Artificial Intelligence 252 (2017) 267–294

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

The first international competition on computational models of argumentation: Results and analysis ☆

Matthias Thimm [a,*], Serena Villata [b]

[a] Institute for Web Science and Technologies, Universität Koblenz–Landau, Germany
[b] Université Côte d'Azur, CNRS, Inria, I3S, France

ARTICLE INFO

Article history:
Received 14 June 2016
Received in revised form 15 June 2017
Accepted 18 August 2017
Available online 30 August 2017

Keywords:
Formal argumentation
Algorithms

ABSTRACT

We report on the First International Competition on Computational Models of Argumentation (ICCMA'15) which took place in the first half of 2015 and focused on reasoning tasks in abstract argumentation frameworks. Performance of submitted solvers was evaluated on four computational problems wrt. four different semantics relating to the verification of the acceptance status of arguments, and computing jointly acceptable sets of arguments. In this paper, we describe the technical setup of the competition, and give an overview on the submitted solvers. Moreover, we report on the results and discuss our findings.

1. Introduction

Argumentation is a core technique for humans to reach conclusions in the presence of conflicting information and multiple alternatives. It is used both as a means for persuasion in dialogues as well as one owns deliberation mechanism. An argument can be regarded as some concise set of pieces of information that supports a certain conclusion, such as "As Tweety is a bird and birds usually fly, Tweety supposedly flies". Arguments may support contradicting conclusions—consider e.g. "As Tweety is a penguin and penguins do not fly, Tweety does not fly despite the fact that he is a bird"—and the process of argumentation aims at comparing and weighing arguments and counterarguments and ultimately deciding which arguments prevail. While the field of *argumentation theory* [96] studies the structure and interaction of arguments from a philosophical perspective, within artificial intelligence, the field of *computational models of argumentation* [6,8] has gained some attention in recent years. In general, this field is concerned with logical formalizations of models of argumentation that can be used by automatic reasoning systems to cope with uncertainty and inconsistency. Thus, these models are closely related to approaches to non-monotonic reasoning and offer a novel perspective on those. After some earlier works of e.g. Pollock [82] and Simari & Louie [87], abstract argumentation frameworks have been proposed by Dung [35] as a general and abstract formalism to represent arguments and their interactions and have, since then, been most influential. In abstract argumentation frameworks, arguments are represented as vertices in a directed graph and an arc from a vertex $\mathcal{A}$ to a vertex $\mathcal{B}$ means that $\mathcal{A}$ is a counterargument for $\mathcal{B}$ or that $\mathcal{A}$ "attacks" $\mathcal{B}$. Thus, this model abstracts from most issues of argumentation scenarios—including the inner structure of arguments—and provides a clean formal view on the issue of conflict between arguments. Given an abstract argumentation framework the central question is to decide whether arguments are *acceptable*, i.e., whether they "survive" the attacks of their counterarguments due to backing by other arguments. A set of jointly acceptable arguments is then also called *extension*.

☆ This paper was submitted to the Competition Section of the journal.

* Corresponding author.
E-mail address: thimm@uni-koblenz.de (M. Thimm).

http://dx.doi.org/10.1016/j.artint.2017.08.006
0004-3702/

Abstract argumentation provides a nice framework to discuss issues of non-monotonic reasoning in general as many other non-monotonic formalisms such as default theory and logic programs under the stable model semantics can be cast into abstract argumentation frameworks, cf. [35]. On the other hand, the multitude of different semantics and extensions go beyond the expressivity of previous formalisms and provide a novel general approach to non-monotonic reasoning, cf. e.g. [39]. This makes abstract argumentation frameworks a versatile knowledge representation formalism. Many research topics have been spawned around these frameworks including, among others, semantical issues [4], extensions on support [31], quantitative approaches [38,90,65], and in particular algorithms [30]. The computational challenges of various reasoning problems are vast and range up to the second level of the polynomial hierarchy for certain semantics [40,44]. Among the first implementations for reasoning with abstract argumentation frameworks—which appeared around 2008—were Dungine [88] and ASPARTIX [47]. More followed in the years after and, starting from 2013 up till now, a number of comparative analyses among argumentation solvers have been conducted, e.g., [42,10,43,95,11,12,14,27], in order to address a systematic performance comparison. Following the tradition of the communities of other approaches to knowledge representation and reasoning, such as the SAT and the *Answer Set Programming* (ASP) communities, a public competition for solver evaluation was planned soon after.

This paper reports on the First International Competition on Computational Models of Argumentation (ICCMA'15) which took place in the first half of 2015. The results of the competition had been officially presented at the International Workshop on Theory and Applications of Formal Argument (TAFA'15) which was co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI'15) in Buenos Aires, Argentina. The competition called for solvers on four classical computational problems in abstract argumentation frameworks wrt. the four classical semantics proposed in [35], including enumerating all extensions of a particular semantics and deciding whether a certain argument is contained in all of them. Submitted solvers were evaluated wrt. their runtime performance on these tasks on a series of artificially generated argumentation frameworks.

Abstract argumentation frameworks are arguably the most investigated formalism for formal argumentation. However, there are also formalisms for structured argumentation, such as deductive argumentation [8] and defeasible logic programming [55]. In structured argumentation, arguments are a set of (e.g. propositional) formulas (the support of an argument) that derive a certain conclusion (the claim of an argument). The attack relation between arguments is then derived from logical inconsistency. For ICCMA'15 only problems of abstract argumentation have been considered as this is simple and well-understood formalism for representing computational argumentation. However, considering tracks on structured argumentation may be a worthwhile endeavor for future competitions.

The competition received 18 solvers from research groups in Austria, China, Cyprus, Finland, France, Germany, Italy, Romania, and the UK. The solvers were based on different approaches and algorithmic design patterns to solve problems, ranging from reductions to SAT or ASP problems to novel heuristic algorithms. This paper gives an overview on the setup of the competition, the submitted solvers, and the results. More specifically, the remainder of this paper is organized as follows. In Section 2 we provide some necessary background on abstract argumentation and give an overview on the computational tasks considered in the competition. In Section 3 we describe the technical setup of the competition, including the approach for benchmark generation, the used evaluation methodology, and the technical interface requirements. In Section 4 we give an overview on the submitted solvers. Afterwards, we present and analyze the results of the competition in Section 5 and we discuss the lessons learned from this first experience in Section 6. We conclude with a summary in Section 7. Appendix A provides pseudo code of the graph generators used for creating the benchmark graphs of the competition. Appendix B gives detailed graph-theoretic statistics on the benchmark graphs.

2. Background and competition overview

In the following, we give a brief overview on abstract argumentation, the computational problems considered in the competition, and some brief overviews on answer set programming and satisfiability solving. The latter are intended to provide some formal background on the inner workings of solvers based on reductions to those.

2.1. Abstract argumentation

Abstract argumentation frameworks [35] take a very simple view on argumentation as they do not presuppose any internal structure of an argument. Abstract argumentation frameworks only consider the interactions of arguments by means of an attack relation between arguments.

Definition 1 *(Abstract argumentation framework).* An *abstract argumentation framework* AF is a tuple $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ where Arg is a set of arguments and $\rightarrow$ is a relation $\rightarrow \subseteq \mathsf{Arg} \times \mathsf{Arg}$.

For two arguments $\mathcal{A}, \mathcal{B} \in \mathsf{Arg}$ the relation $\mathcal{A} \rightarrow \mathcal{B}$ means that argument $\mathcal{A}$ attacks argument $\mathcal{B}$. Abstract argumentation frameworks can be concisely represented by directed graphs, where arguments are represented as nodes and edges model the attack relation. Note that we only consider finite argumentation frameworks here, i.e., argumentation frameworks with a finite number of arguments.

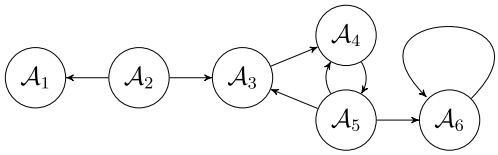

Fig. 1. A simple argumentation framework.

Example 1. Consider the abstract argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ depicted in Fig. 1. Here it is $\mathsf{Arg} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5\}$ and $\rightarrow = \{(\mathcal{A}_2, \mathcal{A}_1), (\mathcal{A}_2, \mathcal{A}_3), (\mathcal{A}_3, \mathcal{A}_4), (\mathcal{A}_4, \mathcal{A}_5), (\mathcal{A}_5, \mathcal{A}_4), (\mathcal{A}_5, \mathcal{A}_3), (\mathcal{A}_5, \mathcal{A}_6), (\mathcal{A}_6, \mathcal{A}_6)\}$.

Semantics are usually given to abstract argumentation frameworks by means of extensions [35]. An *extension* E of an argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ is a set of arguments $E \subseteq \mathsf{Arg}$ that gives some coherent view on the argumentation underlying AF.

In the literature [35,22] a wide variety of different types of semantics has been proposed. In the competition we focused on the four classical semantics of Dung [35], namely grounded, complete, preferred, and stable semantics. For a set of arguments $S \subseteq \mathsf{Arg}$ let $S^- = \{\mathcal{B} \mid \exists \mathcal{A} \in S : \mathcal{B} \rightarrow \mathcal{A}\}$ denote the set of attackers of S and let $S^+ = \{\mathcal{B} \mid \exists \mathcal{A} \in S : \mathcal{A} \rightarrow \mathcal{B}\}$ denote the set of attacked arguments of S.

Definition 2. Let $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ be an argumentation framework.

1. A set of arguments $E \subseteq \mathsf{Arg}$ is *conflict-free* iff there are no $\mathcal{A}, \mathcal{B} \in E$ with $\mathcal{A} \rightarrow \mathcal{B}$.
2. An argument $\mathcal{A} \in \mathsf{Arg}$ is *acceptable* with respect to a set of arguments $E \subseteq \mathsf{Arg}$ iff for every $\mathcal{B} \in \mathsf{Arg}$ with $\mathcal{B} \rightarrow \mathcal{A}$ there is $\mathcal{A}' \in E$ with $\mathcal{A}' \rightarrow \mathcal{B}$.
3. A set of arguments $E \subseteq \mathsf{Arg}$ is an *admissible extension* iff it is conflict-free and all $\mathcal{A} \in E$ are acceptable with respect to E.
4. A set of arguments $E \subseteq \mathsf{Arg}$ is a *complete extension* (CO) iff it is admissible and there is no $\mathcal{A} \in \mathsf{Arg} \setminus E$ which is acceptable with respect to E.
5. A set of arguments $E \subseteq \mathsf{Arg}$ is a *grounded extension* (GR) iff it is complete and E is minimal with respect to set inclusion.
6. A set of arguments $E \subseteq \mathsf{Arg}$ is a *preferred extension* (PR) iff it is complete and E is maximal with respect to set inclusion.
7. A set of arguments $E \subseteq \mathsf{Arg}$ is a *stable extension* (ST) iff it is complete and $E \cup E^+ = \mathsf{Arg}$.

If E is some extension we say that each $\mathcal{A}$ is accepted wrt. E. The intuition behind admissibility is that an argument can only be accepted if there are no attackers that are accepted and if an argument is not accepted then there has to be an acceptable argument attacking it. The idea behind the completeness property is that all acceptable arguments should be accepted. The grounded extension is the minimal set of acceptable arguments and uniquely determined [35]. A preferred extension is a maximal set of acceptable arguments and a stable extension is a complete extension that attacks all arguments not contained in it. Note that for complete, preferred, and stable semantics, their extensions are not necessarily uniquely defined and that for stable semantics an extension does not necessarily exist [35].

For the remainder of the paper we use σ to denote any semantics of GR, CO, PR, ST.

Example 2. Consider again the argumentation framework AF in Fig. 1. The complete extensions of AF are $E_1 = \{\mathcal{A}_2\}$, $E_2 = \{\mathcal{A}_2, \mathcal{A}_4\}$, and $E_3 = \{\mathcal{A}_2, \mathcal{A}_5\}$. Furthermore, E_1 is the grounded extension, E_2 and E_3 are both preferred extensions, and only E_3 is stable.

An alternative approach to define the semantics of an argumentation framework is to use *labelings* instead of extensions [21].

Definition 3. (AF-labeling) Let $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ be an abstract argumentation framework. An AF-labeling is a total function $lab : \mathsf{Arg} \rightarrow \{in, out, undec\}$. We define $in(lab) = \{a_i \in \mathsf{Arg} | lab(a_i) = in\}$, $out(lab) = \{a_i \in \mathsf{Arg} | lab(a_i) = out\}$, $undec(lab) = \{a_i \in \mathsf{Arg} | lab(a_i) = undec\}$.

While extensions only allow for a two-valued assessment of the justification status of an argument—either the argument is in the extension or it is not—labelings allow a three-valued assessment where the additional assessment value "undec" represents an *undecided* assessment. Similar conditions as in Definition 2 can be defined for labelings in order to formalize when a labeling is conflict-free, admissible, complete, etc., e.g., a labeling *lab* is *conflict-free* iff there are no $\mathcal{A}, \mathcal{B} \in in(lab)$ with $\mathcal{A} \rightarrow \mathcal{B}$. Indeed, labeling-based and extension-based semantics are equivalent [21] through the following transformations. If E is a conflict-free (admissible, complete, ...) extension then the labeling *lab* defined through $in(lab) = E$, $out(lab) = E^+$, and $undec(lab) = \mathsf{Arg} \setminus (E \cup E^+)$ is a conflict-free (admissible, complete, ...) labeling. Furthermore, if *lab* is a conflict-free (admissible, complete, ...) labeling then $in(lab)$ is a conflict-free (admissible, complete, ...) extension. For this reason we may use the terms labeling and extension interchangeably.

Table 1
Computational complexity of important decision problems in ($\mathcal{C}$-c denotes completeness for complexity class $\mathcal{C}$). P is the class of decision problems solvable by a deterministic Turing machine in polynomial time; NP (resp. coNP) is the class of decision problems where the YES (resp. NO) instances can be accepted by a non-deterministic Turing machine in polynomial time; Π_2^P is the class of decision problems where the complement can be decided by a non-deterministic Turing machine, that has additionally access to an NP-oracle, in polynomial time, see also [81].

σ	CRED_σ	SKEPT_σ	VER_σ	EXISTS_σ	$\text{EXISTS}_\sigma^{\neg\emptyset}$
CO	NP-c	in P	in P	trivial	NP-c
PR	NP-c	Π_2^P-c	coNP-c	trivial	NP-c
GR	in P	in P	in P	trivial	in P
ST	NP-c	coNP-c	in P	NP-c	NP-c

2.2. Computational problems

The most important decision problems discussed in the context of abstract argumentation are as follows (let σ be any semantics):

CRED_σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ and an argument $\mathcal{A} \in \mathsf{Arg}$
Output: YES iff $\mathcal{A}$ is contained in at least one σ-extension of AF

SKEPT_σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ and an argument $\mathcal{A} \in \mathsf{Arg}$
Output: YES iff $\mathcal{A}$ is contained in all σ-extensions of AF

VER_σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ and a set $E \subseteq \mathsf{Arg}$
Output: YES iff E is a σ-extension of AF

EXISTS_σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$
Output: YES iff AF has at least one σ-extension

$\text{EXISTS}_\sigma^{\neg\emptyset}$ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$
Output: YES iff $\mathcal{A}$ has at least one non-empty σ-extension

The decision problem CRED_σ is about *credulous acceptance* of an argument, i.e., whether it is contained in any σ-extension. The problem SKEPT_σ is about *skeptical acceptance* of an argument, i.e., whether it is contained in all σ-extensions. Furthermore, VER_σ is about verifying whether a given set of arguments is indeed a σ-extension. Finally, the problems EXISTS_σ and $\text{EXISTS}_\sigma^{\neg\emptyset}$ relate to existence problems of extensions. Note that EXISTS_σ is trivial for most semantics except stable semantics, as they guarantee the existence of at least one extension. The harder decision problem $\text{EXISTS}_\sigma^{\neg\emptyset}$ is about checking whether there exist a non-empty σ-extension.

Table 1 gives an overview on the computational complexity of the decision problems discussed above. The results on the grounded semantics as well as $\text{EXISTS}_{\text{PR}}$ and $\text{EXISTS}_{\text{CO}}$ follow immediately from the properties of these semantics shown in [35]. The remaining results for complete semantics are initially by [32]. The results for stable and preferred semantics follow from their corresponding results for logic programming [33], except the SKEPT_{PR} result which is from [36]. For a more detailed discussion of these results and the employed techniques see [41,44]. As can be seen, grounded semantics is the only semantics where all five decision problems are tractable. A naive algorithm for computing the grounded extension can easily be given: first, all arguments that have no attackers are added to an empty extension E and those arguments and all arguments that are attacked by one of these arguments are removed from the framework; then this process is repeated; if one obtains a framework where there is no unattacked argument, the final set E is the grounded extension. Clearly, this is a polynomial algorithm that can be used to solve all the above decision problems wrt. grounded semantics. The problems related to complete and stable semantics usually reside on the first level of the polynomial hierarchy and are thus intractable in practice. Preferred semantics is usually assessed to be computationally harder than the other semantics and particularly the decision problem SKEPT_σ lies on the second level of the polynomial hierarchy.

