Argument & Computation 13 (2022) 361–368
DOI 10.3233/AAC-210014
IOS Press

Argument & Computation Community Resources (ACCR) Corner

A tool for merging extensions of abstract argumentation frameworks

Jérôme Delobelle and Jean-Guy Mailly [*,**]
Université de Paris, LIPADE, F-75006 Paris, France
E-mails: jerome.delobelle@u-paris.fr, jean-guy.mailly@u-paris.fr

Abstract. We describe a tool that allows the merging of extensions of argumentation frameworks, following the approach defined by (In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'16)* (2016) 33–42). The tool is implemented in Java, and is highly modular thanks to Object Oriented Programming (OOP) principles. We describe a short experimental study that assesses the scalability of the approach, as well as the impact on runtime of using an integrity constraint.

Keywords: Abstract argumentation, belief merging

In multi-agent systems, merging the beliefs of several agents in order to have a global view of the groups beliefs is an important task [16]. When the agents' beliefs are expressed through argumentation frameworks [11], there are two options: aggregate the graphs (see *e.g.* [6,7]), or merge the extensions [8]. Here we present a Java tool that follows this second approach, then allowing the user to reason with the merged beliefs of the group (*e.g.* for determining whether a given argument is credulously or skeptically accepted w.r.t. the merged beliefs). Notice that the original paper also presents so-called *generation operators*, that allow to obtain new argumentation frameworks from the merged extensions. This part is not covered by our tool.

1. Background: Abstract argumentation

Let us briefly recall basic notions of abstract argumentation [11]. An *abstract argumentation framework* (AF) is a directed graph whose nodes represent *arguments* and edges represent *attacks* between arguments. The origin or the internal structure of arguments are ignored, and their acceptance is purely defined from the (attack) relations between them. Formally, $F = \langle A, R\rangle$, where A is the (finite) set of arguments, and $R \subseteq A \times A$ is the attack relation. We say that b *attacks* a when $(b, a) \in R$. If $(c, b) \in R$ also holds, then c *defends* a against b. Attack and defense can be adapted to sets of arguments: $S \subseteq A$ attacks (respectively defends) an argument $a \in A$ if $\exists b \in S$ that attacks (respectively defends) a.

The acceptability of arguments is usually evaluated collectively through the notion of *extensions*. The semantics used to obtain these extensions rely on two basic concepts: *conflict-freeness* and *defense*.

[*]Corresponding author. E-mail: jean-guy.mailly@u-paris.fr.
[**]The authors thank Lydia Bechroune and Yanis-Fady Akroun for their work on a preliminary version of the software.

Given an AF $F = \langle A, R \rangle$, a set $S \subseteq A$ is conflict-free iff $\forall a, b \in S$, $(a, b) \notin R$, and admissible iff it is conflict-free, and defends each $a \in S$ against its attackers. We use $Ext_{\mathrm{cf}}(F)$ and $Ext_{\mathrm{ad}}(F)$ for denoting the sets of conflict-free and admissible sets of an argumentation framework F. The intuition behind these principles is that a set of arguments may be accepted only if it is internally consistent (conflict-freeness) and able to defend itself against potential threats (admissibility). Then, given an AF $F = \langle A, R \rangle$, an admissible set $S \subseteq A$ is a complete extension iff it contains every argument that it defends; a preferred extension iff it is a $\subseteq$-maximal complete extension; the unique grounded extension iff it is the $\subseteq$-minimal complete extension; and finally a stable extension iff it attacks every argument in $A \setminus S$. The sets of extensions of an AF F, for these four semantics, are denoted (respectively) $Ext_{\mathrm{co}}(F)$, $Ext_{pr}(F)$, $Ext_{gr}(F)$ and $Ext_{st}(F)$. Given a semantics $\sigma \in \{\mathrm{co}, pr, gr, st\}$, we can define the status of any (set of) argument(s), namely *skeptically accepted* (belonging to each σ-extension), *credulously accepted* (belonging to at least one σ-extension) and *rejected* (belonging to no σ-extension).

For more details about argumentation semantics, we refer the interested reader to *e.g.* [2,11].

2. Merging argumentation frameworks

The aggregation of several AFs is an important problem for multiagent systems: each agent may be associated with a different AF on the same set of arguments (*i.e.* each agent may have different views on what constitutes a valid attack, for instance because of different preferences over arguments [15]) that represents his beliefs. The problem is to define a suitable representation of the beliefs of the group. Two families of approaches have been studied: the aggregation of the graphs [6,7,9,17] and the aggregation of extensions [8]. In this paper, we focus on the second type.

