A First Overview of ICCMA'19*

Stefano Bistarelli[1], Lars Kotthoff[2], Francesco Santini[1], Carlo Taticchi[3]

[1] Dipartimento di Matematica e Informatica, University of Perugia, Italy
[stefano.bistarelli,francesco.santini]@unipg.it
[2] Department of Computer Science, University of Wyoming, USA
larsko@uwyo.edu
[3] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
carlo.taticchi@gssi.it

Abstract. The third International Competition on Computational Models of Argumentation (ICCMA'19) focuses on reasoning tasks in Abstract Argumentation. Submitted solvers are tested on a selected collection of benchmark instances, including artificially generated argumentation frameworks and some frameworks formalizing real-world problems. In this paper we introduce the testing environment set for the competition, including its problems and participants.

1 Introduction

The *International Competition on Computational Models of Argumentation* (*ICCMA*)[4] aims at nurturing research and development of implementations for computational models of argumentation. The objectives of the competition are to provide a forum for empirical comparison of solvers, to highlight challenges to the community, to propose new directions for research and to provide a core of common benchmark instances and a representation formalism that can aid in the comparison and evaluation of solvers [8]. Similar competitions are organised for many different problems. The *MiniZinc Challenge*[5] is an annual competition of *Constraint Programming* solvers on a variety of benchmarks (since 2008). The annual *SAT Competition* is related to *Boolean Satisfiability* (SAT) problems (since 2002).[6] The *International Planning Competition* is a biannual challenge whose aim is to empirically evaluate state-of-the-art planning systems, among several others.[7]

As organisers of the third edition of the competition (ICCMA'19) [7], we proposed two main novelties with respect to the two previous editions, which are described in [17] (ICCMA'15) and [15] (ICCMA'17).

[4] ICCMA Website: http://argumentationcompetition.org.
[5] MiniZinc Challenge: https://www.minizinc.org/challenge.html.
[6] SAT Competition: http://www.satcompetition.org.
[7] Planning copetitions: https://tinyurl.com/uezhalg.

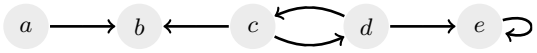

Fig. 1. An example of AF.

First of all, the competition features a new track concerning dynamic *Argumentation Frameworks* (AFs, see Sect. 2), which can measure the efficiency of solvers in recomputing extensions with small modifications to a starting AF. In this track, dedicated to dynamic solvers and approaches [2,4], that is working on dynamic AFs, previous results can be used to rapidly reach a solution in a slightly modified framework, instead of solving the whole problem from scratch.

The second novelty in ICCMA'19 concerns the use of the *Docker*[8]. Docker is a *platform-as-a-service* software that uses operating-system level virtualization to deliver software in packages called *containers*. In this case, our purpose is to encourage the development of solves that can easily run everywhere, so to ease the evaluation phase and allow for the recomputation of the competition results.

The paper is structured as follows: in Sect. 3 we describe the main novelties introduced in ICCMA'19 with respect to previous editions. In Sect. 4 we describe the tasks we tested in the competition. Section 5 shows the input and output formats of abstract argumentation frameworks that solvers were required to deal with. Finally, Sect. 6 outlines ICCMA'19 participants and benchmarks, and how results were evaluated in order to obtain a final ranking. Section 7 reports the final conclusions and future improvements.

2 Abstract Argumentation

An *Abstract Argumentation Framework* (AF, for short) [12] is a tuple $\mathcal{F} = (\mathsf{A}, \rightarrow)$ where A is a set of arguments and $\rightarrow$ is a relation $\rightarrow \subseteq \mathsf{A} \times \mathsf{A}$. For two arguments $a, b \in \mathsf{A}$ the relation $a \rightarrow b$ means that argument a *attacks* argument b. An argument $a \in \mathsf{A}$ is *defended* by $S \subseteq \mathsf{A}$ (in $\mathcal{F}$) if for each $b \in \mathsf{A}$ such that $b \rightarrow a$ there is some $c \in S$ such that $c \rightarrow b$. A set $E \subseteq \mathsf{A}$ is *conflict-free* (in $\mathcal{F}$) if and only if there are no $a, b \in E$ with $a \rightarrow b$. E is *admissible* (in $\mathcal{F}$) if and only if it is conflict-free and each $a \in E$ is defended by E. Finally, the *range* of E (in $\mathcal{F}$) is the set of arguments attacked by E: $E^+ = \{a \in \mathsf{A} \mid \exists b \in E : b \rightarrow a\}$. A directed graph can straightforwardly represent an AF: an example is given in Figure 1.