Functional problems, such as computing all σ-extensions of an argumentation framework AF, have not been investigated much in the literature. This is in line with general research on computational complexity as functional problems may also heavily depend on the size of the output. However, the computational complexity of the corresponding decision problems are usually sufficient to judge the hardness for the functional problems as well.

Still, solving functional problems is important for the actual usability of systems using abstract argumentation and have therefore been considered in the competition as well. More precisely, the problems considered in the competition are given as follows (with the actual naming convention used for the competition):

DC-σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ and an argument $\mathcal{A} \in \mathsf{Arg}$
Output: YES iff $\mathcal{A}$ is contained in at least one σ-extension of AF (equivalent to $\textsc{Cred}_\sigma$)

DS-σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ and an argument $\mathcal{A} \in \mathsf{Arg}$
Output: YES iff $\mathcal{A}$ is contained in all σ-extensions of AF (equivalent to $\textsc{Skept}_\sigma$)

SE-σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$
Output: any σ-extension E of AF or NO if there are no σ-extensions

EE-σ **Input**: An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$
Output: the set $\{E_1, \ldots, E_n\}$ of all σ-extensions of AF

In the above notation, the abbreviation DC stands for "decide credulous", DS for "decide skeptical", "SE" for "some extension", and "EE" for "enumerate extensions". In the following, we refer to DC, DS, SE, and EE as *computational problems* (or simply *problems*) and to a combination of a problem and a semantics, e.g. SE-PR, as a *track*. In the competition we considered the four problems in combination with each of the four discussed semantics, resulting in a total of 16 tracks. For each track, the aim of the competition was to evaluate solvers on how fast instances of these tracks could be correctly solved. Solvers were permitted to enter the competition if they supported at least one of these 16 tracks, but were not obliged to support all of them.

2.3. Argumentation, answer-set programming and satisfiability solvers

In the following, we provide some basics about Answer-set Programming and Satisfiability solvers. This background is intended to support the reader in understanding the main insights of the competition solvers, described in Section 4, implementing such encodings.

2.3.1. Answer-set programming

Answer set programming (ASP) [57] is a modern approach to declarative programming, where a user focuses on declaratively specifying her problem. ASP has its roots in deductive databases, logic programming, logic-based knowledge representation and reasoning, constraint solving, and satisfiability testing. It can be applied in a uniform way to search problems in the classes P, NP, and NP^{NP} in applications like planning, decision support, model checking, and many more.

As discussed in [93], ASP relies upon:

1. the representation of knowledge in terms of disjunctive logic programs with negation as failure (possibly including explicit negation and various forms of constraints),
2. the interpretation of these logic programs under the stable model/answer set semantics and its extensions (dealing with explicit negation and constraints), and
3. efficient computational mechanisms, called ASP solvers, to compute answer sets for grounded logic programs.

We fix a countable set $\mathcal{U}$ of domain elements, called *constants*. An atom is an expression $p(t_1, \ldots, t_n)$, where p is a predicate of arity $n \geq 0$, and each t_i is either a variable or an element from $\mathcal{U}$. An atom is called *ground* if it is free of variables. $B_\mathcal{U}$ denotes the set of all ground atoms over $\mathcal{U}$.

A disjunctive rule r is of the form

$$a_1 | \ldots | a_n \leftarrow b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are literals, and *not* represents default negation. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $B(r) = \{b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m\}$. Furthermore, we have that $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a constraint is normal if $n = 0$. A rule r is safe if each variable in r occurs in $B^+(r)$. A rule r is *ground* if no variable occurs in r. A fact is a ground rule without disjunction and empty body. An input database is a set of facts. A program is a finite set of disjunctive rules. For a program π and an input database D, we write $\pi(D)$ instead of $D \cup \pi$. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground). For any program π, let U_π be the set of all constants appearing in π. $Gr(\pi)$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \pi$, all possible substitutions σ from the variables in r to elements of U_π. An interpretation $\mathcal{I} \subseteq B_\mathcal{U}$ satisfies a ground rule r iff $H(r) \cap \mathcal{I} \neq \emptyset$ whenever $B^+(r) \subseteq \mathcal{I}$ and $B^-(r) \cap \mathcal{I} = \emptyset$. An interpretation $\mathcal{I}$ satisfies a ground program π, if each $r \in \pi$ is satisfied by $\mathcal{I}$. A non-ground rule r (resp., a program π) is satisfied by an interpretation $\mathcal{I}$ iff $\mathcal{I}$ satisfies all groundings of r (resp., $Gr(\pi)$). We have that $\mathcal{I} \subseteq B_\mathcal{U}$ is an answer set of π iff it is a subset-minimal set satisfying the Gelfond–Lifschitz reduct $\pi^\mathcal{I} = \{H(r) \leftarrow B^+(r) | \mathcal{I} \cap B^-(r) = \emptyset, r \in Gr(\pi)\}$. For a program π, we denote the set of its answer sets by $AS(\pi)$.

ASP is particularly well-suited for enumeration problems since these systems enumerate by default all solutions of a given program, thus enabling the enumeration of extensions of an abstract argumentation framework in an easy manner. Moreover, disjunctive ASP is capable of expressing problems being even complete for the second level of the polynomial hierarchy, which is of interest for abstract argumentation considering that several semantics such as the preferred semantics are of this high complexity, cf. the previous section.

Several approaches have been proposed in the literature for computing the extensions of abstract argumentation frameworks using ASP solvers, e.g., [76,47,53,99]. All these approaches rely upon the mapping of an argumentation framework into a logic program whose answer sets are in one-to-one correspondence with the extensions of the original abstract argumentation framework. The approaches differ in the kinds of extensions they focus on, and in the mappings and correspondences they define. For an exhaustive overview, we refer the reader to [93]. In Section 4, we will provide the specific features of the ASP solvers which participated in the competition.

2.3.2. Satisfiability solvers

A propositional formula over a set of Boolean variables is satisfiable iff there exists a truth assignment of the variables such that the formula evaluates to `true`. Checking whether such an assignment exists is the satisfiability (SAT) problem [101]. SAT solvers largely owe their success to efficient search heuristics (e.g., [75]) and conflict-driven backtracking [86].

Let us consider the standard setting of propositional logic over a set $P = \{a, b, c, \ldots\}$ of propositional atoms, and the standard logical connectives $\wedge, \vee, \neg$, denoting conjunction, disjunction, and negation, respectively. A literal is an atom $p \in P$ or its negation $\neg p$. A clause C is a set of literals representing the disjunction $\bigvee_{l \in C} l$. A propositional formula in Conjunctive Normal Form (CNF) is a conjunction of clauses. An interpretation $\mathcal{I} : P \to \{true, false\}$ maps atoms to Boolean values. An interpretation $\mathcal{I}$ satisfies a formula φ ($\mathcal{I} \models \varphi$) if φ evaluates to $true$ under the assignment determined by $\mathcal{I}$. A formula φ is satisfiable if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \varphi$, and unsatisfiable otherwise. A satisfiability solver is a decision procedure which determines whether a given formula φ, in CNF, is satisfiable or not. State-of-the-art SAT solvers are capable of solving instances with hundreds of thousands of literals and clauses. SAT solvers operate in the following way: conflict clauses derived from a previous instance φ can be retained in a subsequent run of the solver on a formula ψ if $\varphi \subseteq \psi$. Moreover, the back-tracking capabilities of SAT solvers make it possible to fix a tentative assignment (or assumption, respectively) for a subset S of A in form of a conjunction of literals over S. Assumptions can be discarded in subsequent calls. This capability to perform iterative calls is crucial to the performance of the SAT-based algorithms proposed for abstract argumentation problems.

One method for using SAT solvers in abstract argumentation is to reduce the argumentation problem at hand to a formula in propositional logic. Reductions of this kind make sophisticated SAT solvers amenable for the field of argumentation. Using classical propositional logic to evaluate abstract argumentation frameworks was first advocated by [7] and then extended to quantified propositional logic [1,49] to efficiently reduce abstract argumentation problems with complexity beyond NP. Several implementations show how modern SAT technology can be used for solving such hard problems in the area of argumentation. In Section 4, we will provide the specific features of the SAT solvers which participated to the competition.

3. Technical setup and evaluation

In this section, we give an overview on how the benchmarks for the competition were generated (Section 3.1), describe the evaluation measures (Section 3.2), and give some details on the execution of the competition (Section 3.3).

3.1. Benchmarks

The availability of real-world benchmarks for argumentation problems was, at the point of time of the competition, quite limited, some few exceptions are [20,19] or AIFdb.[1] However, these benchmarks are tailored towards problems of argument mining [102] and their representation as abstract argumentation frameworks usually lead to topologically simple graphs, such as cycle-free graphs. These kinds of graphs are not suitable for comparing the computational performance of solvers for abstract argumentation problems, as, e.g., all classical semantics coincide with grounded semantics on cycle-free graphs [35], for which all considered computational problems are tractable, cf. Section 2.3. Another possibility to obtain benchmark examples is to utilize other problem areas such as automatic planning, satisfiability, or other reasoning problems and encode these problems as abstract argumentation frameworks. Although these problem transformations generally lead to complex and challenging graphs, they are all structurally similar. As a consequence, solvers optimized for specific graph-theoretic features may have an advantage over other solvers. In order to be able to distinguish the computational performance on all considered semantics and on different graph-theoretic features, we decided to use artificially generated graphs as benchmarks, in line with the preliminary performance evaluation of [10].

In order to provide challenging benchmarks for the abstract argumentation setting, we created three different graph generators called `GroundedGenerator`, `StableGenerator`, and `SccGenerator`, each addressing different aspects

[1] http://corpora.aifdb.org.

of computationally hard benchmark graphs. These graph generators implement heuristic algorithms for generating graphs with specific features such as a large number or size of extensions of a specific semantics. Another possibility to generate graphs for argumentation problems would have been to use exact methods such as algorithms solving the *realizability problem* [37] or the recently proposed method for synthesizing frameworks of [77]. Given a specific set of extensions these methods would construct an argumentation framework with exactly this set of extensions. We decided to use heuristic algorithms instead of these exact methods because of the following two reasons. First, exact methods rely on a deterministic approach to construct an argumentation framework. Consequently, the generated graphs possess similar graph-theoretic features that could be exploited by specific solvers. Although it is possible to alter these algorithms in order to incorporate some randomness this would then lead to heuristic algorithms as well that do not necessarily give the desired result. Second, solving the realizability problem is computationally hard. Initial experiments showed that it was more feasible to run our heuristic algorithms and test whether the resulting graphs have sufficiently good characteristics. We will now briefly outline algorithms for these generators and the features of graphs generated by the algorithms. Pseudo code formalizations of the algorithms can be found in Appendix A.

A: `GroundedGenerator` This graph generator aims at generating graphs with a large grounded extension. As all extensions of all considered semantics always contain the grounded extension [35], graphs generated by this generator test whether solvers can exploit this property to efficiently compute extensions.

Given some upper bound "maxA" for the number of arguments and some predefined probability "p", this generator first randomly determines the actual number "A" of arguments (uniformly distributed in $\{1, \ldots, \text{maxA}\}$) which are named $a_1, \ldots, a_A$. Afterwards, for each pair $i, j = 1, \ldots, A$ with $j < i$ an attack between a_i and a_j is added with probability "p". The resulting intermediate graph component is therefore guaranteed to be acyclic and possibly not fully connected. Afterwards random attacks are added between the not-yet connected arguments and the graph component from before (uniformly distributed).

B: `StableGenerator` This graph generator aims at generating graphs with many stable extensions (and therefore also many complete and preferred extensions). Graphs generated by this generator pose huge combinatorial challenges for solvers addressing the computational tasks of determining (skeptical or credulous) acceptance of arguments and enumerating extensions.

After having determined the number of arguments "A" as in `GroundedGenerator`, this generator first identifies a set of arguments *grounded* to form an acyclic subgraph which will contain the grounded extension. Afterwards another subset M (a candidate for a stable extension) of arguments is randomly selected and attacks are randomly added from some arguments within M to all arguments neither in M nor *grounded*. This process is repeated until a number of desired stable extensions M is reached.

C: `SccGenerator` The third graph generator aims at generating graphs which feature many strongly-connected components and are therefore challenging for solvers which do not rely on decomposition techniques [72].

After having determined the number of arguments "A" as in `GroundedGenerator`, in a first step these arguments are partitioned (with a uniform distribution) into a given number N of components $C_1, \ldots, C_n$. Within each component attacks are added randomly with a high probability given as a parameter (and thus likely forming a strongly connected component). In-between components attacks are randomly added with less probability (also given as parameter), but only from a component C_i to C_j with $i > j$ (in order to avoid having few large strongly connected components).

The source code for the above generators can be found in the source code repository[2] of probo [25], the benchmark suite used to run the competition, which will be discussed in more detail in Section 3.3. In contrast to the preliminary performance evaluation of [10], we decided to use these proprietary graph generators, instead of well-known graph models from network theory such as the Erdös–Rényi [52], Watts–Strogatz [100], or Barabási–Albert [3] models, because of the following reason. Graph models from network theory are designed to explain the topology of e.g. social networks. An important concept often (indirectly) implemented in graph models is that of *triangle closure*, i.e., the tendency of nodes directly connecting to the neighbors of its neighbors (as in the saying "the friend of my friend is also my friend"). From the perspective of challenging benchmarks for abstract argumentation, this feature often trivializes computation. Initial experiments suggest that the generated graphs contain empty or very small grounded extensions, usually no stable extensions, and very few and small complete and preferred extensions. The latter observation is also due to the fact that these graph models aim at modeling the "small world" property of many real-world graphs.[3] This leads to many arguments directly or indirectly being in conflict with each other.

For each of the three benchmark generators A, B, and C, we generated 72 argumentation graphs of different sizes and partitioned each set into three equal-sized subsets of small, medium, and large instances. This results in 9 test sets, each having 24 argumentation graphs (see Appendix B for the exact numbers of arguments and attacks). We conducted some

[2] http://sourceforge.net/p/probo/code/HEAD/tree/trunk/src/net/sf/probo/generators/.

[3] This property basically states that there are always "relatively short" paths from any node to every other node [100]; for example the theory of "six degrees of separation" suggests that in the social network of the known world the longest shortest path between any two persons is six.

preliminary experiments using alpha versions of available solvers in order to check whether these graphs are not too easy or too hard. There we discovered that the test set corresponding to the largest argumentation graphs generated by B was too difficult for every solver. As a consequence, the whole test set was removed from the evaluation. All other test sets seemed to be appropriate to be used for the actual competition. Therefore, the 192 instances used for the evaluation in the competition consisted of

- 24 small-sized argumentation graphs from generator A (test set 1)
- 24 medium-sized argumentation graphs from generator A (test set 2)
- 24 large-sized argumentation graphs from generator A (test set 3)
- 24 small-sized argumentation graphs from generator B (test set 4)
- 24 medium-sized argumentation graphs from generator B (test set 5)
- 24 small-sized argumentation graphs from generator C (test set 6)
- 24 medium-sized argumentation graphs from generator C (test set 7)
- 24 large-sized argumentation graphs from generator C (test set 8)

All argumentation graphs of the competition can be downloaded from the competition website.[4] Appendix B gives some more detailed statistics on the benchmark graphs. For a discussion on the relationships between these statistics and argumentation-specific properties see [95].

3.2. Evaluation measures

The aim of the competition was to measure and compare the computational performance of the submitted solvers on solving instances of the problems presented in Section 2.3. For the problems SE (compute some extension) and EE (compute all extensions) we used every one of the 192 argumentation graphs (see previous section) as an individual instance for each semantics. For the problems DS (decide skeptical acceptance) and DC (decide credulous acceptance) we randomly selected three arguments from every argumentation graph, yielding in total 576 instances for each semantics.

For each instance of a track, a solver was given 10 minutes time to compute the answer. In case of a timeout or a wrong answer, the solver received zero points for this instance.[5] If it gave the correct answer within the time limit, it received one point and the actual runtime for solving the instance was saved separately. For each track, the cumulative number of points was used as the main ranking criterion (solvers which received more points were ranked higher than solvers with less points). If two or more solvers reached the same number of points, their cumulative runtimes on all correctly solved instances were compared to break ties (solver with smaller total runtime were ranked higher than solvers with larger total runtime). This procedure results in a total of 16 rankings of the solvers, one for each track.