In the following, $\mathcal{F} = (F_1, \ldots, F_k)$ denotes the profile (*i.e.* the tuple) of AFs in input representing the group of agents, with each AF built on the set of arguments A. Inspired by belief merging with integrity constraint in propositional logic [16], we use a propositional formula to express a constraint on the extensions of the result. This *integrity constraint* represents some information that must hold in the result of the merging (*e.g.* laws of physics, legal rules,...). The propositional atoms of this constraint are arguments names, and usual connectives (*e.g.* $\neg$, $\vee$, $\wedge$,...) can be used. For instance, the formula $\phi_1 = (a_1 \wedge a_2 \wedge a_3) \vee (a_1 \wedge \neg a_2 \wedge \neg a_3)$ expresses that in the result, a_1 must be accepted and a_2 and a_3 must be both accepted or both rejected. We call $\mathbf{L}_A$ the set of all the formulas built on the set of arguments A. The satisfaction of a formula $\phi \in \mathbf{L}_A$ by a set of arguments $S \subseteq A$ is defined in a classical way: for $a \in S$, the associated propositional atom is evaluated as *true*; for $a \notin S$, the atom is evaluated as *false*. Then, rules of classical logic are used to determine whether S satisfies ϕ or not. The satisfaction of ϕ by S is denoted $S \mathrel{|\!\sim} \phi$. A merging operator is then defined as follows:

Definition 1. An *AF merging operator* Δ is a mapping from a profile $\mathcal{F}$ and an integrity constraint $\mu \in \mathbf{L}_A$ to a set of argumentation frameworks, denoted by $\Delta_\mu(\mathcal{F})$.

The extension-based approach for merging AFs is a two-step process, depicted in Fig. 1. We first compute the extensions of each AF in input, and aggregate them in order to obtain a set of "extensions" (the part in the dashed frame in Fig. 1). Then, we generate the corresponding AFs resulting from the merging process, *i.e.* build one or several AFs whose extensions correspond exactly to the result of the first step (the part out of the dashed frame in Fig. 1).

In this work, we focus exclusively on the aggregation of extensions (first step). The goal of this step is to determine a set of "extensions" that satisfy the integrity constraint, and correspond as closely as possible to the extensions of the AFs in input. We call a *candidate* any set of arguments which may

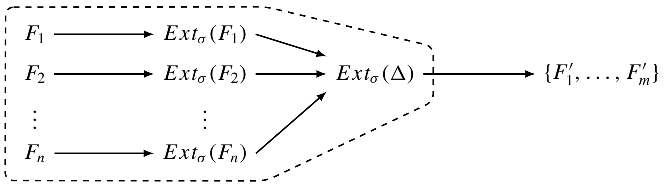


Fig. 1. Extension-based approach to merging of AFs.

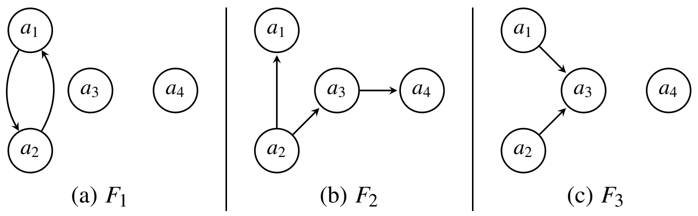


Fig. 2. The profile $\mathcal{F} = (F_1, F_2, F_3)$ of AFs to be merged.

possibly belong to the result of the aggregation. One possible way to define the merging operator is to use distances (or, more generally pseudo-distances) between candidates, as well as aggregation functions.

Definition 2. A *(pseudo-)distance* d between sets of arguments is a mapping $2^A \times 2^A \to \mathbb{R}^+$ such that for each $c_1, c_2 \subseteq A$, $d(c_1, c_2) = 0$ iff $c_1 = c_2$ and $d(c_1, c_2) = d(c_2, c_1)$.

Then, given a candidate c_1 and a set of candidates C, we define $d(c_1, C) = \min_{c_2 \in C}(d(c_1, c_2))$.