The *collective acceptability* of arguments depends on the definition of different *semantics*. Four of them are proposed by Dung in his seminal paper [12], namely the complete (**CO**), preferred (**PR**), stable (**ST**) and grounded (**GR**) semantics. In ICCMA we consider them and three additional semantics: semi-stable (**SST**) [10], stage (**STG**) [18], and ideal (**ID**) [13].

[8] Docker.com: `https://www.docker.com`

Semantics determine sets of jointly acceptable arguments, called *extensions*, by mapping each $\mathcal{F} = (\mathsf{A}, \rightarrow)$ to a set $\sigma(\mathcal{F}) \subseteq 2^{\mathsf{A}}$, where 2^{A} is the power-set of A, and σ parametrically stands for any of the considered semantics. The extensions under complete, preferred, stable, semi-stable [10], stage [18], grounded and ideal [13] semantics are defined as follows. Given $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and a set $E \subseteq \mathsf{A}$,

- $E \in \mathbf{CO}(\mathcal{F})$ iff E is admissible in $\mathcal{F}$ and if $a \in \mathsf{A}$ is defended by E in $\mathcal{F}$ then $a \in E$,
- $E \in \mathbf{PR}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and there is no $E' \in \mathbf{CO}(\mathcal{F})$ s.t. $E' \supset E$,
- $E \in \mathbf{SST}(\mathcal{F})$ iff E is complete extension in $\mathcal{F}$ and $E \cup E^+$ is maximal (w.r.t. set inclusion) among all complete extensions in $\mathcal{F}$,
- $E \in \mathbf{ST}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and $E \cup E^+ = \mathsf{A}$,
- $E \in \mathbf{STG}(\mathcal{F})$ iff E is conflict-free in $\mathcal{F}$ and $E \cup E^+$ is maximal (w.r.t. set inclusion) among all conflict-free sets in $\mathcal{F}$,
- $E \in \mathbf{GR}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and there is no $E' \in \mathbf{CO}(\mathcal{F})$ s.t. $E' \subset E$,
- $E \in \mathbf{ID}(\mathcal{F})$ if and only if E is admissible, $E \subseteq \bigcap \mathbf{PR}(\mathcal{F})$ and there is no admissible $E' \subseteq \bigcap \mathbf{PR}(\mathcal{F})$ s.t. $E' \supset E$.

For a more detailed view on these semantics please refer to [3]. Note that both grounded and ideal extensions are uniquely determined and always exist [12,13].

3 Novelties in ICCMA'19

In Sect. 3.1 and Sect. 3.2 we respectively introduce the Docker environment we used to execute and test solvers, and the literature about dynamic solvers.

3.1 The Docker Platform

Docker is an open-source implementation of operating-system-level virtualisation, also known as *containerisation*. It can be used for developing, shipping, and running applications, separating applications from infrastructure with the purpose to deliver software quickly.

Docker is primarily developed for Linux, where it uses the resource isolation features of the Linux kernel such as *cgroups* and kernel *namespaces*, and a *union-capable file system*, to allow independent "containers" to run within a single Linux instance. The main aim is to avoid the overhead of starting and maintaining virtual machines. Docker allows applications to use the same Linux kernel as the system that they are running on and only requires applications to be shipped with things not already running on the host computer. The same container can also be executed on different operating systems: besides different Linux distros such as Debian, Fedora, and Ubuntu, there also exist Docker engines for MacOS, Windows, Amazon Web Services, and Microsoft Azure, which allow for directly moving an application into the cloud without modifications.

The Docker Engine is a client-server application. The first component is a server, i.e., *dockerd*. It listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. The second component is a *REST API* which specifies interfaces that programs can use to talk to the daemon and instruct it what to do. Finally, the third component is a *command line interface* (CLI) client: the CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands.

We require each solver to be submitted to ICCMA'19 to be packaged in a Docker container. To do so, a participant needs three files at least: i) a Dockerfile, ii) a *solver_interface.sh* file, and iii) the solver itself. The Dockerfile defines the environment in the container. The access to resources like networking interfaces is virtualised inside this environment. We suggested to package solvers by using Alpine[9], which is a minimal distro, around 5Mbyte.

3.2 Motivations to Dynamic Frameworks

In previous ICCMA editions, all the frameworks in each database were considered static, in the sense that all the AFs were sequentially passed as input to solvers, representing different and independent problem instances: all tasks were computed from scratch without taking any potentially useful knowledge from previous runs into account.