For those solvers, which supported all 16 tracks of the competition, we aggregated their scores in the individual tracks to obtain a global ranking using Borda count. For that, every solver received one point for every first place in any ranking, two points for every second place in any ranking, and so on. A global ranking was obtained by ordering the resulting total number of points from smallest to largest.

3.3. Competition details

The competition was realized using the benchmark framework probo [25], which provides the possibility to run the instances on the individual solvers, verify the results, measure the runtime, and log the results accordingly. The software probo is written in Java and requires the implementation of a simple command line interface from the participating solvers.[6]

All benchmark graphs were made available in two file formats. The *trivial graph format*[7] (TGF) is a simple representation of a directed graph which simply lists all appearing vertices and edges. The *ASPARTIX format* (APX) [47] is an abstract argumentation-specific format which represents an argumentation framework as facts in a logic programming-like way.

In order to verify the answers of solvers, the solutions for all instances were computed in advance using the *Tweety libraries for logical aspects of artificial intelligence and knowledge representation* [91]. Tweety contains naive algorithms for all considered semantics that implement the formal definitions of all semantics in a straightforward manner and thus provides verified reference implementations for all considered problems.

Besides serving as the benchmark framework for executing the competition, probo also contains several abstract classes and interfaces for solver specification that could be used by participants in order to easily comply with the solver interface

[4] http://argumentationcompetition.org/2015/iccma2015_benchmarks.zip, note that the test sets are numbered differently in the competition and on the website; in particular, test sets 6, 7, and 8 from above are numbered 7, 8, and 9 there to accommodate for the removed test set 6.

[5] The initial policy for wrong answers was to disqualify the solver completely for the track. However, quite a few solvers occasionally produced wrong results and in order to provide a comprehensive picture on the state-of-the-art we revised this policy; the final results would differ only slightly when enforcing this policy though (see Tables 5–8).

[6] See http://argumentationcompetition.org/2015/iccma15notes_v3.pdf for the formal interface description.

[7] http://en.wikipedia.org/wiki/Trivial_Graph_Format.

Table 2
Supported tracks of the participating solvers.

No.	Solver	SE				EE				DC				DS			
		CO	PR	GR	ST	CO	PR	GR	ST	CO	PR	GR	ST	CO	PR	GR	ST
1	LabSATSolver	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2	ArgSemSAT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	ArgTools	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	Cegartix		✓				✓								✓		
5	Dungell	✓	✓	✓	✓	✓	✓	✓	✓								
6	ZJU-ARG						✓	✓									
7	ASPARTIX-V		✓				✓								✓		
8	CoQuiAAS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
9	ASPARTIX-D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	ConArg	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓			✓
11	GRIS		✓	✓			✓	✓			✓	✓			✓	✓	
12	ASGL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
13	LamatzSolver							✓									
14	ProGraph				✓								✓				
15	DIAMOND	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
16	Carneades	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
17	prefMaxSAT						✓										
18	ASSA				✓				✓				✓				✓

specification. We also provided a tutorial[8] and a generic shell-script[9] that implements the solver interface specification, in order to enable participants to implement their solvers in a way that is compatible with the competition requirements.

The competition itself was executed on a cloud computing platform available at the University of Koblenz–Landau, which provides 320 cores with 2.9 GHz each and 2.2 TB of usable RAM. For each test set of benchmark graphs we used a single virtual machine with 1 CPU and 8 GB of RAM to run all tracks on the set. The results for all tracks were aggregated afterwards.

4. Participants

In this section, we provide a description of the solvers which participated in the competition, and we classify them with respect to their supported tracks. Note that the solvers are numbered according to their registration number.

Table 2 gives an overview on the participating solvers, and their supported tracks.

Table 3 gives some further information on the solvers, i.e., development country, programming language and paradigm, total number of lines of code, and a reference to contributions describing the solver in more detail. Most of the solvers have been developed in Europe with the exception of the ZJU-ARG solver from China and the ArgTools solver from Jordan. Moreover, we can note that solvers have been developed mainly using logic and object-oriented programming languages.

Table 4 lists the solvers' license information (when available), and provides a link to their source code repositories. Most of the repositories are available under the GNU GPL license, while some of them chose more specific licenses like the MIT license. In general, however, the source code of all solvers participating in the competition has been made available for research purposes, which was also a requirement for participating.

In the following, we give some details on the individual solvers that participated in the competition. For the complete system descriptions, we refer the reader to [92].

LabSATSolver

The LabSAT solver [17] solves all tasks of the competition wrt. all semantics. It encodes the labeling approach [21] as a boolean satisfiability problem (SAT) following the proposal of Cerutti and colleagues [23]. Roughly, the approach proposed in [23], called PrefSAT, is a depth-first search in the space of complete extensions to identify those that are maximal, namely preferred extensions and enumerate them. Each step of the search process requires the solution of a SAT problem through the invocation of a SAT solver. The algorithm is based on the idea of encoding the constraints corresponding to complete labelings of an AF [21] as a SAT problem, and then iteratively producing and solving modified versions of the initial SAT problem according to the needs of the search process. For more details about the encoding, we refer the reader to [23]. Complete and preferred extensions are computed by the LabSAT solver using the PrefSAT approach [23]. To compute the stable extension, additional clauses are added to the SAT solver (i.e., the label *undec* is excluded). The SAT solver used in LabSAT is lingeling [9]. It is worth noticing that the grounded semantics is computed without the use of the SAT solver, relying on a Java implementation of the algorithm proposed in [74].

[8] http://argumentationcompetition.org/2015/iccma15probotutorial_v2.pdf.
[9] http://sourceforge.net/p/probo/code/HEAD/tree/trunk/solvers/generic-interface.sh?format=raw.

Table 3
Detailed information of the participating solvers.

No.	Solver	Country	Programming language	Programming paradigm	# Lines of code	Reference
1	LabSAT	Germany	Java	object-oriented	1300	[17]
2	ArgSemSAT	UK, Italy	C++	object-oriented	7800	[26]
3	ArgTools	Jordan, UK	C++	object-oriented	6000	[79]
4	Cegartix	Austria, Finland	C++	object-oriented	850	[45]
5	Dungell	UK	Haskell	direct/functional	240	[97]
6	ZJU-ARG	China, Luxembourg	Java	object-oriented	2100	[62]
7	ASPARTIX-V	Italy, Finland, Austria	ASP	logic	750	[85]
8	CoQuiAAS	France	C++	object-oriented	1600	[66]
9	ASPARTIX-D	Germany	ASP	logic	640	[54]
10	ConArg2	Italy	C++	object-oriented	3644	[13]
11	GRIS	UK	C++	object-oriented	2190	[83]
12	ASGL	Germany	Lisp	functional	900	[89]
13	LamatzSolver	Germany	Java	object-oriented	550	[68]
14	ProGraph	Romania	Prolog	logic	305	[61]
15	DIAMOND	Germany	Python/ASP	logic	420	[51]
16	Carneades	Germany	Go	N/A	1242	[58]
17	prefMaxSAT	UK, Italy	C++	object-oriented	750	[94]
18	ASSA	Cyprus	Java	object-oriented	N/A	[63]

Table 4
Licenses and source code repositories of the participating solvers.

No.	Solver	License	Source code repository
1	LabSAT	GNU LGP	https://github.com/fbrns/LabSATSolver
2	ArgSemSAT	MIT	http://sourceforge.net/projects/argsemsat/
3	ArgTools	GNU GPL	http://sourceforge.net/projects/argtools/
4	Cegartix	GNU GPL	http://www.dbai.tuwien.ac.at/research/project/argumentation/cegartix/
5	Dungell	BSD3	http://www.cs.nott.ac.uk/~psxbv/DungICCMA/
6	ZJU-ARG	N/A	http://mypage.zju.edu.cn/en/beishui/685664.html
7	ASPARTIX-V	N/A	http://www.dbai.tuwien.ac.at/proj/argumentation/systempage/
8	CoQuiAAS	GNU GPL	http://www.cril.univ-artois.fr/coquiaas/
9	ASPARTIX-D	ad-hoc	https://ddll.inf.tu-dresden.de/web/Sarah_Alice_Gaggl/ASPARTIX-D
10	ConArg	N/A	http://www.dmi.unipg.it/conarg/
11	GRIS	GNU GPL	http://www.inf.kcl.ac.uk/staff/odinaldo/gris/
12	ASGL	ad-hoc	https://github.com/kisp/asgl
13	LamatzSolver	N/A	https://bitbucket.org/Cloudkenny/lamatzsolver/
14	ProGraph	N/A	http://cs-gw.utcluj.ro/~adrian/tools/prograph/ProGraph-ArgComp2015.tar
15	DIAMOND	GNU GPL	http://diamond-adf.sourceforge.net/
16	Carneades	MPL 2.0	https://carneades.github.io/
17	prefMaxSAT	MIT	http://sourceforge.net/projects/prefmaxsat/
18	ASSA	N/A	http://www.mertjiandata.com/assa.html

ArgSemSAT

The ArgSemSAT solver [26,23,28,29] implements a collection of algorithms for solving all tasks of the competition wrt. all semantics. ArgSemSAT-1.0 encodes the constraints corresponding to complete labelings of an AF as a SAT problem, and then iteratively produces and solves modified versions of the initial SAT problem according to the needs of the search process. As for the LabSAT solver, also ArgSemSAT implements the PrefSAT approach [23] described above, for enumerating the preferred extensions. PrefSAT first solves a SAT problem whose solutions correspond to the complete extensions of an AF, and second, a hill-climbing approach is used to find a maximal wrt. set inclusion complete extension, i.e., a preferred extension. Already computed extensions are excluded from subsequent search steps. In addition, ArgSemSAT-1.0 implements the SCC-P algorithm [24] exploiting the SCC-recursiveness schema [5] using the partial order of strongly connected components (SCCs). In SCC-P, the extensions of the frameworks restricted to the SCC not receiving any attack are computed and combined together. Then, each SCC which is attacked only by such unattacked SCCs is considered: the extensions of such a SCC are computed and merged with those already obtained. Finally, the subsequent (wrt. the partial order) SCCs are considered until no remaining SCCs are left. The schema is recursive, and the base of the recursion is reached when there is only one SCC: in this case a solver similar to PrefSAT is called. It is worth noticing that SCC-P resulted to be more efficient than PrefSAT on AFs with numerous SCCs. ArgSemSAT-1.0 exploits the Glucose solver [2] and the PrecoSAT[10] solver.

[10] http://fmv.jku.at/precosat/.

ArgTools

The ArgTools solver (Argumentation Tools) [79,78] is a system based on backtracking algorithms for solving all tasks of the competition wrt. all semantics. The backtracking algorithms of ArgTools are based on exploration of an abstract binary search tree. The two key features of ArgTools are *i)* to enhance the backtracking search for sets of acceptable arguments by a new pruning strategy, called the global looking-ahead strategy, and *ii)* to set out a backtracking-based approach to decide acceptance under different semantics, i.e., whether an argument is in some/all set(s) of acceptable arguments of a given AF, without necessarily enumerating all such sets. Roughly, the global looking-ahead pruning strategy enables a backtracking procedure during traversing the search space to regularly look-ahead for dead-ends, i.e., for paths that do not lead to solutions, early enough such that considerable time is saved. For more details about this strategy, we refer the reader to [80].

CEGARTIX

The CEGARTIX (Counter-Example Guided Argumentation Reasoning Tool) solver [45,43] supports the computation of credulous acceptance under semi-stable, and stage semantics, the skeptical acceptance under preferred, semi-stable, and stage semantics, it returns an arbitrary preferred extension, and enumerates all preferred extensions. Note that only the part regarding the preferred semantics is relevant for the participation in the competition. Each step in the exploration is delegated to a complete Boolean satisfiability (SAT) solver. The strategy exploited by this solver consists first in the identification of first-level fragments (NP/coNP layer) of second-level reasoning tasks for two main reasons: *i)* such fragments present particular sources of complexity of the considered problems, and *ii)* NP fragments can be efficiently reduced to the SAT problem. CEGARTIX uses the NP decision procedures as NP *oracles* in an iterative way. For problems complete for the second level of the polynomial hierarchy, this leads to general procedures which, in the worst case, require an exponential number of calls to the NP oracle, which is indeed unavoidable under the assumption that the polynomial hierarchy does not collapse. Nevertheless, such procedures can be designed to behave adequately on input instances that fall into the considered NP fragment and on instances for which a relatively low number of oracle calls is sufficient. CEGARTIX exploits current state-of-the-art conflict-driven clause learning (CDCL) SAT-solver technology as the underlying NP oracle. CEGARTIX employs the CDCL SAT-solver Minisat [46]. For more details about the NP decision procedure, we refer the reader to [43].

Dungell

The Dungell solver [97,98] supports the computation of some and all grounded, complete, preferred and stable extensions. The characterizing feature of Dungell consists in providing a solver that is as close to the mathematical definitions as possible. The rationale behind this feature is to tackle the problem of implementing structured argumentation models and their translations by providing a framework that allows implementation close to the mathematical specification and thus facilitates checking and formal proof of properties. Dungell implements two steps in the pipeline: first, it allows for the translation of a Carneades [59] structured argumentation framework into an abstract one, and second, it computes the extensions for the grounded, complete, preferred and stable semantics. Given an AF, it is possible to verify whether a *list* of arguments is conflict-free by checking that the list of attacks between those arguments is empty. Acceptability of an argument with respect to a set of arguments in an AF can be determined by verifying that all its attackers are in turn attacked by an attacker in that set. This solver is one of the very few ones using functional programming, specifically Haskell, for the implementation of structured and abstract models of argumentation.

ZJU-ARG

The ZJU-ARG solver [62,70] enumerates all preferred extensions, and the grounded extension of an AF. It adopts a divide-and-conquer strategy. As for the LabSATSolver, the grounded extension of an AF is computed directly by following the algorithm proposed by [74]. The main feature of the ZJU-ARG solver is the application of the notion of modularity to an argumentation framework, close to the SCC concept, as for other solvers like ArgTools. To overcome the fact that the efficiency of the SCC approach is highly limited by the maximal SCC of an AF, the solver implements a solution by exploiting the most skeptically rejected arguments of an AF. Roughly, given an AF, its grounded labeling [21] is first generated. Then, the attacks between the undecided arguments and the rejected arguments are removed. It turns out that the modified AF has the same preferred labeling as the original AF, but the maximal SCC in it could be much smaller than that of the original AF. Since the ZJU-ARG solver adopts a divide-and-conquer strategy without employing more efficient algorithms to compute the semantics of each sub-framework, its efficiency highly depends on the topologies of the argumentation frameworks in input.

ASPARTIX-V

The ASPARTIX-V solver (Answer Set Programming Argumentation Reasoning Tool–Vienna version) [85,48] supports the computation of skeptical acceptance under preferred semantics, returns a single preferred extension, and enumerates all preferred extensions. Together with an ASP encoding for preferred semantics, the answer-sets are in a 1-to-1 correspondence with the preferred extensions of the given argumentation framework AF. ASP solvers themselves offer enumeration of all answer-sets and returning an arbitrary one. In ASPARTIX-V, a single program is used to encode a particular argumentation semantics, while the instance of an argumentation framework is given as an input database. ASPARTIX-V improves the

performances of its predecessor ASPARTIX[11] as follows. While preferred semantics is encoded as a disjunctive logic program making heavy use of so-called loop constructs in ASP in the ASPARTIX system, ASPARTIX-V is able to do without and uses conditional literals for enhancing the performance. Intuitively, conditional literals allow to use, e.g., a dynamic head in a disjunctive rule that contains a literal iff its condition is true. The loop constructs can be avoided by alternative characterizations of preferred semantics. ASPARTIX-V employs the so-called saturation technique: in the encodings for preferred semantics, a first "guess" is made for a set of arguments in the AF, and then the solver verifies if this set is admissible. To verify if this set is also a subset-maximal admissible one, ASPARTIX-V performs a second guess and verifies if this second guess is an admissible set that is a superset of the first guess. The idea is to keep this second guess small to overcome computational overhead. Additional rules then verify if the witness set represents an admissible set that may be combined with the first guess to result in a larger admissible set. If this is the case, the first guess does not represent a preferred extension. The underlying ASP solver is Clingo 4.4 [56].