Definition 3. An aggregation function is a function $\otimes$ associating a non-negative real number to every finite tuple of non-negative real numbers and satisfying the following properties:

- If $x \leqslant y$, then $\otimes(x_1, \dots, x, \dots, x_n) \leqslant \otimes(x_1, \dots, y, \dots, x_n)$. *(non-decreasingness)*
- $\otimes(x_1, \dots, x_n) = 0$ iff $x_1 = \dots = x_n = 0$. *(minimality)*
- For every non-negative real number x, $\otimes(x) = x$. *(identity)*

Definition 4 (Distance-based Pre-Order over Candidates)**.** Given a semantics σ, a profile $\mathcal{F}$, a (pseudo-)distance between candidates d and an aggregation function $\otimes$, we define the pre-order $\leqslant_{\mathcal{F}}^{\otimes,d}$ between candidates as $c_1 \leqslant_{\mathcal{F}}^{\otimes,d} c_2$ iff $\bigotimes_{F \in \mathcal{F}} d(c_1, Ext_\sigma(F)) \leqslant \bigotimes_{F \in \mathcal{F}} d(c_2, Ext_\sigma(F))$.

The result of the (first step of the) merging is therefore the set of candidates that satisfy the integrity constraint and that are never strictly dominated by another candidate w.r.t. $\leqslant_{\mathcal{F}}^{\otimes,d}$.

Definition 5 (Distance-based Merging Operator)**.** Given a semantics σ, a profile $\mathcal{F}$ on a set of arguments A, an integrity constraint $\mu \in \mathbf{L}_A$, a (pseudo-)distance between candidates d and an aggregation function $\otimes$, the set of candidates representing the result of the aggregation of $\mathcal{F}$ is $Ext_\sigma(\Delta_\mu^{\otimes,d}(\mathcal{F})) = \{c \subseteq A \mid c \mid\!\sim \mu$ and $\forall c'$ s.t. $c' \mid\!\sim \mu, c \leqslant_{\mathcal{F}}^{\otimes,d} c'\}$.

Example 1. We consider the Hamming distance [13] between candidates: $d_H(c_1, c_2) = |(c_1 \setminus c_2) \cup (c_2 \setminus c_1)|$. We want to merge the profile $\mathcal{F} = (F_1, F_2, F_3)$ described at Fig. 2, under the stable semantics, and with the integrity constraint $\mu = a_2 \wedge a_4 \wedge (a_1 \vee a_3)$, which expresses that for each extension, the arguments a_2 and a_4 must belong to it, and at least one of the arguments a_1 and a_3 as well.

Table 1

Aggregated distance between the models of μ and the extensions of the profile of AFs

μ	$Ext_{st}(F_1)$	$Ext_{st}(F_2)$	$Ext_{st}(F_3)$	$\sum$
	$\{a_1, a_3, a_4\}$	$\{a_2, a_4\}$	$\{a_1, a_2, a_4\}$	
	$\{a_2, a_3, a_4\}$			
$\{a_1, a_2, a_4\}$	2	1	0	3
$\{a_2, a_3, a_4\}$	0	1	2	3
$\{a_1, a_2, a_3, a_4\}$	1	2	1	4

So we want to compute $\Delta_{\mu}^{\Sigma,d_H}(\mathcal{F})$. The stable extensions of the AFs are $Ext_{st}(F_1) = \{\{a_1, a_3, a_4\}, \{a_2, a_3, a_4\}\}$, $Ext_{st}(F_2) = \{\{a_2, a_4\}\}$ and $Ext_{st}(F_3) = \{\{a_1, a_2, a_4\}\}$. Table 1 exhibits the distance between each model of μ and the extensions of F_1, F_2 and F_3.

Now the distance between each model c of μ and the profile $\mathcal{F}$ is the sum of the distances between c and each of the AFs in $\mathcal{F}$. We obtain the following distances:

- $\sum_{F\in\mathcal{F}} d_H(\{a_1, a_2, a_4\}, Ext_{st}(F)) = 2 + 1 + 0 = 3$,
- $\sum_{F\in\mathcal{F}} d_H(\{a_2, a_3, a_4\}, Ext_{st}(F)) = 0 + 1 + 2 = 3$,
- $\sum_{F\in\mathcal{F}} d_H(\{a_2, a_3, a_3, a_4\}, Ext_{st}(F)) = 1 + 2 + 1 = 4$.

The candidates which are selected at the merging step are those with a sum of distances equal to 3, *i.e.* $Ext_{st}(\Delta_{\mu}^{\Sigma,d_H}(\mathcal{F})) = \{\{a_1, a_2, a_4\}, \{a_2, a_3, a_4\}\}$.

For more details on the merging approach see [8].