However, in many practical applications, an AF represents only a temporary situation: arguments and attacks can be added/retracted to take into account new knowledge that becomes available. For instance, in disputes among users of online social networks [16], arguments/attacks are repeatedly added/retracted by users to express their point of view in response to the last move made by the adversaries in the current digital polylogue (often disclosing as few arguments/attacks as possible).

For this reason, ICCMA'19 also featured additional tracks to evaluate solvers on dynamic Dung's frameworks. The aim was to test those solvers dedicated to efficiently recompute a solution after a small change in the original AF. In this case, a problem instance consists of an initial framework (as for classical tracks) and an additional file storing a sequence of additions/deletions of attacks on the initial framework, that is a list of modifications. This file is provided through a simple text format, e.g., a sequence of $+att(a, b)$ (attack addition) or $-att(d, e)$ (attack deletion). The final single output needs to report the solution for the initial framework and as many outputs as the number of changes.

The dynamics of frameworks has attracted recent and wide interest in the Argumentation community. We describe some related work, which also points to the research groups interested in the organisation of such a track. In [9], the authors investigate the principles according to which a grounded extension of a Dungs AF does not change when the set of arguments/attacks are changed. The work of [11] studies how the extensions can evolve when a new argument is considered. The authors focus on adding one argument interacting with one starting

[9] Alpine Linux: `https://alpinelinux.org`.

argument (i.e., an argument which is not attacked by any other argument). In [19], the authors study the evolution of the set of extensions after performing a change operation (addition/removal of arguments/interaction). The work in [4] proposes a division-based method to divide the updated framework into two parts: "affected" and "unaffected". Only the status of affected arguments is recomputed after updates. A matrix-reduction approach that resembles the previous division method is presented in [19].

A work that tests complete, preferred, stable, and grounded semantics on an AF and a set of updates is [1]. This approach finds a reduced (updated) AF sufficient to compute an extension of the whole AF, and uses state-of-the-art algorithms to recompute an extension of the reduced AF only. In [2] the same authors extend their dynamic techniques to improve the sceptical acceptance of arguments in preferred extensions.

Modifications of AFs are also studied in the literature as a base to compute *robustness* measures of frameworks [6]. In particular, by adding/removing an argument/attack, the set of extensions satisfying a given semantics may or may not change. For instance, one could be interested in computing the number of modifications needed to bring a change in this set, or measure the number of modifications needed to have a different set of extensions satisfying a desired semantics. In the latter case, the user is interested in having an estimate on how distant two different points of views are; this kind of approach has also been proposed in [5].

4 The Competition Tracks and Tasks

ICCMA'19 let solvers participate in 7 classical tracks, exactly the same tracks as in ICCMA'17. The tracks are named along the name of semantics, thus we have a track for each $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{GR}, \mathbf{ID}\}$.

The tasks are characterised by a problem and the semantics with which the problem is solved. The considered problems are:

SE-σ: Given $\mathcal{F} = (\mathsf{A}, \rightarrow)$, **return** some set $E \subseteq \mathsf{A}$ that is a σ-extension of $\mathcal{F}$.
EE-σ: Given $\mathcal{F} = (\mathsf{A}, \rightarrow)$, **enumerate** all sets $E \subseteq \mathsf{A}$ that are σ-extensions of $\mathcal{F}$.
DC-σ: Given $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and $a \in \mathsf{A}$, **decide** whether a is credulously accepted in $\mathcal{F}$ under σ.
DS-σ: Given $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and $a \in \mathsf{A}$, **decide** whether a is skeptically accepted in $\mathcal{F}$ under σ.

For single-status semantics (**GR** and **ID**) the problem **EE** is equivalent to **SE**, and **DS** is equivalent to **DC**. Also note that the **DC** problem returns the same results when computed for **CO** and **PR**, but in order to allow the participation in the **PR** track without implementing tasks on the **CO** semantics (or vice versa), both tasks are maintained. Hence, the tasks in ICCMA'19 were:

CO: Complete Semantics (**SE**, **EE**, **DC**, **DS**);

PR: Preferred Semantics (**SE**, **EE**, **DC**, **DS**);
ST: Stable Semantics (**SE**, **EE**, **DC**, **DS**);
SST: Semi-stable Semantics (**SE**, **EE**, **DC**, **DS**);
STG: Stage Semantics (**SE**, **EE**, **DC**, **DS**);
GR: Grounded Semantics (only **SE** and **DC**);
ID: Ideal Semantics (only **SE** and **DC**).