CoQuiAAS

The CoQuiAAS solver [66,67] solves all tasks of the competition wrt. all semantics by exploiting constraint programming techniques. More precisely, it takes advantage of the encodings proposed by [7]. CoQuiAAS deals with encodings in Negation Normal Form (NNF) formulae, meaning some propositional formulae where the negation operator is only applied on variables. As CoQuiAAS uses SAT solvers, which are only able to tackle propositional formulae in CNF, a translation step from NNF to CNF is required between the encodings which exist in the literature and the ones that used in the system. An interesting question for SAT solvers is to determine an interpretation which maximizes the number of satisfied constraints: this problem is called Max-SAT. We can generalize this problem, giving a weight to each constraint (Weighted Max-SAT), and if some constraints have an infinite weight (i.e., they have to be satisfied), then the problems are said to be "partial" (Partial Max-SAT). CoQuiAAS uses CNF formulae to solve problems from the first level of the polynomial hierarchy, and some encodings in the Partial Max-SAT formalism for higher complexity problems. Discovering an optimal solution of a Max-SAT instance allows to determine a set of constraints from the initial formula which is consistent, such that adding any other constraint from the initial problem makes this new problem inconsistent: a set of constraints which has this property is called a maximal satisfiable subset (MSS). The optimal solutions of the Max-SAT problem are only a subset of all the MSS of a formula. The solver approaches argumentation semantics exploiting SAT and MSS extraction. CoQuiAAS incorporates the software coMSSExtractor [60] to perform the constraint-based process.

ASPARTIX-D

The ASPARTIX-D solver (Answer Set Programming Argumentation Reasoning Tool–Dresden version) [54] is a collection of ASP encodings together with dedicated solvers to solve all tasks of the competition wrt. all semantics. The general approach of ASPARTIX-D is the same the approach of ASPARTIX-V described above but differs in several details. In particular, the ASP encodings used by ASPARTIX-D are those described in [48] and the optimization applied has been presented in [42]. The main aim of the solver is to find the most suitable encodings *and* solver configuration. ASPARTIX-D exploits the potassco ASP solvers.[12]

ConArg

The ConArg (Argumentation with Constraints) solver (version 2.0) [13,15] allows to enumerate all conflict-free, admissible, complete, stable, grounded, preferred, semi-stable, ideal, and stage extensions, to return one extension given one of these semantics, to check the credulous and skeptical acceptance for the conflict-free, admissible, complete, and stable semantics. It is a constraint programming tool where the properties of the semantics are encoded into constraints, and arguments are assigned to 1 (true) if they belong to a valid extension for that semantics, and 0 otherwise. Searching for solutions takes advantage of classical techniques, such as local consistency through constraint propagation, different heuristics for trying to assign values to variables, and a complete search-tree with branch-and-bound. To map an argumentation framework AF to a Constraint Satisfaction Problem (CSP),[13] which is defined by a set of constraints defined over the a set of variables each with domain D, ConArg defines a variable for each argument in the AF, and each of these arguments can be taken or not in an extension, i.e., the domain of each variable is $D = \{1, 0\}$. As an example, preferred extensions are found by assigning as more arguments as possible to 1 while searching for complete extensions. The solver exploits a toolkit called Gecode 4.4.0[14] defined for developing constraint-based systems and applications.

GRIS

The GRIS (Gabbay–Rodrigues Iterative Solver) solver [83,84] allows to produce one or all of the extensions of the argumentation framework under the grounded and preferred semantics, and given an argument a to decide whether it is accepted credulously or skeptically according to one of these two semantics. The peculiarity of the GRIS solver is that it

[11] http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/.
[12] http://potassco.sourceforge.net.
[13] A CSP is a triple $P = \langle V, D, C \rangle$, where C is a set of constraints defined over the variables in V, each with domain D.
[14] http://www.gecode.org.

works with numerical argumentation networks where arguments are given initial values in the interval $[0, 1]$ from which equilibrium values are calculated iteratively yielding traditional extensions through the use of a characterization result [83]. An argumentation framework is represented in GRIS by means of a set of *equations*. As for ArgTools and the ZJU-ARG solver, also GRIS exploits strongly connected components to compute the extensions. More precisely, the solver starts by computing the strongly connected components of the network and arranging them into layers that can be used in successive computation steps, following the idea of [71]. Once the layers are computed, the solver can identify the deepest layer of computation needed according to the layer depth of the input argument and this can be used to terminate the computation of decision problems as early as possible.

ASGL

The ASGL solver [89] solves all tasks of the competition wrt. all semantics. ASGL uses an extension-based encoding for solutions. As for ConArg, also ASGL casts the argumentation framework as a CSP. Constraints are formalized in a so-called *computation space* and the algorithm is based on propagation methods to reach a fixpoint. The task of computing some preferred extension is implemented in ASGL like a classical optimization problem with branch-and-bound search (already part of standard Gecode). As soon as one solution is found, all further solutions are constrained to be better than the current solution. If no more solution can be found, the current solution is maximal. To efficiently enumerate all preferred extensions, a filtering over all complete extensions for maximality is performed. The ASGL solver, in one out of the two solvers participating to the Competition, that were written using a functional language (Lisp). It also features an interface to the Lingeling SAT Solver as an alternative solver backend. Also ASGL, together with ConArg, exploits the Gecode generic CSP solver library.

LamatzSolver

The LamatzSolver system [68] is a solver for computing the grounded extension of argumentation frameworks based on a direct implementation of the *characteristic function* [35]. Sets like attacks are implemented as a HashMap in Java. More precisely, the computation of the grounded extension is addressed along with the following steps: it checks if the HashMap *typeZero* (containing unattacked arguments) is empty. If the answer is positive, than an empty HashMap called the *grounded* is returned, otherwise the algorithm copies each argument of *typeZero* to the HashMap *grounded*. The size of *grounded* is stored in a parameter *prev* and for each argument the defended arguments will be determined and added to the *grounded*. According to this process, the algorithm keeps track of the arguments attacked by the arguments in *grounded*. These arguments are stored in a HashMap *out*, and the attacks of these arguments are candidates for the grounded extension. The algorithm checks if all attackers of a candidate are defeated. These steps are repeated for the *grounded* until it does not grow anymore. Finally, the *grounded* Hash Map is returned.

ProGraph

The ProGraph solver [61] allows to compute some extension and decides whether an argument is credulously inferred, both with respect to the stable semantics. The key feature of the ProGraph solver is that of relying on bipartite graphs. More precisely, the set of arguments is partitioned in two classes: *in* and *out*. The idea behind is that determining an extension which attacks every argument which is not in that extension can be reduced to a relaxed partitioning problem in which the initial set of arguments is split in two partitions, with the arguments from the second partition being free to attack each other. Arguments are sorted such that they will be placed from the one who attacks the most to the one who attacks the less arguments. The algorithm starts by picking a non-attacked argument and adds it to the attackers extension, and then checks if any of the arguments attacked by the selected one is in the first partition. If this is the case, the algorithm starts backtracking. Otherwise, the arguments attacked by the current argument are added to the second partition. These steps are repeated until all arguments are partitioned or until all paths fail. If only attacked arguments are left, the algorithm chooses one of them and supposes it is not attacked. A verification step stops the algorithm if at some point the attacker is to be placed in the second partition.

DIAMOND

The DIAMOND solver [51,50] solves all tasks of the competition wrt. all semantics. DIAMOND employs the declarative programming paradigm of ASP, and the knowledge representation languages implemented in the solver are Abstract Dialectical Frameworks (ADFs) [16], which are a generalization of AFs. In addition to the computation of the semantics of an AF, DIAMOND can also compute the semantics for (bipolar) ADFs in various different input formats, decide whether a given ADF is bipolar, or transform an ADF from one representation into another. The encodings for DIAMOND are built in a modular way. To compute the models of an ADF with respect to a semantics, different modules need to be grounded together to get the desired behavior. As for other solvers described above, DIAMOND exploits the potassco ASP solvers.[15]

Carneades

The Carneades solver [58,59] solves all tasks of the competition wrt. all semantics. Note that Carneades originally included an implementation of a solver for Dung-like abstract argumentation frameworks, using grounded semantics, despite

[15] http://potassco.sourceforge.net.

the focus of the Carneades project has not been abstract argumentation, but rather structured argumentation. The implementation closely follows high-level specifications of abstract argumentation frameworks, and has not been optimized with the exception of the grounded semantics where the implementation keeps track of whether a mutable labeling has changed, in its main loop, and exits the loop when no changes are made, without having to explicitly test whether two labelings are equivalent. The solver's procedures are implemented for finding the first subset of arguments which satisfy a given predicate and for applying some procedure to each subset. Using functions implementing predicates for complete and stable extensions, Carneades finds the first or all complete extensions and filter the complete extensions to find one or more which are also stable. Argument sets are represented as Hash tables, from arguments to Boolean values.

prefMaxSAT

The prefMaxSAT solver [94,24] allows to compute the extension enumeration problem for the preferred semantics. It implements an encoding of preferred extensions search using unweighted MaxSAT. The algorithm exploited in prefMaxSAT is based on the idea of encoding the constraints corresponding to admissible labelings of an AF as a MaxSAT problem, and then iteratively producing and solving modified versions of the initial problem. If at any one step a variable assignment that maximally satisfies the formula is returned, the corresponding labeling is saved in the list of found preferred extensions. Then a hard clause for eliminating the solution is added to the formula and the process is repeated. If no further solution could be found, prefMaxSAT ends and provides the set of found preferred extensions. Each step of the search process requires the solution of a MaxSAT problem. The AF is encoded in a CNF and is then provided to the MaxSAT solver. If a variable assignment that maximally satisfies the formula is returned, then *i)* the corresponding labeling is saved in the list of found preferred extensions; *ii)* a clause for eliminating the solution is added to the CNF; *iii)* a clause forcing to include different arguments is added to the CNF, and finally, *iv)* the process is repeated. If the MaxSAT solver returns that no variable assignment satisfies the constraints, prefMaxSAT ends and provides the set of found preferred extensions. As for ArgSemSAT, also prefMaxSAT exploits the Glucose solver [2].

ASSA

The $\mathcal{ASSA}$ solver [63,64] computes one or all extensions and decide whether an argument is credulously or skeptically inferred with respect to the stable semantics. The solver implements an approach based on mathematical matrix operations to solve abstract argumentation problems. The system first creates a matrix representation of an AF. Then, all possible instances of the selected arguments are created and combined into another matrix S. Based on matrix operations, and more specifically, on left and right matrix multiplication, it is possible to navigate inside the AF to find which arguments attack the other arguments and which arguments are under attack. By constructing a matrix multiplication, it is possible to determine whether a given set of arguments is conflict-free. Using this method, all conflict-free sets are extracted, and based on some comparison to the system output matrices and the matrix S, the system is able to find all stable extensions.

5. Results and analysis

We now report on the results of the competition, which evaluated the participating solvers from Section 4 using the methodology described in Section 3.[16] Tables 5–8 show the obtained rankings of all solvers per track. Each table gives the rank of the solver per semantics, the number of instances where the solver had a timeout (column "#TO"), the number of incorrectly classified instances (column "#−"), the number of correctly classified instances (column "#+"), and the total runtime for all classified instances (column "RT in ms"). The column "Significant" indicates whether the performance of a solver is significantly superior to the solver ranked right after it, according to a standard Student's T-Test with significance level 95%, cf. [18]. A "YES" indicates that the solver in the row is indeed significantly superior than the next one, a "NO" indicates that is not the case, and a "–" indicates that a significance test is not applicable—and not necessary—as the next solver correctly solved strictly less instances. Solvers are grouped by the number of correctly classified instances and ranked in each group by runtime. Therefore, note that the column on runtime is not sorted across the whole table, as solvers, which solved fewer instances within the time limit, may have a smaller total runtime on the remaining instances as solvers which solved more instances. Furthermore, solvers with identical number of solved instances and identical runtime performance are ranked equally. Table 9 shows the aggregated ranking of solvers participating in all sixteen tracks, where the column "Borda count" gives the sum of all ranks of the particular solver in all tracks.

Due to the results depicted in Table 9, the International Competition of Computational Models of Argumentation awarded the following solvers with first, second, and third place, respectively:

1. CoQuiAAS
2. ArgSemSAT
3. LabSATSolver

[16] The raw results and more detailed statistics can be found at http://argumentationcompetition.org/2015/results.html.

Table 5
Results for the problem SE ("some extension") per semantics ($N = 192$); "#TO" is the number of timeouts, "#−" is the number of incorrectly classified instances, "#+" is the number of correctly classified instances, "RT in ms" is the total runtime for all classified instances, and "Significant" indicates whether the performance of a solver is significantly better than the next solver (only applicable if both correctly solved the same number of instances).

σ	Rank	Solver	#TO	#−	#+	RT in ms	Significant
CO	1	CoQuiAAS	0	0	192	30170	YES
	2	ASGL	0	0	192	302730	NO
	3	ASPARTIX-D	0	0	192	411890	NO
	4	ConArg	0	0	192	505960	NO
	5	ArgSemSAT	0	0	192	552790	YES
	6	ArgTools	0	0	192	1627070	–
	7	LabSATSolver	2	1	189	406450	–
	8	DIAMOND	13	28	151	15452520	–
	9	Carneades	137	0	55	6300	YES
	10	Dungell	137	0	55	37400	–
PR	1	Cegartix	0	0	192	859590	YES
	2	ArgSemSAT	0	0	192	1265260	NO
	3	LabSATSolver	0	0	192	1729840	–
	4	ASPARTIX-V	1	0	191	7875480	–
	5	CoQuiAAS	2	0	190	2454510	–
	6	ASGL	12	0	180	4056110	–
	7	ConArg	15	0	177	5360400	–
	8	ASPARTIX-D	17	23	152	12379180	–
	9	ArgTools	59	0	133	4157610	–
	10	GRIS	17	103	72	28153000	–
	11	DIAMOND	2	152	38	6455890	–
	12	Carneades	192	0	0	0	NO
		Dungell	192	0	0	0	–
GR	1	CoQuiAAS	0	0	192	28860	YES
	2	Carneades	0	0	192	88490	YES
	3	LabSATSolver	0	0	192	368110	YES
	4	ArgSemSAT	0	0	192	710930	YES
	5	ArgTools	0	0	192	1654720	YES
	6	GRIS	0	0	192	4191400	–
	7	ASGL	1	0	191	307480	–
	8	ASPARTIX-D	18	13	161	7868920	–
	9	ConArg	40	0	152	2365420	–
	10	Dungell	72	0	120	213950	–
	11	DIAMOND	0	177	15	13524760	–
ST	1	ASPARTIX-D	0	0	192	275270	YES
	2	ArgSemSAT	0	0	192	739930	–
	3	LabSATSolver	1	0	191	877000	–
	4	CoQuiAAS	4	0	188	295160	–
	5	ConArg	6	0	186	7235750	–
	6	ASGL	7	1	184	3147420	–
	7	DIAMOND	14	27	151	15748650	–
	8	ArgTools	46	49	97	4461260	–
	9	ProGraph	101	0	91	11562420	–
	10	Carneades	192	0	0	0	NO
		Dungell	192	0	0	0	–
		ASSA	0	192	0	7024250	NO

Furthermore, the solver Cegartix additionally received the award "Honorable mention" as it achieved the two first places and one second place in the three tracks it participated in (SE-PR, EE-PR, DS-PR).

The statistics on timeouts and runtime performances given in Tables 5–8 show that there is a large diversity between solvers. For example, from the results of the problem SE-PR (Table 5) one can see that there are solvers without any timeout (places 1–3) and solvers not solving any instance within the time limit (place 12). Moreover, the first place (Cegartix) for this track achieved an average runtime of roughly 4,5 seconds on any instance, which is way below the timeout of 10 minutes. Similar observations can be made for the other tracks. Furthermore, many solvers performed quite differently in different tracks, compared to other solvers. For example, solver no. 16 (Carneades) achieved second place for all tracks related to grounded semantics, but only last place in all other tracks. This behavior stems from the fact that some solvers have been developed for a specific semantics (grounded semantics for Carneades), and have not been tailored towards other semantics.

All four tracks related to stable semantics have been won by ASPARTIX-D, often with great lead to the second place. For example, for the problem EE-ST it solved one more instance than the second place (ArgSemSAT) but still needed only roughly a third of the total time, cf. Table 6. ASPARTIX-D is based on reductions of abstract argumentation problems to answer set programming—see also Section 4—and therefore exploits the equivalence of stable semantics in abstract ar-

Table 6
Results for the problem EE ("enumerate extensions") per semantics ($N = 192$); "#TO" is the number of timeouts, "#−" is the number of incorrectly classified instances, "#+" is the number of correctly classified instances, "RT in ms" is the total runtime for all classified instances, and "Significant" indicates whether the performance of a solver is significantly better than the next solver (only applicable if both correctly solved the same number of instances).