3. Implementation

3.1. Software description

Programming details. The source code (in Java) is available here under the open source MIT license: https://github.com/jeris90/fusionAF. Our implementation is based on the solver jArgSemSat [5] for computing the extensions. For guaranteeing the ease of evolution of the software, it highly relies on the *Factory* design pattern [12]. We exemplify it on the case of distances. The abstract class Distance only provides an abstract method that computes the distance between two sets of arguments. If the user wants to add a new distance to the application, everything s/he has to do is implement a new class MyDistance that inherits from Distance, and to modify the method makeDistance in the class DistanceFactory, by adding the line 6 in the following code:

```
1 public Distance makeDistance(String distance){
2     switch(distance):
3     case "HM": return new DistanceHamming();
4     case "JC": return DistanceJaccard();
5     case "SD": return new DistanceSorensenDice();
6     case "MD": return new MyDistance();
7     default:
8         throw new IllegalArgumentException("...");
9 }
```

The strings ("HM" for Hamming, "JC" for Jaccard,. . .) correspond to the command line parameter for choosing the distance. The functioning is similar for choosing the aggregation operator, and for adding new ones. The application already integrates classical operators, like Min, Max, Sum,. . .

Commandline interface. Our application can be used through the following commandline interface:

```
java -jar jarfile -dir <dir_profile> -f <input_format>
        [-IC <int_constraint>] [-AGG <agg_function>] [-D <distance>]
        [-p <EE|DC|DS>] [-a <argument>]  [-print] [-h]
```

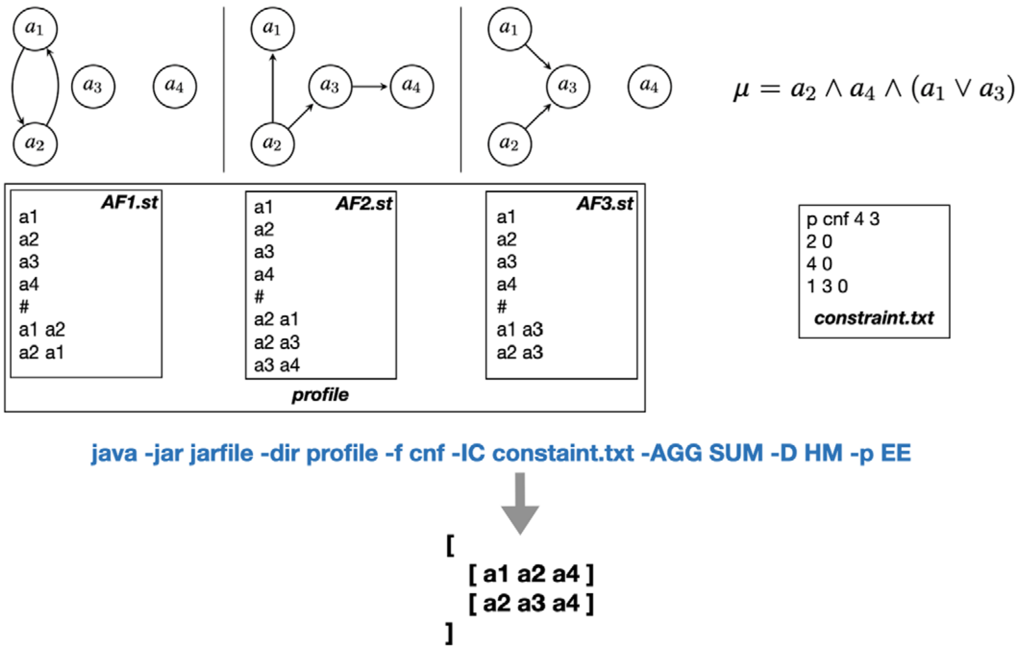
where `jarfile` is the name of the runable `jar` file obtained when compiling the source code. The parameters meanings are:

- `-dir` (or `--dir_profile`) `<dir_profile>` is the mandatory parameter for describing the path to the directory containing the profile of AFs.
- `-f` (or `--input_format`) `<apx|tgf>` is the mandatory parameter for describing the input file format (`apx` or `tgf`).
- `-IC` (or `--int_constraint`) `<int_constraint>` is the path to the file containing the integrity constraint (a CNF formula in dimacs format). If not provided, then the integrity constraint is a tautology (*i.e.* the set of candidates is the power set of the set of arguments).
- `-AGG` (or `--agg_function`) `<AggF>` describes the aggregation function. `AggF` must be `SUM` for sum, `MIN` for minimum, `MAX` for maximum, `MUL` for multiplication, `MEAN` for mean, `MED` for mediane, `LMIN` for leximin, or `LMAX` for leximax.
- `-D` (or `--distance`) `<HM|JC|SD>` is the distance used to compare a candidate and a set of extensions (`HM` for the Hamming distance [13], `JC` for the Jaccard distance [14], `SD` for the Sorensen-Dice distance [10]).
- `-p` (or `--problem`) `<EE|DC|DS>` is used for the choice of the task to be carried out by the program. `EE` means “enumerate the extensions”, while `DC` (respectively `DS`) means “decide the credulous (respectively skeptical) acceptance”.
- `-a` (or `--arg`) `<argument>` is a mandatory option with the option `-p DC` or `-p DS` to specify the targeted argument.
- `-print` prints all details of the aggregation process.
- `-h` (or `--help`) is the help option.