The combination of problems and semantics amounts to a 24 tasks overall. In addition, 4 new tracks were dedicated to the solution of problems over dynamic frameworks, this time using the semantics originally proposed in [12]: $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{GR}\}$. In this case, a problem instance consists of an initial framework and an additional file storing a sequence of additions/deletions of attacks (see Sect. 5 for more details). The dynamic tasks were:

CO: Complete Semantics (**SE**, **EE**, **DC**, **DS**);
PR: Preferred Semantics (**SE**, **EE**, **DC**, **DS**);
ST: Stable Semantics (**SE**, **EE**, **DC**, **DS**);
GR: Grounded Semantics (only **SE** and **DC**).

In this case, the combination of problems with semantics amounts to a total 14 tasks. Tasks in dynamic tracks are invoked by appending "D" at the end of the intended task: for instance, **EE-PR-D** points to the enumeration task with the preferred semantics.

In total, ICCMA'19 was composed of 11 tracks that collect 38 different tasks. Each participating solver could compete in an arbitrary set of tasks. If a solver supported all the tasks of a track (e.g., the track on complete semantics), it also automatically participated in the corresponding track.

5 Input and Output Formats

In the following of this section we first describe the two file format taken as input by solvers. Benchmarks in ICCMA'19 were available in both the formats, in order to allow participating solvers to choose their preferred during the competition.[10] Moreover, we also shortly describe the required output. In this case the format has to be standard in order to evaluate the answers returned by solvers.

5.1 Input Format

Each benchmark instance, that is each AF, is represented in two different file formats: *trivial graph format* (`tgf`) and *aspartix format* (`apx`). We now represent $\mathcal{F} = (\mathsf{A}, \rightarrow)$, where $\mathsf{A} = \{a1, a2, a3\}$ and $\rightarrow = \{(a1, a2), (a2, a3), (a2, a1)\}$, in each of these two formats.

[10] Solvers that can use the two formats were required to select the one they wanted to be tested on in ICCMA'19.

`tgf`[11] is a simple text-based adjacency list file format for describing graphs. The format consists of a list of node labels, followed by a list of edges, which specify node pairs and an optional edge label: in the above example we follow the format 1 2 3 # 1 2 2 3 2 1.

The `apx` format is instead described in [14]. This format is more oriented to Argumentation problems, but carried information is in practice very similar to tgf, even if arguments and attacks are associated with a specific label. In our example we have *arg(a1). arg(a2). arg(a3). att(a1,a2). att(a2,a3). att(a2,a1).*

Both the `tgf` and `apx` formats have been used during the previous editions of ICCMA. The novelty is instead represented by formats for dynamic AFs. For each (dynamic) problem instance, a solver requires to take as input two files: the initial framework (either in `apx` or `tgf` format) and a text file with a list of changes to be applied to it. The file with changes has to report a list of modifications (one per line) over the initial framework. The format of the file with changes has to follow the same format of the original file (either in `apxm` or `tgfm` format, see in the following of this section). Let us introduce an example in `apx`.

Example 1. The initial framework is provided in a file named, for example, `myFile.apx`:

```
arg(a1).
arg(a2).
arg(a3).
att(a1,a2).
att(a2,a3).
```

The second file is a text file containing the list of modifications to be *sequentially* performed on the starting file, one after another. The name of this file has to be the same as the starting framework, with extension .apxm instead of .apx. For this example, `myFile.apxm` is:

```
+att(a3,a2).
-att(a1,a2).
+att(a1,a3).
```

Applying these changes over the initial file corresponds in practice to three full frameworks (besides the initial one), which are represented in Figure 2:

We propose a second example by using the same framework expressed in the `tgf` format this time.

Example 2. The text file with modification needs to have the same name of the the initial framework, with suffix .tgfm instead of .tgf. Hence, in this case `myFile.tgfm` is:

[11] Trivial graph format: `http://en.wikipedia.org/wiki/Trivial_Graph_Format`.

```
arg(a1).      arg(a1).      arg(a1).
arg(a2).      arg(a2).      arg(a2).
arg(a3).      arg(a3).      arg(a3).
att(a1,a2).   att(a2,a3).   att(a2,a3).
att(a2,a3).   att(a3,a2).   att(a3,a2).
att(a3,a2).                 att(a1,a3).
```

Fig. 2. The three AFs obtained from the modifications in `myFile.apxm`.

```
+3 2
-1 2
+1 3
```

As for the previous example, even in this case we obtain three different frameworks, in this case represented in `tgf`. Such AFs are in Figure 3.

```
1         1         1
2         2         2
3         3         3
#         #         #
1 2       2 3       2 3
2 3       3 2       3 2
3 2                 1 3
```

Fig. 3. The three AFs obtained from the modifications in `myFile.tgfm`.