σ	Rank	Solver	#TO	#−	#+	RT in ms	Significant
CO	1	ASPARTIX-D	5	1	186	1040810	–
	2	ArgSemSAT	9	0	183	4518420	–
	3	CoQuiAAS	10	0	182	1776270	NO
	4	LabSATSolver	7	3	182	2631520	–
	5	ASGL	30	0	162	7979820	–
	6	ConArg	42	0	150	2459560	–
	7	DIAMOND	22	36	134	20949590	–
	8	ArgTools	71	0	121	2612080	–
	9	Carneades	192	0	0	0	NO
		Dungell	192	0	0	0	–
PR	1	Cegartix	1	0	191	1520400	–
	2	ArgSemSAT	2	0	190	3563780	NO
	3	CoQuiAAS	2	0	190	4896610	–
	4	ASPARTIX-V	3	0	189	9926900	–
	5	LabSATSolver	3	2	187	4775950	–
	6	prefMaxSAT	27	0	165	6863850	–
	7	ASGL	29	0	163	6116050	–
	8	ASPARTIX-D	18	23	151	13381150	–
	9	ConArg	46	0	146	2650920	–
	10	ArgTools	65	0	127	3048350	–
	11	ZJU-ARG	100	17	75	1131130	-
	12	GRIS	16	104	72	28214730	–
	13	DIAMOND	9	141	42	9497860	–
	14	Carneades	192	0	0	0	NO
		Dungell	192	0	0	0	–
GR	1	CoQuiAAS	0	0	192	30390	YES
	2	Carneades	0	0	192	87000	YES
	3	LamatzSolver	0	0	192	287020	NO
	4	LabSATSolver	0	0	192	338540	YES
	5	ArgSemSAT	0	0	192	691780	NO
	6	ZJU-ARG	0	0	192	801200	YES
	7	ArgTools	0	0	192	1660070	YES
	8	GRIS	0	0	192	4184350	–
	9	ASGL	1	0	191	304810	–
	10	ASPARTIX-D	17	14	161	8572200	–
	11	ConArg	40	0	152	2353550	–
	12	Dungell	72	0	120	212280	–
	13	DIAMOND	0	177	15	13597410	–
ST	1	ASPARTIX-D	1	0	191	575620	–
	2	ArgSemSAT	2	0	190	1708400	–
	3	CoQuiAAS	4	0	188	620350	–
	4	ASGL	11	0	181	8147390	NO
	5	ConArg	11	0	181	8321000	–
	6	ArgTools	57	0	135	2371760	–
	7	LabSATSolver	1	74	117	1530240	–
	8	DIAMOND	14	104	74	19026710	–
	9	Carneades	192	0	0	0	NO
		Dungell	192	0	0	0	–
		ASSA	0	192	0	6939650	NO

gumentation to answer set semantics in a direct way. Note, that also the solver DIAMOND is based on an answer set programming reduction. In contrast to ASPARTIX-D, its approach is, however, actually a two-level reduction. In a first step, abstract argumentation problems are reduced to an equivalent formalization using *Abstract Dialectical Frameworks* [16]. In the second step, the latter is then reduced to an answer set program. This overhead is a probable cause for the lower ranking of this solver. In addition, DIAMOND shows the higher total runtime for all *incorrect* classified instances, meaning that this two-level approach has a serious impact on the performances of the solver. Notice that the second solver for higher total runtime for all incorrect classified instances is ASPARTIX-D, even if the impact on the overall performance of the solver is less significant than for DIAMOND.

Despite the exception from above, it can be seen that solvers that rely on a reduction to other established formalisms, such as SAT solving, constraint satisfaction problems, or, as mentioned, answer set programming, performed better than solvers implementing a direct algorithm for abstract argumentation. In fact, the first three places in Table 9 (CoQuiAAS,

Table 7
Results for the problem DC ("decide credulous") per semantics ($N = 576$); "#TO" is the number of timeouts, "#−" is the number of incorrectly classified instances, "#+" is the number of correctly classified instances, "RT in ms" is the total runtime for all classified instances, and "Significant" indicates whether the performance of a solver is significantly better than the next solver (only applicable if both correctly solved the same number of instances).

σ	Rank	Solver	#TO	#−	#+	RT in ms	Significant
CO	1	ArgSemSAT	0	0	576	1018060	YES
	2	ASPARTIX-D	0	0	576	2530280	–
	3	LabSATSolver	1	0	575	1705780	–
	4	CoQuiAAS	3	0	573	439500	–
	5	ASGL	8	0	568	8495780	–
	6	ConArg	21	0	555	13877500	–
	7	DIAMOND	29	5	542	48188790	–
	8	ArgTools	93	0	483	10278620	–
	9	Carneades	576	0	0	0	–
PR	1	ArgSemSAT	0	0	576	884960	YES
	2	LabSATSolver	0	0	576	1992860	–
	3	CoQuiAAS	2	0	574	412620	–
	4	ASGL	5	0	571	8841570	–
	5	DIAMOND	2	12	562	66137810	–
	6	GRIS	48	40	488	69419300	–
	7	ArgTools	93	0	483	8939600	–
	8	ASPARTIX-D	144	6	426	18318950	–
	9	Carneades	576	0	0	0	–
GR	1	CoQuiAAS	0	0	576	92580	YES
	2	Carneades	0	0	576	248430	YES
	3	LabSATSolver	0	0	576	885950	NO
	4	ASGL	0	0	576	967200	YES
	5	ArgSemSAT	0	0	576	2140360	YES
	6	ArgTools	0	0	576	5036130	YES
	7	GRIS	0	0	576	13246200	–
	8	DIAMOND	0	22	554	39350510	–
	9	ASPARTIX-D	44	1	531	23996940	–
ST	1	ASPARTIX-D	0	0	576	513390	YES
	2	ArgSemSAT	0	0	576	779820	–
	3	LabSATSolver	1	0	575	1234520	–
	4	CoQuiAAS	2	0	574	170390	–
	5	ConArg	5	0	571	4658410	–
	6	ASGL	7	0	569	4891350	–
	7	DIAMOND	30	2	544	46999150	–
	8	ASSA	0	46	530	20966220	–
	9	ArgTools	89	0	487	9716900	–
	10	ProGraph	246	18	312	37377190	–
	11	Carneades	576	0	0	0	–

ArgSemSAT, and LabSATSolver) and the honorable mention Cegartix rely on reductions to (maximum) satisfiability problems and make use of mature SAT solvers for solving argumentation problems, and all first places in all tracks use one of the three reductions mentioned above. Solvers using direct algorithms—i.e. solvers not using any other formalism than abstract argumentation—such as ArgTools and Carneades usually performed below average.

Tables 10 and 11 show the performance of the solvers wrt. the different test sets, accumulated over all tracks. For each solver, the column N in each indicates the number of instances solved for each test set (be reminded that each test set contains 24 benchmark graphs and that for DS and DC problems each benchmark graph was tested three times). For each test set generated by the generators A, B, and C the corresponding cells contain the number of incorrectly classified instances and timeouts wrt. the given total number of instances N. Table 10 gives several interesting insights into the behavior of the solvers wrt. different characteristics. For example, considering solver no. 9 (ASPARTIX-D), it can be seen that it had a hard time solving instances generated by the graph generator A (which featured large graphs with a large grounded extension), but was significantly better in solving instances of graph generators B and C (both generated smaller graphs but with a more complex attack structure). However, other solvers such as no. 3 (ArgTools) and no. 10 (ConArg) featured quite the opposite behavior, solving instances of generator A usually easier than instances of B and C. Furthermore, the average behavior of solvers on the different test sets—indicated by the summed values in the last row of the table—is quite homogeneous, where data sets 4 and 5 (generator B) are slightly harder on average. But the individual different behavior of the solvers also justifies the decision to use different graph models and challenging graph features for the competition, as otherwise some solvers would have been at an advantage.

Table 12 reports on the percentage of correctly classified instances for each solver for each of the tracks it participated in. We report with 0% when the solver participated in the track but without providing any correct answer, and we leave

Table 8
Results for the problem DS ("decide skeptical") per semantics ($N = 576$); "#TO" is the number of timeouts, "#−" is the number of incorrectly classified instances, "#+" is the number of correctly classified instances, "RT in ms" is the total runtime for all classified instances, and "Significant" indicates whether the performance of a solver is significantly better than the next solver (only applicable if both correctly solved the same number of instances).

σ	Rank	Solver	#TO	#−	#+	RT in ms	Significant
CO	1	ASGL	0	0	576	900660	NO
	2	LabSATSolver	0	0	576	1005630	YES
	3	ConArg	0	0	576	1479110	NO
	4	ArgSemSAT	0	0	576	1700390	YES
	5	ASPARTIX-D	0	0	576	3127750	YES
	6	ArgTools	0	0	576	4852460	–
	7	CoQuiAAS	1	0	575	91310	–
	8	DIAMOND	44	5	527	62109540	–
	9	Carneades	576	0	0	0	–
PR	1	ArgSemSAT	0	0	576	2005760	–
	2	Cegartix	1	0	575	1979600	YES
	3	LabSATSolver	1	0	575	5520220	–
	4	ASPARTIX-V	4	0	572	23387890	–
	5	CoQuiAAS	6	0	570	9580080	–
	6	DIAMOND	25	36	515	808113010	–
	7	GRIS	49	18	509	64506430	–
	8	ASGL	69	0	507	20655510	–
	9	ArgTools	111	0	465	6675920	–
	10	ASPARTIX-D	155	3	418	23109590	–
	11	Carneades	576	0	0	0	–
GR	1	CoQuiAAS	0	0	576	95130	YES
	2	Carneades	0	0	576	252900	YES
	3	ASGL	0	0	576	935140	NO
	4	LabSATSolver	0	0	576	1005340	YES
	5	ArgSemSAT	0	0	576	2134270	YES
	6	ArgTools	0	0	576	4852170	YES
	7	GRIS	0	0	576	11615450	–
	8	ASPARTIX-D	39	2	535	24273490	–
	9	DIAMOND	0	113	463	40100710	–
ST	1	ASPARTIX-D	0	0	576	863480	YES
	2	LabSATSolver	0	0	576	2831570	–
	3	CoQuiAAS	3	0	573	1237220	–
	4	ConArg	19	0	557	20402070	–
	5	ASGL	23	0	553	16240150	–
	6	DIAMOND	31	7	538	53686090	–
	7	ArgSemSAT	0	222	354	3009520	–
	8	ASSA	0	254	322	20818370	–
	9	ArgTools	122	172	282	10172600	–
	10	Carneades	576	0	0	0	–

Table 9
Aggregated ranking for solvers participating in all tracks.

Rank	Solver	Borda count
1	CoQuiAAS	49
2	ArgSemSAT	50
3	LabSATSolver	58
4	ASGL	82
5	ASPARTIX-D	84
6	ArgTools	119
7	Carneades	130
8	DIAMOND	134

the cell empty when the solver did not participate in that specific track. From this view on the results of the competition it emerges that systems employing SAT-solvers perform better wrt. instance classifications than those employing ASP. Only few systems have participated in tasks where they were unable to provide correct answers, i.e., Carneades, Dungell, and ASSA. It must be noted that the common point of these three systems is of being *ad-hoc* implementations of abstract argumentation frameworks. These negative results are explained by the fact that these systems take too much time to compute the extensions, and thus they time out before returning any answer, e.g., Carneades is optimized for grounded semantics only.

Table 10
Number of timeouts per solver and test set, summed up over all tracks.

No.	Solver	*N*	A			B		C		
			#1	#2	#3	#4	#5	#6	#7	#8
1	LabSATSolver	768	0	5	0	4	0	0	2	6
2	ArgSemSAT	768	0	0	0	3	0	1	3	6
3	ArgTools	768	3	9	24	221	366	0	78	105
4	Cegartix	120	0	0	0	0	1	0	0	1
5	Dungell	192	192	192	192	144	144	128	122	127
6	ZJU-ARG	48	0	0	0	23	24	14	21	18
7	ASPARTIX-V	120	0	1	6	0	0	0	0	1
8	CoQuiAAS	768	3	0	2	14	2	0	5	13
9	ASPARTIX-D	768	104	190	143	0	10	0	1	10
10	ConArg	480	0	0	0	61	118	0	40	26
11	GRIS	384	0	0	0	0	1	30	64	40
12	ASGL	768	1	0	7	30	86	0	41	38
13	LamatzSolver	24	0	0	0	0	0	0	0	0
14	ProGraph	96	29	23	27	72	72	51	34	39
15	DIAMOND	768	53	107	6	34	32	1	0	2
16	Carneades	768	576	576	576	576	576	560	554	559
17	prefMaxSAT	24	0	0	0	11	15	0	0	1
18	ASSA	192	0	0	0	0	0	0	0	0
	Sum	7824	961	1103	983	1193	1447	785	965	992

Table 11
Number of incorrectly classified instances per solver and test set, summed up over all tracks.

No.	Solver	*N*	A			B		C		
			#1	#2	#3	#4	#5	#6	#7	#8
1	LabSATSolver	768	2	1	0	7	6	24	22	18
2	ArgSemSAT	768	0	0	0	18	12	72	66	54
3	ArgTools	768	0	0	0	7	4	96	66	48
4	Cegartix	120	0	0	0	0	0	0	0	0
5	Dungell	192	0	0	0	0	0	0	0	0
6	ZJU-ARG	48	0	0	0	1	0	10	1	4
7	ASPARTIX-V	120	0	0	0	0	0	0	0	0
8	CoQuiAAS	768	0	0	0	0	0	0	0	0
9	ASPARTIX-D	768	2	7	76	0	0	0	2	0
10	ConArg	480	0	0	0	0	0	0	0	0
11	GRIS	384	0	0	0	68	67	42	34	49
12	ASGL	768	0	0	0	0	0	0	0	1
13	LamatzSolver	24	0	0	0	0	0	0	0	0
14	ProGraph	96	2	1	0	7	5	0	1	2
15	DIAMOND	768	120	146	217	65	137	113	127	119
16	Carneades	768	0	0	0	0	0	0	0	0
17	prefMaxSAT	24	0	0	0	0	0	0	0	0
18	ASSA	192	62	63	51	85	79	120	116	108
	Sum	7824	188	218	344	258	310	477	435	403

6. Lessons learned

The competition has substantially contributed to the advancement of the state-of-the-art of abstract argumentation solvers, but also made apparent where optimizations and new developments may take root. The best solvers of ICCMA'15 were based on reductions to other formalisms and thus used general multi-purpose tools. Although these solvers benefit from the maturity of e.g., current SAT solvers, the approach of reduction still adds some overhead. For one, translating a possibly huge abstract argumentation problem into an equivalent SAT instance and calling a SAT solver using a specific syntax may be time-consuming, despite the fact that the translation is polynomial from the perspective of computational complexity. Furthermore, the strategies of SAT solvers to solve SAT instances are tailored towards general or "typical" problems expressed in SAT instances. It is not apparent that SAT instances compiled from abstract argumentation problems are included in these sets of problems. To give an analogy, consider the problem of finding shortest paths in a graph. It is possible to phrase this problem as a combinatorial optimization problem and use general-purpose methods such as simulated annealing [69]. However, domain-specific algorithms such as Dijkstra's algorithm [34] clearly outperform these general-purpose methods in their domain.[17] Still, in the competition, the introduced overhead of reduction-based approaches did

[17] Note that in the given analogy, the complexity classes actually differ as *general* combinatorial optimization is not polynomial while shortest paths problems are; however, the argument is similar for reductions between problems of the same complexity.

Table 12
Percentage of correctly classified instances for each track supported by the participating solvers. Empty cells represent tracks not supported by the related solver.

No.	Solver	SE				EE				DC				DS			
		CO	PR	GR	ST	CO	PR	GR	ST	CO	PR	GR	ST	CO	PR	GR	ST
1	LabSATSolver	98%	100%	100%	99%	94%	97%	100%	60%	99%	100%	100%	99%	100%	99%	100%	100%
2	ArgSemSAT	100%	100%	100%	100%	95%	98%	100%	98%	100%	100%	100%	100%	100%	100%	100%	61%
3	ArgTools	100%	69%	100%	50%	63%	66%	100%	50%	83%	83%	100%	84%	100%	80%	100%	48%
4	Cegartix		100%				99%								99%		
5	Dungell	28%	0%	62%	0%	0%	0%	62%	0%								
6	ZJU-ARG						39%	100%									
7	ASPARTIX-V		99%				98%								99%		
8	CoQuiAAS	100%	98%	100%	97%	94%	98%	100%	97%	99%	99%	100%	99%	99%	98%	100%	99%
9	ASPARTIX-D	100%	79%	83%	100%	96%	78%	83%	99%	100%	73%	92%	100%	100%	72%	92%	100%
10	ConArg	100%	92%	79%	96%	78%	76%	79%	94%	96%			99%	100%			96%
11	GRIS		37%	100%			37%	100%			84%	100%			88%	100%	
12	ASGL	100%	93%	99%	95%	84%	84%	99%	94%	98%	99%	100%	98%	100%	88%	100%	96%
13	LamatzSolver							100%									
14	ProGraph				47%								54%				
15	DIAMOND	78%	19%	7%	78%	69%	21%	7%	38%	94%	97%	96%	94%	91%	89%	80%	93%
16	Carneades	9%	0%	100%	0%	0%	0%	100%	0%	0%	0%	100%	0%	0%	0%	100%	0%
17	prefMaxSAT						95%										
18	ASSA				0%				0%				92%				55%

not significantly outweigh the maturity of the utilized tools. This fact indicates that focused research and development of domain-specific approaches to abstract argumentation may outperform reduction-based approaches in the future. Whilst on the one side, we expect the next generation of solvers to outperform the general purpose SAT-solvers exploited by the systems participating in the competition, on the other side, we should question about the *actual necessity* of doing so. Applications of such abstract argumentation solvers to concrete usage scenarios are hard to find, and recent results in the argument mining community [19] show that existing graphs extracted from real natural language argumentation interactions, e.g., online dialogues in blogs, do not (almost) present cycles among the arguments. Moreover, such dialogues end up with pretty small graphs (e.g., 40 nodes for an online debate about a specific topic). These observations seem to suggest that the actual need of the community is not to outperform SAT-based solvers. However, it must be noticed that the results provided by the argument mining community are still preliminary, and there is actually a potential in mining for huge argumentation graphs reporting the views of hundreds of users about a certain topic widely discussed on the Web (e.g., including the opinions reported on blogs, social networks, online debate platforms). In conclusion, even if the fact of outperforming existing solvers does not answer a present need in the community, it will in the near future in combination with argument mining techniques.