The default parameters are `--distance HM`, `--agg_function SUM`, and `--problem EE`. Notice that every AF in the profile can be evaluated with its own semantics, specified by the extension of the file name. For instance, `af1.apx.st` means that the first AF uses the stable semantics, while `af2.apx.pr` means that the second AF uses the preferred semantics. All the semantics implemented in jArgSemSat are available. Figure 3 is an example of how to use our program (with inputs and output) to merge the profile of AFs described in Example 1.

3.2. *Experimental evaluation*

Protocol. We have conducted a preliminary experimental study in order to know the limits of our implementation with respect to the two complex problems included in our approach, namely the computation of candidates (*i.e.* the enumeration of models of the constraint) and the computation of extensions via extension-based semantics. We have arbitrarily chosen the Barabási-Albert model [1] (BA) to generate the benchmark. For recall, this model provides networks, called scale-free networks, with a structure



$\mu = a_2 \wedge a_4 \wedge (a_1 \vee a_3)$

AF1.st
```
a1
a2
a3
a4
#
a1 a2
a2 a1
```

AF2.st
```
a1
a2
a3
a4
#
a2 a1
a2 a3
a3 a4
```

AF3.st
```
a1
a2
a3
a4
#
a1 a3
a2 a3
```

profile

constraint.txt
```
p cnf 4 3
2 0
4 0
1 3 0
```

```
java -jar jarfile -dir profile -f cnf -IC constaint.txt -AGG SUM -D HM -p EE
```

```
[
  [ a1 a2 a4 ]
  [ a2 a3 a4 ]
]
```

Fig. 3. One detailed input/output example of usage of our software.

Table 2

Mean of execution time (in sec) for each category of profile (in relation to the number of arguments) in the benchmark without integrity constraint (benchmark_WC) and with a given integrity constraint (benchmark_C)

Number of arguments per AF	5	10	15	20	25	30
benchmark_WC	0.442	0.648	2.71	225.765	900	900
benchmark_C	0.431	0.528	0.688	1.29	2.958	300.97

in which some nodes have a huge number of links, but in which nearly all nodes are connected to only a few other nodes. We use the AFBenchGen2 generator [4] to generate 200 random instances for each (numArg, probAttacks) in $\{5, 10, 15, 20, 25, 30\} \times \{0.5\}$ giving a total of 1200 AFs. For each size of AFs, we randomly and equally distributed the AFs in 20 different profiles, which gives us 10 AFs per profile. Concerning the parameters related to the merging of AFs belonging to the same profile, we have used the default parameters, *i.e.* we list all the results (`--problem EE`) with the sum as aggregation function (`--agg_function SUM`) and the Hamming distance (`--distance HM`). The preferred semantics is the only extension-based semantics used. Finally, in order to measure the impact of the integrity constraint on the execution time, we chose to merge a first time without any constraint (*i.e.* the constraint is a tautology) and a second time with an integrity constraint specific to each size of AFs. For each size of AFs $n \in \{5, 10, 15, 20, 25, 30\}$, the constraint contains $n/5$ unit clauses (*i.e.* one specific argument must be in each extension), and $n/5$ (non-unit) clauses that enforce some disjunction between arguments. For instance, for $n = 15$, $\mu = a_4 \wedge a_6 \wedge a_{11} \wedge (a_1 \vee a_3) \wedge (a_2 \vee a_4 \vee a_6 \vee a_9) \wedge (a_2 \vee a_5 \vee a_6 \vee a_7 \vee a_8 \vee a_{12})$. The resources (benchmark and constraints) are available in our Github project.

Results. The results are presented in Table 2. We observe that using an integrity constraint has a positive impact on the runtime, since even profiles of large AFs ($n = 30$) are merged in around 300 seconds in average, while smaller profiles reach the timeout (900 seconds) without an integrity constraint. This can be explained by the exponential number of models of the tautology.

