Dvořák, Matti Järvisalo, Johannes Peter Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artificial Intelligence*, 206:53–78, 2014.

[15] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence*, 186:1–37, 2012.

[16] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[17] Beishui Liao. Toward incremental computation of argumentation semantics: A decomposition-based approach. *Annals of Mathematics and Artificial Intelligence*, 67(3-4):319–358, 2013.

[18] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[19] Fragkiskos D Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142, 2013.

[20] Keith Robert Matthews. *Elementary linear algebra*. University of Queenland, 2013.

[21] Farid Nouioua and Vincent Risch. Bipolar argumentation frameworks with specialized supports. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, pages 215–218. IEEE, 2010.

[22] Patrick Saint-Dizier. Challenges of argument mining: generating an argument synthesis based on the qualia structure. In *9th International Conference on Natural Language Generation (INLG 2016)*, pages pp–79, 2016.

[23] Camila Pereira Santos, Desiree Maldonado Carvalho, and Mariá CV Nascimento. A consensus graph clustering algorithm for directed networks. *Expert Systems with Applications*, 54:121–135, 2016.

[24] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[25] Edward Tsang. *Foundations of constraint satisfaction: the classic text*. BoD–Books on Demand, 2014.

[26] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.