Concerning the semantics, being ICCMA'15 the first edition of such a competition, it was decided to focus on the four standard semantics [35]. Given the results, this appears to be a reasonable choice, as many of the solvers were unable to tackle the whole range of tracks. It would have been useless to provide even more semantics, as it would affected only very few solvers. The next ICCMA competition scheduled for 2017 will consider also ideal, semi-stable and stage semantics, in addition to the four standard semantics.

Concerning the input graphs, we believe that we covered a sufficient range of graph structures in order to avoid penalizing some implementations over others. In general, we conclude that one of the main insights of the competition is that there is still great potential for developing new sophisticated algorithmic approaches to abstract argumentation problems.

Finally, the ASPARTIX format has emerged as the standard format for the input data, and almost all participants have adapted their solvers to accept such an input format. This is in line with other well-known competitions like for instance the SAT solver competition, where the standard input format is the DIMACS CNF format.

7. Conclusions

This paper gave an overview on the First International Competition of Computational Models of Argumentation (ICCMA'15). We described the computational tasks of the competition, its technical setups, and presented the participants. Furthermore, we reported on its results and provided some analysis and interpretation. Being the first instance of ICCMA, the organizers were very satisfied with the engagement of the community and its 18 submitted solvers. The competition provided a common background to compare different solvers developed in the last years in the computational models of argumentation community with the adoption, on the one side of novel algorithms, and on the other side, of SAT- and ASP-based standard solutions, to address heterogeneous goals. Thanks to the results the competition publicly made available, informed decisions about the choice of the right solver to adopt with respect to the computational task to be performed are now possible. The results show unsurprisingly that SAT-based and ASP-based solvers outperform ad-hoc algorithms, but there is still great potential for developing new sophisticated algorithmic approaches to abstract argumentation problems.

The competition has also highlighted new needs that the ICCMA steering committee is evaluating in order to propose new tracks for the upcoming editions of the competition. First of all, a track about structured argumentation frameworks has been envisioned, and it is currently under discussion. The main issue is that there is no structured formalism with a sufficient number of competing solvers, and addressing a comparison of systems implementing different formalisms is extremely laborious. The lack of existing benchmarks for such a task is another issue in this direction. Second, a track about natural language argumentation is envisaged as well. The idea is to include an argument mining tack where tasks such as arguments detection and relations prediction are proposed to the systems. Also in this case, the main issue is the lack of a common annotation schema, and as a consequence, of a common annotated benchmark to compare all the existing approaches.

The competition was the first in an upcoming series of competitions, the next instance is planned for 2017.[18]

Acknowledgements

We would like to thank all participants of ICCMA'15 for submitting solvers and actively engaging in the effort of advancing implementations for computational models of argumentation. We also thank the ICCMA steering committee for providing support and shaping the competition.

Appendix A. Algorithms of benchmark generators

We now provide pseudo code for the algorithms used to generate the benchmark graphs of the competition. Algorithm 1 shows the code for `GroundedGenerator` (Generator A), Algorithm 2 shows the code for `StableGenerator` (Generator B), and Algorithm 3 shows the code for `SccGenerator` (Generator C).

[18] http://argumentationcompetition.org/2017/index.html.

Algorithm 1 A `GroundedGenerator`.

Require: maxA (maximal number of arguments)
Require: p (probability of attack)
A = random integer in $\{1, \ldots, \text{maxA}\}$
G = (V,E), V = $\{a_1, \ldots, a_A\}$, E = $\emptyset$
unconnected = $\{a_1, \ldots, a_A\}$
for all $i = 1, \ldots, A$ **do**
  for all $j = 1, \ldots, i-1$ **do**
    if random number in $[0, 1]$ is smaller p **then**
      E = E $\cup$ (a_i, a_j)
      unconnected = unconnected $\setminus$ $\{a_i\}$
for all b $\in$ unconnected **do**
  k = random integer in $\{1, \ldots, \text{A}\}$
  if coin flip shows heads **then**
    E = E $\cup$ $\{(b, a_k)\}$
  else
    E = E $\cup$ $\{(a_k, b)\}$
return G

Algorithm 2 B: `StableGenerator`.

Require: maxA (maximal number of arguments)
Require: minNumExtensions (approx. minimal number of stable extensions)
Require: maxNumExtensions (approx. maximal number of stable extensions)
Require: minSizeOfExtensions (approx. minimal size of a stable extension)
Require: maxSizeOfExtensions (approx. maximal size of a stable extension)
Require: minSizeOfGrounded (approx. minimal size of grounded extension)
Require: maxSizeOfGrounded (approx. maximal size of grounded extension)
Require: p (probability of attack)
A = random integer in {1, ..., maxA}
X = random integer in {minNumExtensions,...,maxNumExtensions}
S = random integer in {minSizeOfExtensions,...,maxSizeOfExtensions}
R = random integer in {minSizeOfGrounded,...,maxSizeOfGrounded}
G = (V,E), V = $\{a_1, \ldots, a_A\}$, E = $\emptyset$
grounded = $\{a_1, \ldots, a_R\}$
for all $i = 1, \ldots$,R **do**
  for all $k = 0, \ldots, i-1$ **do**
    if random number in $[0, 1]$ is smaller p **then**
      E = E $\cup$ (a_i, a_k)
for all $j = 1, \ldots$, X **do**
  Let M be a random set of S arguments in V
  for all $i = R+1, \ldots,$ A **do**
    if $a_i \notin M$ **then**
      Let a_k be a random argument in M
      E = E $\cup$ (a_k, a_i)
return G

Appendix B. Statistics on benchmark graphs

Tables B.13–B.16 give a detailed overview on the structure and properties of the benchmark graphs considered for ICCMA'15. In particular, the table show for each graph of each test set the number of arguments and number of attacks, the average in-degree (= number of attackers), the (global) *clustering coefficient* CC, the number of strongly connected components (SCCs), its *density*, the number of complete, preferred, and stable extensions, the size of the grounded extension, and the average size of its complete, preferred, and stable extensions. For an argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$, the global clustering coefficient [73] is defined as (let $\mathcal{A} \rightleftharpoons \mathcal{B}$ denote "$\mathcal{A} \rightarrow \mathcal{B}$ *or* $\mathcal{B} \rightarrow \mathcal{A}$" for any $\mathcal{A}, \mathcal{B} \in \mathsf{Arg}$):

$$CC(\mathsf{AF}) = \frac{|\{\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\} \subseteq \mathsf{Arg} \mid \mathcal{A}_1 \rightleftharpoons \mathcal{A}_2, \mathcal{A}_2 \rightleftharpoons \mathcal{A}_3, \mathcal{A}_3 \rightleftharpoons \mathcal{A}_1\}|}{|\{\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\} \subseteq \mathsf{Arg} \mid \mathcal{A}_1 \rightleftharpoons \mathcal{A}_2, \mathcal{A}_2 \rightleftharpoons \mathcal{A}_3\}|}$$

In other words, $CC(\mathsf{AF})$ is the ratio of the number of undirected triangles in AF and the number of connected triples of arguments and is thus a value in $[0, 1]$. Large values indicate a high clustering of the arguments. For an argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$, its *density* is defined as

$$D(\mathsf{AF}) = \frac{|\rightarrow|}{|\mathsf{Arg}|(|\mathsf{Arg}| - 1)}$$

Algorithm 3 `SccGenerator`.

Require: maxA (maximal number of arguments)
Require: maxNumSccs (approx. maximal number of SCCs)
Require: p_{inner} (probability of attack in SCCs)
Require: p_{outer} (probability of attack between SCCs)
Require: p_{scc} (probability to connect two SCCs)

```
A = random integer in {1, ..., maxA}
N = random integer in {1, ..., maxNumSccs}
G = (V,E), V = {a_1, ..., a_A}, E = ∅
C[1] = ∅, ..., C[N] = ∅
for all i = 1, ..., A do
    k = random integer in {1, ..., N}
    C[k] = C[k] ∪ {a_i}
for all i = 1, ..., N do
    for all arg1 ∈ C[i] do
        for all arg2 ∈ C[i] do
            if random number in [0, 1] is smaller than p_inner then
                E = E ∪ {(arg1, arg2)}
for all i = 1, ..., N-1 do
    for all j = i + 1, ..., N do
        if random number in [0, 1] is smaller than p_scc then
            for all arg1 ∈ C[i] do
                for all arg2 ∈ C[j] do
                    if random number in [0, 1] is smaller than p_outer then
                        E = E ∪ {(arg1,arg2)}
return G
```

Table B.13
Statistics on the benchmark graphs of test sets 1 (graphs 1–24) and 2 (graphs 25–48); #Arg = number of arguments, #Att = number of attacks, in = average in-degree, cc = clustering coefficient, dens. = density, #sccs = number of strongly connected components, #CO = number of complete extensions, #PR = number of preferred extensions, #ST = number of stable extensions, |GR| = size of grounded extension, |CO| = average size of complete extensions, |PR| = average size of preferred extensions, $|ST|$ = average size of stable extensions.

ID	#Arg	#Att	in	cc	dens.	#sccs	#CO	#PR	#ST	|GR|	|CO|	|PR|	|ST|
1	1909	3668	19.21	0.01	0.01	245	1	1	1	183	183.0	183.0	183.0
2	1880	3555	18.91	0.01	0.01	302	1	1	1	184	184.0	184.0	184.0
3	2223	4956	22.3	0.01	0.01	350	1	1	1	189	189.0	189.0	189.0
4	2511	6277	25.0	0.01	0.01	472	1	1	1	193	193.0	193.0	193.0
5	4166	17289	41.5	0.01	0.01	658	1	1	1	217	217.0	217.0	217.0
6	2033	4143	0.38	0.01	0.01	285	1	1	1	183	183.0	183.0	183.0
7	2636	6886	26.12	0.01	0.01	393	1	1	1	199	199.0	199.0	199.0
8	2696	7280	27.0	0.01	0.01	739	1	1	1	199	199.0	199.0	199.0
9	2743	7508	27.37	0.01	0.01	246	1	1	1	199	199.0	199.0	199.0
10	2943	8663	29.44	0.01	0.01	389	1	1	1	200	200.0	200.0	200.0
11	2631	6893	26.2	0.01	0.01	310	1	1	1	198	198.0	198.0	198.0
12	2846	8153	28.65	0.01	0.01	340	1	1	1	197	197.0	197.0	197.0
13	1224	1484	12.13	0.01	0.01	254	1	1	1	154	154.0	154.0	154.0
14	1315	1730	13.16	0.01	0.01	331	1	1	1	160	160.0	160.0	160.0
15	1278	1633	12.78	0.01	0.01	231	1	1	1	166	166.0	166.0	166.0
16	1624	2659	16.37	0.01	0.01	256	1	1	1	183	183.0	183.0	183.0
17	1866	3504	18.78	0.01	0.01	465	1	1	1	186	186.0	186.0	186.0
18	3144	9886	31.45	0.01	0.01	577	1	1	1	208	208.0	208.0	208.0
19	2797	7855	28.08	0.01	0.01	190	1	1	1	200	200.0	200.0	200.0
20	3740	14045	37.56	0.01	0.01	537	1	1	1	215	215.0	215.0	215.0
21	3600	12957	35.99	0.01	0.01	239	1	1	1	216	216.0	216.0	216.0
22	3589	12931	36.03	0.01	0.01	317	1	1	1	208	208.0	208.0	208.0
23	3737	14018	37.51	0.01	0.01	386	1	1	1	214	214.0	214.0	214.0
24	3713	13856	37.32	0.01	0.01	355	1	1	1	213	213.0	213.0	213.0
25	2219	4936	22.25	0.01	0.01	497	1	1	1	187	187.0	187.0	187.0
26	2838	8042	28.34	0.01	0.01	390	1	1	1	196	196.0	196.0	196.0
27	3896	15163	38.92	0.01	0.01	568	1	1	1	216	216.0	216.0	216.0
28	4459	19886	44.6	0.01	0.01	387	1	1	1	219	219.0	219.0	219.0
29	4827	23222	48.11	0.01	0.01	560	1	1	1	231	231.0	231.0	231.0
30	4685	21863	46.67	0.01	0.01	548	1	1	1	224	224.0	224.0	224.0

(*continued on next page*)

Table B.13 (*continued*)

ID	#Arg	#Att	in	cc	dens.	#sccs	#CO	#PR	#ST	|GR|	|CO|	|PR|	|ST|
31	2450	6010	24.53	0.01	0.01	343	1	1	1	191	191.0	191.0	191.0
32	3005	9059	30.15	0.01	0.01	276	1	1	1	208	208.0	208.0	208.0
33	2684	7233	26.95	0.01	0.01	424	1	1	1	197	197.0	197.0	197.0
34	4959	24612	49.63	0.01	0.01	729	1	1	1	228	228.0	228.0	228.0
35	3478	12141	34.91	0.01	0.01	657	1	1	1	206	206.0	206.0	206.0
36	5900	34720	58.85	0.01	0.01	438	1	1	1	237	237.0	237.0	237.0
37	4470	20063	44.88	0.01	0.01	375	1	1	1	227	227.0	227.0	227.0
38	6017	36208	60.18	0.01	0.01	572	1	1	1	240	240.0	240.0	240.0
39	6546	42923	65.57	0.01	0.01	275	1	1	1	242	242.0	242.0	242.0
40	3955	15584	39.41	0.01	0.01	228	1	1	1	229	229.0	229.0	229.0
41	1945	3797	19.53	0.01	0.01	302	1	1	1	185	185.0	185.0	185.0
42	5257	27616	52.53	0.01	0.01	238	1	1	1	231	231.0	231.0	231.0
43	1937	3770	19.47	0.01	0.01	245	1	1	1	182	182.0	182.0	182.0
44	3994	15976	40.0	0.01	0.01	190	1	1	1	222	222.0	222.0	222.0
45	1152	1346	11.69	0.01	0.01	260	1	1	1	152	152.0	152.0	152.0
46	3954	15641	39.56	0.01	0.01	588	1	1	1	217	217.0	217.0	217.0
47	3168	10029	31.66	0.01	0.01	247	1	1	1	206	206.0	206.0	206.0
48	2085	4348	20.86	0.01	0.01	263	1	1	1	193	193.0	193.0	193.0

Table B.14
Statistics on the benchmark graphs of test sets 3 (graphs 49–72) and 4 (graphs 73–96); #Arg = number of arguments, #Att = number of attacks, in = average in-degree, cc = clustering coefficient, dens. = density, #sccs = number of strongly connected components, #CO = number of complete extensions, #PR = number of preferred extensions, #ST = number of stable extensions, |GR| = size of grounded extension, |CO| = average size of complete extensions, |PR| = average size of preferred extensions, |ST| = average size of stable extensions.