4. Conclusion

In this paper, we describe a modular tool, implemented in Java, that allows to compute the merging approach defined in [8]. More precisely, it takes as input a profile of AFs, a distance, an aggregation function, and an integrity constraint, and it allows to obtain the merged extensions for the input profile w.r.t. the merging operator induced by the distance and the aggregation function. It also allows to determine the (credulous or skeptical) acceptability of a given argument. Experiments show that it scales up well, especially when the integrity constraint allows to reduce the number of potential extensions of the result.

As an interesting future work, we wish to study the impact of other parameters, like the distance, the aggregation function, the semantics, the size of the profile, or the nature of the integrity constraint and the AFs. We also plan to work on an implementation of other aggregation approaches, *i.e.* those based on the aggregation of graphs instead of the merging of extensions. Notice that the approach by [7] has been implemented in [3].[1]

References

[1] A.-L. Barabasi and R. Albert, Emergence of scaling in random networks, *Science* **286**(5439) (1999), 509–512. doi:10.1126/science.286.5439.509.

[2] P. Baroni, M. Caminada and M. Giacomin, Abstract argumentation frameworks and their semantics, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 159–236.

[3] C. Cayrol, S. Doutre and M.-C. Lagasquie-Schiex, GRAFIX: A tool for abstract argumentation, in: *Proceedings of Computational Models of Argument (COMMA)*, 2014, pp. 453–454.

[4] F. Cerutti, M. Giacomin and M. Vallati, Generating structured argumentation frameworks: AFBenchGen2, in: *Proceedings of the 6th International Conference on Computational Models of Argument (COMMA'16)*, 2016, pp. 467–468.

[5] F. Cerutti, M. Vallati and M. Giacomin, An efficient Java-based solver for abstract argumentation frameworks: jArgSemSAT, *Int. J. Artif. Intell. Tools* **26**(2) (2017), 1750002–1175000226. doi:10.1142/S0218213017500026.

[6] W. Chen and U. Endriss, Preservation of semantic properties in collective argumentation: The case of aggregating abstract argumentation frameworks, *Artif. Intell.* **269** (2019), 27–48. doi:10.1016/j.artint.2018.10.003.

[7] S. Coste-Marquis, C. Devred, S. Konieczny, M. Lagasquie-Schiex and P. Marquis, On the merging of Dung's argumentation systems, *Artif. Intell.* **171**(10–15) (2007), 730–753. doi:10.1016/j.artint.2007.04.012.

[8] J. Delobelle, A. Haret, S. Konieczny, J.-G. Mailly, J. Rossit and S. Woltran, Merging of abstract argumentation frameworks, in: *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'16)*, 2016, pp. 33–42.

[9] J. Delobelle, S. Konieczny and S. Vesic, On the aggregation of argumentation frameworks, in: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 2015, pp. 2911–2917.

[10] L.R. Dice, Measures of the amount of ecologic association between species, *Ecology* **26**(3) (1945), 297–302. doi:10.2307/1932409.

[11] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artif. Intell.* **77** (1995), 321–357. doi:10.1016/0004-3702(94)00041-X.

[12] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.

[13] R. Hamming, Error detecting and error correcting codes, *The Bell System Technical Journal* **29**(2) (1950), 147–160. doi:10.1002/j.1538-7305.1950.tb00463.x.

[1] See http://www.irit.fr/grafix.

[14] P. Jaccard, The distribution of the flora in the Alpine zone, *New Phytologist* **11**(2) (1912), 37–50. doi:10.1111/j.1469-8137.1912.tb05611.x.

[15] S. Kaci, L.W.N. van der Torre and S. Villata, Preference in abstract argumentation, in: *Proceedings of the 7th International Conference on Computational Models of Argument (COMMA'18)*, Vol. 305, 2018, pp. 405–412.

[16] S. Konieczny and R.P. Pérez, Merging information under constraints: A logical framework, *J. Log. Comput.* **12**(5) (2002), 773–808. doi:10.1093/logcom/12.5.773.

[17] F.A. Tohmé, G.A. Bodanza and G.R. Simari, Aggregation of attack relations: A social-choice theoretical analysis of defeasibility criteria, in: *Proceedings of the 5th International Symposium on Foundations of Information and Knowledge Systems (FoIKS'08)*, 2008, pp. 8–23. doi:10.1007/978-3-540-77684-0_4.