ID	#Arg	#Att	in	cc	dens.	#sccs	#CO	#PR	#ST	|GR|	|CO|	|PR|	|ST|
49	7191	51655	71.83	0.01	0.01	305	1	1	1	245	245.0	245.0	245.0
50	6142	37683	61.35	0.01	0.01	394	1	1	1	238	238.0	238.0	238.0
51	6961	48482	69.65	0.01	0.01	469	1	1	1	244	244.0	244.0	244.0
52	4872	23657	48.56	0.01	0.01	341	1	1	1	239	239.0	239.0	239.0
53	7861	61863	78.7	0.01	0.01	367	1	1	1	255	255.0	255.0	255.0
54	6977	48531	69.56	0.01	0.01	443	1	1	1	251	251.0	251.0	251.0
55	7242	52543	72.55	0.01	0.01	410	1	1	1	245	245.0	245.0	245.0
56	8402	70560	83.98	0.01	0.01	463	1	1	1	257	257.0	257.0	257.0
57	7521	56661	75.34	0.01	0.01	573	1	1	1	251	251.0	251.0	251.0
58	7833	61518	78.54	0.01	0.01	754	1	1	1	249	249.0	249.0	249.0
59	8366	69910	83.57	0.01	0.01	503	1	1	1	258	258.0	258.0	258.0
60	7318	53649	73.31	0.01	0.01	881	1	1	1	243	243.0	243.0	243.0
61	5593	31399	56.14	0.01	0.01	1267	1	1	1	234	234.0	234.0	234.0
62	7746	60078	77.56	0.01	0.01	1722	1	1	1	253	253.0	253.0	253.0
63	5774	33330	57.73	0.01	0.01	423	1	1	1	231	231.0	231.0	231.0
64	8393	70558	84.07	0.01	0.01	347	1	1	1	263	263.0	263.0	263.0
65	9473	89713	94.7	0.01	0.01	436	1	1	1	262	262.0	262.0	262.0
66	6499	42252	65.01	0.01	0.01	290	1	1	1	246	246.0	246.0	246.0
67	9360	87622	93.61	0.01	0.01	519	1	1	1	261	261.0	261.0	261.0
68	6958	48439	69.62	0.01	0.01	365	1	1	1	252	252.0	252.0	252.0
69	6518	42442	65.12	0.01	0.01	433	1	1	1	238	238.0	238.0	238.0
70	8548	73157	85.58	0.01	0.01	592	1	1	1	258	258.0	258.0	258.0
71	6219	38707	62.24	0.01	0.01	562	1	1	1	236	236.0	236.0	236.0
72	5013	25121	50.11	0.01	0.01	370	1	1	1	238	238.0	238.0	238.0
73	141	38	2.74	0.02	0.02	105	360	40	32	19	34.71	43.2	44.06
74	299	424	14.21	0.03	0.05	6	4	3	3	4	29.5	38.0	38.0
75	289	416	14.4	0.03	0.05	6	3	2	2	3	27.33	39.5	39.5
76	277	399	14.43	0.04	0.05	6	1	1	0	3	3.0	3.0	–
77	202	50	2.51	0.02	0.01	91	159	12	12	15	49.12	68.83	68.83
78	286	280	9.81	0.02	0.03	4	1	1	0	3	3.0	3.0	–
79	280	367	13.11	0.03	0.05	24	16	8	8	12	33.88	38.13	38.13
80	276	261	9.49	0.02	0.03	6	3	2	2	3	30.33	44.0	44.0
81	291	414	14.23	0.03	0.05	6	2	1	1	3	20.0	37.0	37.0
82	232	333	14.38	0.04	0.06	6	6	4	4	3	25.17	29.75	29.75
83	258	250	9.72	0.03	0.04	6	9	6	5	4	37.11	41.5	41.8
84	197	271	13.79	0.05	0.07	6	13	10	10	3	23.38	25.5	25.5
85	280	399	14.28	0.03	0.05	6	14	7	7	3	33.71	36.43	36.43
86	300	287	9.57	0.02	0.03	6	1	1	0	3	3.0	3.0	–
87	193	46	2.42	0.01	0.01	79	70	11	0	16	40.31	52.0	–

Table B.14 (*continued*)

ID	#Arg	#Att	in	cc	dens.	#sccs	#CO	#PR	#ST	|GR|	|CO|	|PR|	|ST|
88	262	376	14.37	0.04	0.06	6	1	1	0	5	5.0	5.0	–
89	292	284	9.73	0.02	0.03	6	5	3	3	3	40.4	50.0	50.0
90	246	354	14.39	0.04	0.06	6	1	1	0	3	3.0	3.0	–
91	259	246	9.53	0.02	0.04	6	11	7	7	3	37.27	41.43	41.43
92	300	290	9.69	0.02	0.03	6	15	9	9	3	45.73	49.44	49.44
93	162	218	13.48	0.06	0.08	6	3	2	2	3	15.33	21.5	21.5
94	290	416	14.36	0.03	0.05	6	5	4	4	2	29.8	36.75	36.75
95	293	427	14.58	0.03	0.05	5	2	1	1	3	21.5	40.0	40.0
96	287	412	14.36	0.03	0.05	6	4	2	2	3	25.0	33.0	33.0

Table B.15
Statistics on the benchmark graphs of test sets 5 (graphs 97–120) and 6 (graphs 121–144); #Arg = number of arguments, #Att = number of attacks, in = average in-degree, cc = clustering coefficient, dens. = density, #sccs = number of strongly connected components, #CO = number of complete extensions, #PR = number of preferred extensions, #ST = number of stable extensions, |GR| = size of grounded extension, |CO| = average size of complete extensions, |PR| = average size of preferred extensions, |ST| = average size of stable extensions.

ID	#Arg	#Att	in	cc	dens.	#sccs	#CO	#PR	#ST	|GR|	|CO|	|PR|	|ST|
97	400	379	9.5	0.02	0.02	31	3	2	1	8	44.67	63.0	63.0
98	400	383	9.59	0.02	0.02	20	2	1	1	8	36.5	65.0	65.0
99	400	539	13.49	0.02	0.03	41	4	2	2	9	39.5	51.5	51.5
100	400	541	13.55	0.02	0.03	32	7	5	5	11	46.29	52.4	52.4
101	400	541	13.53	0.02	0.03	19	7	5	5	10	48.43	55.0	55.0
102	400	383	9.59	0.02	0.02	29	1	1	0	7	7.0	7.0	–
103	400	562	14.05	0.02	0.04	21	1	1	0	9	9.0	9.0	–
104	400	563	14.09	0.02	0.04	23	1	1	0	7	7.0	7.0	–
105	400	545	13.63	0.02	0.03	41	2	1	1	12	34.0	56.0	56.0
106	400	556	13.92	0.02	0.03	31	1	1	0	8	8.0	8.0	–
107	400	550	13.77	0.02	0.03	32	7	4	2	10	44.86	51.0	52.0
108	400	374	9.37	0.02	0.02	40	8	6	6	8	58.75	66.67	66.67
109	400	378	9.45	0.02	0.02	22	3	2	2	8	49.33	70.0	70.0
110	400	375	9.39	0.02	0.02	20	6	4	4	8	58.5	69.0	69.0
111	400	559	13.98	0.02	0.04	21	3	2	2	9	38.67	53.5	53.5
112	400	377	9.45	0.02	0.02	32	6	3	3	8	52.17	63.0	63.0
113	400	546	13.65	0.02	0.03	40	5	4	3	9	42.8	51.25	51.67
114	400	384	9.6	0.02	0.02	21	3	2	2	6	43.0	61.5	61.5
115	400	374	9.37	0.02	0.02	21	6	4	4	5	55.67	66.75	66.75
116	400	374	9.37	0.02	0.02	29	2	1	1	8	36.0	64.0	64.0
117	400	575	4.39	0.03	0.04	21	3	2	2	8	35.33	49.0	49.0
118	400	376	9.42	0.02	0.02	31	5	4	3	9	52.6	63.5	65.0
119	400	379	9.48	0.02	0.02	20	3	2	2	10	47.33	66.0	66.0
120	400	558	13.97	0.02	0.04	30	5	3	3	7	43.0	52.0	52.0
121	283	214	7.59	0.09	0.03	61	288	2	0	4	8.83	13.0	–
122	319	250	7.86	0.07	0.02	75	152	3	0	1	5.84	10.0	–
123	334	329	9.87	0.09	0.03	37	162	7	0	0	4.28	7.0	–
124	189	98	5.2	0.05	0.03	100	800	1	0	3	13.86	24.0	–
125	397	401	10.11	0.05	0.03	76	14	1	0	2	4.64	7.0	–
126	242	143	5.95	0.06	0.02	98	496	4	0	4	13.5	19.0	–
127	260	187	7.22	0.08	0.03	52	28	6	0	0	3.43	4.83	–
128	217	166	7.66	0.1	0.04	52	48	4	0	0	5.0	8.5	–
129	308	984	31.98	0.23	0.1	6	1	1	0	0	0.0	0.0	–
130	332	1451	43.71	0.42	0.13	4	1	1	0	0	0.0	0.0	–
131	318	335	10.56	0.1	0.03	28	4	1	0	0	2.0	4.0	–
132	379	935	24.67	0.3	0.07	9	4	2	0	0	3.25	4.5	–
133	330	362	10.99	0.09	0.03	28	2	1	0	0	1.0	2.0	–
134	332	443	13.36	0.14	0.04	20	51	3	0	0	6.69	10.67	–
135	306	223	7.31	0.06	0.02	89	4	1	0	3	4.0	5.0	–
136	379	379	10.02	0.06	0.03	68	768	8	0	0	6.9	13.0	–
137	360	533	14.81	0.12	0.04	22	4	2	0	0	2.5	3.5	–
138	285	301	10.56	0.12	0.04	21	180	4	0	0	6.13	11.5	–
139	360	529	14.72	0.12	0.04	21	8	2	0	0	3.0	5.5	–
140	185	87	4.73	0.05	0.03	93	200	2	0	11	20.27	27.0	–
141	291	215	7.39	0.06	0.03	89	30	1	0	2	7.33	12.0	–
142	374	499	13.36	0.13	0.04	23	10	1	0	0	3.9	8.0	–
143	381	558	14.65	0.11	0.04	23	2	1	0	0	1.0	2.0	–
144	366	456	12.48	0.11	0.03	28	28	3	0	0	4.43	7.33	–

Table B.16
Statistics on the benchmark graphs of test sets 7 (graphs 145–168) and 8 (graphs 169–192); #Arg = number of arguments, #Att = number of attacks, in = average in-degree, cc = clustering coefficient, dens. = density, #sccs = number of strongly connected components, #CO = number of complete extensions, #PR = number of preferred extensions, #ST = number of stable extensions, |GR| = size of grounded extension, $|CO|$ = average size of complete extensions, |PR| = average size of preferred extensions, |ST| = average size of stable extensions.

ID	#Arg	#Att	in	cc	dens.	#sccs	#CO	#PR	#ST	|GR|	|CO|	|PR|	|ST|
145	826	5044	61.07	0.27	0.07	8	4	1	0	0	4.5	9.0	–
146	777	1179	15.18	0.28	0.02	18	9	4	0	0	5.0	7.5	–
147	754	1711	22.7	0.08	0.03	34	2	1	0	0	0.5	1.0	–
148	311	284	9.14	0.06	0.03	65	3916	5	0	0	10.27	18.0	–
149	780	2567	32.92	0.12	0.04	20	2	1	0	0	1.5	3.0	–
150	500	1211	24.23	0.2	0.05	14	1	1	0	0	0.0	0.0	–
151	315	2735	86.83	0.35	0.28	2	4	2	2	0	5.5	9.0	9.0
152	393	407	10.36	0.06	0.03	63	44	2	0	2	7.0	10.5	–
153	499	1180	23.66	0.15	0.05	16	18	4	0	0	7.17	11.5	–
154	442	563	12.75	0.08	0.03	39	6	2	0	0	1.5	2.5	–
155	243	2936	120.85	0.5	0.5	1	1	1	0	0	0.0	0.0	–
156	455	2839	62.41	0.36	0.14	4	3	1	0	0	4.0	8.0	–
157	636	1186	18.66	0.1	0.03	34	20	3	0	0	6.15	9.33	–
158	417	467	11.2	0.06	0.03	50	1011	3	0	0	10.84	19.0	–
159	679	2609	38.43	0.19	0.06	12	4	1	0	0	4.0	8.0	–
160	649	1410	21.74	0.1	0.03	27	1	1	0	0	0.0	0.0	–
161	458	594	12.98	0.09	0.03	39	1297	8	0	0	10.8	18.5	–
162	652	2534	38.88	0.15	0.06	13	2	1	0	0	2.0	4.0	–
163	490	6622	135.15	0.35	0.28	2	1	1	0	0	0.0	0.0	–
164	351	6119	174.36	0.5	0.5	1	1	1	0	0	0.0	0.0	–
165	696	73	1.06	0.0	0.0	686	27	8	8	349	356.67	360.5	360.5
166	733	1904	25.99	0.12	0.04	24	9	2	0	0	5.0	9.0	–
167	703	1670	23.76	0.1	0.03	28	2	1	0	0	1.0	2.0	–
168	757	2903	38.36	0.17	0.05	14	1	1	0	0	0.0	0.0	–
169	461	136	2.95	0.01	0.01	395	324	24	24	123	130.5	134.67	134.67
170	457	112	2.46	0.02	0.01	345	222	20	0	119	137.34	142.9	–
171	755	1538	20.38	0.06	0.03	46	12	3	0	0	4.83	7.33	–
172	671	278	4.15	0.01	0.01	617	18	4	4	136	140.5	143.0	143.0
173	795	1707	21.48	0.07	0.03	42	7	2	0	0	3.43	5.5	–
174	769	1484	19.3	0.06	0.03	48	190	4	0	0	9.17	14.0	–
175	778	1565	20.12	0.06	0.03	49	6	1	0	0	1.5	3.0	–
176	595	277	4.66	0.02	0.01	301	5274	192	192	34	125.65	152.5	152.5
177	507	410	8.09	0.01	0.02	416	90	16	16	48	58.97	66.0	66.0
178	837	1925	23.01	0.08	0.03	38	3	1	0	0	1.33	3.0	–
179	374	152	4.08	0.02	0.01	290	495	44	44	36	64.88	83.95	83.95
180	868	2094	24.13	0.07	0.03	42	6	2	0	0	3.5	6.0	–
181	853	3393	39.79	0.12	0.05	18	1	1	0	0	0.0	0.0	–
182	847	1793	21.17	0.07	0.03	47	1	1	0	0	0.0	0.0	–
183	940	3027	32.21	0.11	0.03	26	1	1	0	0	0.0	0.0	–
184	960	2297	23.93	0.08	0.02	42	2	1	0	0	1.0	2.0	–
185	996	3264	32.78	0.12	0.03	26	1	1	0	0	0.0	0.0	–
186	481	11551	240.16	0.5	0.5	1	1	1	0	0	0.0	0.0	–
187	935	10939	116.99	0.23	0.13	5	1	1	0	0	0.0	0.0	–
188	939	2711	28.87	0.1	0.03	31	2	1	0	0	1.0	2.0	–
189	489	11955	244.49	0.5	0.5	1	1	1	0	0	0.0	0.0	–
190	992	18650	188.01	0.32	0.19	3	2	1	0	0	3.0	6.0	–
191	964	440	4.57	0.01	0.0	852	54	8	8	152	162.33	171.5	171.5
192	583	17018	291.92	0.5	0.5	1	1	1	0	0	0.0	0.0	–

References

[1] O. Arieli, M.W.A. Caminada, A QBF-based formalization of abstract argumentation semantics, J. Appl. Log. 11 (2) (2013) 229–252.
[2] G. Audemard, J. Lagniez, L. Simon, Improving glucose for incremental SAT solving with assumptions: application to MUS extraction, in: Theory and Applications of Satisfiability Testing, SAT 2013, 16th International Conference Proceedings, Helsinki, Finland, July 8–12, 2013, in: Lect. Notes Comput. Sci., vol. 7962, Springer, 2013, pp. 309–317.
[3] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[4] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowl. Eng. Rev. 26 (4) (2011) 365–410.
[5] P. Baroni, M. Giacomin, G. Guida, SCC-recursiveness: a general schema for argumentation semantics, Artif. Intell. 168 (1–2) (2005) 162–210.
[6] T.J.M. Bench-Capon, P.E. Dunne, Argumentation in artificial intelligence, Artif. Intell. 171 (2007) 619–641.
[7] P. Besnard, S. Doutre, Checking the acceptability of a set of arguments, in: Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR 2004, Whistler, Canada, June 6–8, 2004, 2004, pp. 59–64.
[8] P. Besnard, A. Hunter, Elements of Argumentation, The MIT Press, 2008.
[9] A. Biere, Lingeling essentials, a tutorial on design and implementation aspects of the SAT solver lingeling, in: Fifth Pragmatics of SAT Workshop, Workshop of the SAT 2014 Conference, Part of FLoC 2014 During the Vienna Summer of Logic, POS-14, July 13, 2014, Vienna, Austria, in: EPiC Ser. Comput., vol. 27, EasyChair, 2014, p. 88.

[10] S. Bistarelli, F. Rossi, F. Santini, A first comparison of abstract argumentation systems: a computational perspective, in: D. Cantone, M.N. Asmundo (Eds.), Proceedings of the 28th Italian Conference on Computational Logic, 2013, pp. 241–245.
[11] S. Bistarelli, F. Rossi, F. Santini, A first comparison of abstract argumentation reasoning-tools, in: ECAI 2014 – 21st European Conference on Artificial Intelligence, Including Prestigious Applications of Intelligent Systems, PAIS 2014, 18–22 August 2014, Prague, Czech Republic, 2014, pp. 969–970.
[12] S. Bistarelli, F. Rossi, F. Santini, Benchmarking hard problems in random abstract AFs: the stable semantics, in: Computational Models of Argument – Proceedings of COMMA 2014, 2014, pp. 153–160.
[13] S. Bistarelli, F. Rossi, F. Santini, ConArg2: a constraint-based tool for abstract argumentation, in: [92], pp. 33–36, arXiv:1510.05373, 2015.
[14] S. Bistarelli, F. Rossi, F. Santini, A comparative test on the enumeration of extensions in abstract argumentation, Fund. Inform. 140 (3–4) (2015) 263–278.
[15] S. Bistarelli, F. Rossi, F. Santini, ConArg: a tool for classical and weighted argumentation, in: Computational Models of Argument: Proceedings of COMMA 2016, vol. 287, 2016, p. 463.
[16] G. Brewka, S. Ellmauthaler, H. Strass, J.P. Wallner, S. Woltran, Abstract dialectical frameworks revisited, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI'13, 2013.
[17] F. Brons, LabSAT-solver: utilizing Caminada's labelling approach as a Boolean satisfiability problem, in: [92], pp. 1–3, arXiv:1510.05373, 2015.
[18] M.G. Bulmer, Principles of Statistics, Dover Publications, 1979.
[19] E. Cabrio, S. Villata, Node: a benchmark of natural language arguments, in: Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014, in: Front. Artif. Intell. Appl., vol. 266, IOS Press, 2015, pp. 449–450.
[20] E. Cabrio, S. Villata, F. Gandon, A support framework for argumentative discussions management in the web, in: Proceedings of the 10th Extended Semantic Web Conference, ESWC'13, Springer-Verlag, 2013, pp. 412–426.
[21] M. Caminada, On the issue of reinstatement in argumentation, in: Proceedings of the 10th European Conference on Logics in Artificial Intelligence, JELIA'06, 2006, pp. 111–123.
[22] M. Caminada, Semi-stable semantics, in: P. Dunne, T. Bench-Capon (Eds.), Proceedings of the First International Conference on Computational Models of Argument, COMMA'06, IOS Press, 2006, pp. 121–130.
[23] F. Cerutti, P.E. Dunne, M. Giacomin, M. Vallati, Computing preferred extensions in abstract argumentation: a SAT-based approach, in: Theory and Applications of Formal Argumentation – Second International Workshop, TAFA 2013, Beijing, China, August 3–5, 2013, in: Lect. Notes Comput. Sci., vol. 8306, Springer, 2013, pp. 176–193, Revised Selected Papers.
[24] F. Cerutti, M. Giacomin, M. Vallati, M. 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, Vienna, Austria, July 20–24, 2014, AAAI Press, 2014.
[25] F. Cerutti, N. Oren, H. Strass, M. Thimm, M. Vallati, A benchmark framework for a computational argumentation competition, in: Proceedings of the 5th International Conference on Computational Models of Argument, 2014, pp. 459–460.
[26] F. Cerutti, M. Vallati, M. Giacomin, ArgSemSAT-1.0: exploiting SAT solvers in abstract argumentation, in: [92], pp. 4–7, arXiv:1510.05373, 2015.
[27] F. Cerutti, M. Vallati, M. Giacomin, Where are we now? State of the art and future trends of solvers for hard argumentation problems, in: Computational Models of Argument – Proceedings of COMMA 2016, 2015, pp. 207–218.
[28] F. Cerutti, M. Vallati, M. Giacomin, Efficient and off-the-shelf solver: jArgSemSAT, in: Computational Models of Argument: Proceedings of COMMA 2016, vol. 287, 2016, p. 465.
[29] F. Cerutti, M. Vallati, M. Giacomin, jArgSemSAT: an efficient off-the-shelf solver for abstract argumentation frameworks, in: Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, 2016, pp. 541–544.
[30] G. Charwat, W. Dvorak, S.A. Gaggl, J.P. Wallner, S. Woltran, Methods for solving reasoning problems in abstract argumentation – a survey, Artif. Intell. 220 (2015) 28–63.
[31] A. Cohen, S. Gottifredi, A.J. Garcia, G.R. Simari, A survey of different approaches to support in argumentation systems, Knowl. Eng. Rev. 29 (5) (2014) 513–550.
[32] S. Coste-Marquis, C. Devred, P. Marquis, Symmetric argumentation frameworks, in: Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'05, in: Lect. Notes Comput. Sci., vol. 3571, Springer, 2005, pp. 317–328.
[33] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[34] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.
[35] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artif. Intell. 77 (2) (1995) 321–358.
[36] P.E. Dunne, T.J.M. Bench-Capon, Coherence in finite argument systems, Artif. Intell. 141 (1–2) (2002) 187–203.
[37] P.E. Dunne, W. Dvořák, T. Linsbichler, S. Woltran, Characteristics of multiple viewpoints in abstract argumentation, Artif. Intell. 228 (2015) 153–178.
[38] P.E. Dunne, A. Hunter, P. McBurney, S. Parsons, M. Wooldridge, Weighted argument systems: basic definitions, algorithms, and complexity results, Artif. Intell. 175 (2) (2011) 457–486.
[39] P.E. Dunne, C. Spanring, T. Linsbichler, S. Woltran, Investigating the relationship between argumentation semantics via signatures, in: Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI'16, 2016.
[40] P.E. Dunne, M. Wooldridge, Complexity of abstract argumentation, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 85–104, Ch. 5.
[41] P.E. Dunne, M. Wooldridge, Complexity of abstract argumentation, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 85–104.
[42] W. Dvorák, S.A. Gaggl, J.P. Wallner, S. Woltran, Making use of advances in answer-set programming for abstract argumentation systems, in: Applications of Declarative Programming and Knowledge Management – 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28–30, 2011, in: Lect. Notes Comput. Sci., vol. 7773, Springer, 2011, pp. 114–133, Revised Selected Papers.
[43] W. Dvorák, M. Järvisalo, J.P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, Artif. Intell. 206 (2014) 53–78.
[44] W. Dvořák, Computational Aspects of Abstract Argumentation, Ph.D. thesis, Technische Universität Wien, 2012.
[45] W. Dvořák, M. Järvisalo, J.P. Wallner, S. Woltran, CEGARTIX v0.4: a SAT-based counter-example guided argumentation reasoning tool, in: [92], pp. 12–14, arXiv:1510.05373, 2015.
[46] N. Eén, N. Sörensson, An extensible SAT-solver, in: Proceedings of the International Conference on Theory and Applications of Satisfiability Testing 2004, in: Lect. Notes Comput. Sci., vol. 2919, Springer, 2004, pp. 502–518.
[47] U. Egly, S.A. Gaggl, S. Woltran, ASPARTIX: implementing argumentation frameworks using answer-set programming, in: Logic Programming, Proceedings of the 24th International Conference, ICLP 2008, Udine, Italy, December 9–13, 2008, in: Lect. Notes Comput. Sci., vol. 5366, Springer, 2008, pp. 734–738.
[48] U. Egly, S.A. Gaggl, S. Woltran, Answer-set programming encodings for argumentation frameworks, Argum. Comput. 1 (2) (2010) 147–177.
[49] U. Egly, S. Woltran, Reasoning in argumentation frameworks using quantified Boolean formulas, in: Computational Models of Argument: Proceedings of COMMA 2006, September 11–12, 2006, in: Front. Artif. Intell. Appl., vol. 144, IOS Press, Liverpool, UK, 2006, pp. 133–144.
[50] S. Ellmauthaler, H. Strass, The DIAMOND system for computing with abstract dialectical frameworks, in: Computational Models of Argument: Proceedings of COMMA 2014, 2014, pp. 233–240.

[51] S. Ellmauthaler, H. Strass, DIAMOND: a system for computing with abstract dialectical frameworks, in: [92], pp. 51–53, arXiv:1510.05373, 2015.
[52] P. Erdös, A. Rényi, On random graphs I, Publ. Math. 6 (1959) 290–297.
[53] W. Faber, S. Woltran, Manifold answer-set programs for meta-reasoning, in: Logic Programming and Nonmonotonic Reasoning, Proceedings of the 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14–18, 2009, in: Lect. Notes Comput. Sci., vol. 5753, Springer, 2009, pp. 115–128.
[54] S.A. Gaggl, N. Manthey, ASPARTIX-D: ASP argumentation reasoning tool – Dresden, in: [92], pp. 29–32, arXiv:1510.05373, 2015.
[55] A. Garcia, G.R. Simari, Defeasible logic programming: an argumentative approach, Theory Pract. Log. Program. 4 (1–2) (2004) 95–138.
[56] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Clingo = ASP + control: preliminary report, CoRR arXiv:1405.3694 [abs], 2014.
[57] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–385.
[58] T.F. Gordon, Carneades ICCMA: a straightforward implementation of a solver for abstract argumentation in the Go programming language, in: [92], pp. 54–57, arXiv:1510.05373, 2015.
[59] T.F. Gordon, H. Prakken, D. Walton, The Carneades model of argument and burden of proof, Artif. Intell. 171 (10–15) (2007) 875–896.
[60] É. Grégoire, J. Lagniez, B. Mazure, An experimentally efficient method for (MSS, coMSS) partitioning, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada, AAAI Press, 2014, pp. 2666–2673.
[61] S. Groza, A. Groza, ProGraph: towards enacting bipartite graphs for abstract argumentation frameworks, in: [92], pp. 49–50, arXiv:1510.05373, 2015.
[62] Q. Guo, B. Liao, ZJU-ARG: a decomposition-based solver for abstract argumentation, in: [92], pp. 19–21, arXiv:1510.05373, 2015.
[63] E. Hadjisoteriou, M. Georgiou, ASSA: computing stable extensions with matrices, in: [92], pp. 62–65, arXiv:1510.05373, 2015.
[64] E. Hadjisoteriou, Computing argumentation with matrices, in: Proceedings of the 2015 Imperial College Computing Student Workshop, 2015, p. 29.
[65] A. Hunter, A probabilistic approach to modelling uncertain logical arguments, Int. J. Approx. Reason. 54 (1) (2013) 47–81.
[66] J.-M. Lagniez, E. Lonca, J.-G. Mailly, CoQuiAAS: application of constraint programming for abstract argumentation, in: [92], pp. 25–28, arXiv:1510.05373, 2015.
[67] J.M. Lagniez, E. Lonca, J.G. Mailly, CoQuiAAS: a constraint-based quick abstract argumentation solver, in: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence, JCTAI, November 2015, pp. 928–935.
[68] N. Lamatz, LamatzSolver-v0.1: a grounded extension finder based on the Java-collection-framework, in: [92], pp. 45–48, arXiv:1510.05373, 2015.
[69] D. Lawrence, Genetic Algorithms and Simulated Annealing, Pitman Publishing, 1987.
[70] B. Liao, Toward incremental computation of argumentation semantics: a decomposition-based approach, Ann. Math. Artif. Intell. 67 (3–4) (2013) 319–358.
[71] B. Liao, Efficient Computation of Argumentation Semantics, Intelligent Systems Series, Academic Press, 2014.
[72] B. Liao, L. Jin, R.C. Koons, Dynamics of argumentation systems: a division-based method, Artif. Intell. 175 (2011) 1790–1814.
[73] R.D. Luce, A.D. Perry, A method of matrix analysis of group structure, Psychometrika 14 (1) (1949) 95–116.
[74] S. Modgil, M.W. Caminada, Proof theories and algorithms for abstract argumentation frameworks, in: Argumentation in Artificial Intelligence, Springer Publishing Company, Inc., 2009, pp. 105–129.
[75] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient SAT solver, in: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18–22, 2001, ACM, 2001, pp. 530–535.
[76] J.C. Nieves, U. Cortés, M. Osorio, Preferred extensions as stable models, Theory Pract. Log. Program. 8 (4) (2008) 527–543.
[77] A. Niskanen, J.P. Wallner, M. Järvisalo, Synthesizing argumentation frameworks from examples, in: Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI 2016, in: Front. Artif. Intell. Appl., vol. 285, IOS Press, 2016, pp. 551–559.
[78] S. Nofal, K. Atkinson, P.E. Dunne, Algorithms for decision problems in argument systems under preferred semantics, Artif. Intell. 207 (2014) 23–51.
[79] S. Nofal, K. Atkinson, P.E. Dunne, ArgTools: a backtracking-based solver for abstract argumentation, in: [92], pp. 8–11, arXiv:1510.05373, 2015.
[80] S. Nofal, K. Atkinson, P.E. Dunne, Looking-ahead in backtracking algorithms for abstract argumentation, Int. J. Approx. Reason. 78 (2016) 265–282.
[81] C.H. Papadimitriou, Computational Complexity, Addison Wesley, 1994.
[82] J.L. Pollock, Justification and defeat, Artif. Intell. 67 (1994) 377–407.
[83] O. Rodrigues, GRIS: computing traditional argumentation semantics through numerical iterations, in: [92], pp. 37–40, arXiv:1510.05373, 2015.
[84] O. Rodrigues, Introducing EqArgSolver: an argumentation solver using equational semantics, in: Proceedings of the First International Workshop on Systems and Algorithms for Formal Argumentation, SAFA'2016, in: CEUR Workshop Proc., vol. 1672, 2016, pp. 22–33.
[85] A. Ronca, J.P. Wallner, S. Woltran, ASPARTIX-V: utilizing improved ASP encodings, in: [92], pp. 22–24, arXiv:1510.05373, 2015.
[86] J.P.M. Silva, K.A. Sakallah, GRASP – a new search algorithm for satisfiability, in: ICCAD, 1996, pp. 220–227.
[87] G.R. Simari, R.P. Loui, A mathematical treatment of defeasible reasoning and its implementation, Artif. Intell. 53 (2–3) (1992) 125–157.
[88] M. South, G. Vreeswijk, J. Fox, Dungine: a Java dung reasoner, in: Proceedings of the 2008 Conference on Computational Models of Argument: Proceedings of COMMA 2008, IOS Press, Amsterdam, The Netherlands, 2008, pp. 360–368.
[89] K. Sprotte, ASGL: argumentation semantics in Gecode and Lisp, in: [92], pp. 41–44, arXiv:1510.05373, 2015.
[90] M. Thimm, A probabilistic semantics for abstract argumentation, in: Proceedings of the 20th European Conference on Artificial Intelligence, ECAI'12, August 2012.
[91] M. Thimm, Tweety – a comprehensive collection of Java libraries for logical aspects of artificial intelligence and knowledge representation, in: Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR'14, July 2014, pp. 528–537.
[92] M. Thimm, S. Villata (Eds.), System Descriptions of the First International Competition on Computational Models of Argumentation, ICCMA'15, 2015, arXiv:1510.05373.
[93] F. Toni, M. Sergot, Argumentation and answer set programming, in: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond, in: Lect. Notes Comput. Sci., vol. 6565, Springer, 2011, pp. 164–180.
[94] M. Vallati, F. Cerutti, W. Faber, M. Giacomin, prefMaxSAT: exploiting MaxSAT for enumerating preferred extensions, in: [92], pp. 58–61, arXiv:1510.05373, 2015.
[95] M. Vallati, F. Cerutti, M. Giacomin, Argumentation frameworks features: an initial study, in: Proceedings of the 21st European Conference on Artificial Intelligence, ECAI'14, 2014.
[96] F.H. van Eemeren, B. Garssen, E.C.W. Krabbe, F.A.S. Henkemans, B. Verheij, J.H.M. Wagemans, Handbook of Argumentation Theory, Springer, 2014.
[97] B. van Gijzel, Dungell: a reference implementation of Dung's argumentation frameworks in Haskell, in: [92], pp. 15–18, arXiv:1510.05373, 2015.
[98] B. Van Gijzel, H. Nilsson, A principled approach to the implementation of argumentation models, in: Proceedings of the 2014 Conference on Computational Models of Argument, 2014, pp. 293–300.
[99] T. Wakaki, Preference-based argumentation capturing prioritized logic programming, in: Argumentation in Multi-Agent Systems – 7th International Workshop, ArgMAS 2010, Toronto, ON, Canada, May 10, 2010, in: Lect. Notes Comput. Sci., vol. 6614, Springer, 2010, pp. 306–325, Revised Selected and Invited Papers.
[100] D.J. Watts, S.H. Strogatz, Collective dynamics of small-world networks, Nature 393 (6684) (1998) 440–442.
[101] G. Weissenbacher, S. Malik, Boolean satisfiability solvers: techniques and extensions, in: Software Safety and Security – Tools for Analysis and Verification, in: NATO Sci. Peace Secur. Ser. D Inf. Commun. Secur., vol. 33, IOS Press, 2012, pp. 205–253.
[102] S. Wells, Argument mining: Was ist das?, in: Proceedings of the 14th International Workshop on Computational Models of Natural Argument, CMNA14, 2014.