We required ICCMA'19 benchmark generators to produce modifications where attacks have to be added only between existing arguments: no new argument can be introduced. In case all the attacks connected to an argument were removed, such an argument is *not* removed from the framework.

Moreover, benchmarks and benchmark generators needed to provide both the initial framework and the modification file for each instance. For each initial framework, a modification file with at least 15 attack additions/deletions was required.

Therefore, if a modification file had n changes (one per text line), a solver had to run n different problems by applying such modifications in sequence (from the top of the file).[12]

5.2 Output Format

Concerning the same tasks tested in previous ICCMA editions, the format that the output needs to follow does not change. For **DC** and **DS** tasks, the printed

[12] We asked the participants to find solutions sequentially, one modification after another, even if such changes are all in the same file.

Table 1. The list of ICCMA'19 participants. A detailed description of the solvers can be found at: `https://iccma2019.dmi.unipg.it/submissions.html`.

Solver	Authors
Argpref	Alessandro Previti and Matti Järvisalo
ASPARTIX-V19	Wolfgang Dvořák, Anna Rapberger, Johannes P. Wallner and Stefan Woltran
CoQuiAAS v3.0	Jean Marie Lagniez, Emmanuel Lonca and Jean-Guy Mailly
DREDD	Matthias Thimm
EqArgSolver	Odinaldo Rodrigues
Mace4/Prover9	Adrian Groza, Liana Toderean, Emanuel Baba, Eliza Olariu, George Bogdan and Oana Avasi
μ-toksia	Andreas Niskanen and Matti Järvisalo
PYGLAF	Mario Alviano
Yonas	Lars Malmqvist

standard output was required to be either `YES` or `NO`. The **SE** task had to output the list of arguments belonging to the returned extension: for example, $[a1, a2, a4]$. Finally, **EE** had to return the list of extensions in the form $[[a1, a2]\ [a1, a3]\ [a2, a3]]$.

Concerning the dynamic tracks instead, all the answers were in the form of a list where the first element represents the solution of the required task on the initial framework; each following element in this list is the answer returned on the $(i + 1)^{th}$ framework obtained by sequentially applying the first i changes in the modification file: $i \in [1..n]$ and n is the total number of changes in the modification file. **DC**-σ-**D** and **DS**-σ-**D** returns a list of `YES` or `NO`, one for each modification including the initial framework: for example, $[[\texttt{YES}], [\texttt{YES}], [\texttt{NO}], \dots]$. In **SE**-$\sigma$-**D** we have a list of extensions: $[[a1, a3][a1][a1, a2]]$. Finally, the result of **EE**-σ-**D** tasks corresponded to lists of extensions, one for each obtained AF: for instance, $[[[a1, a3]]\ [[a1, a3]]\ [[a1][a1, a3]\ [a1, a2]]\ [[a1, a2]]]$.

6 Participants, Benchmarks, Evaluation

6.1 Participants

The competition received 9 solvers from research groups in Austria, Finland, France, Germany, Italy, Romania and the UK: see Table 1. Among them, 3 were submitted to all the tracks (including dynamic tracks). The authors of the solvers used different techniques to implement their tools. In particular, 4 were based on the transformation of argumentation problems to SAT, 1 on transformation to ASP, 1 relied on machine learning, and 3 were built on tailor-made algorithms.

Table 2 reports every single track each solver participated in.

Table 2. Tasks supported by solvers.

	Dynamic	CO				PR				ST				SST				STG				GR		ID	
		DC	DS	SE	EE	DC	DS	SE	EE	DC	DS	SE	EE	DC	DS	SE	EE	DC	DS	SE	EE	DC	SE	DC	SE
DREDD		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									✓	✓		
CoQuiAAS v3.0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PYGLAF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ASPARTIX-V19		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yonas		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									✓	✓		
Argpref																								✓	✓
EqArgSolver		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									✓	✓		
μ-toksia	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mace4/Prover9		✓	✓	✓																					

6.2 Benchmarks

Each solver had 4GByte of RAM to compute the results of tasks in both the classical and dynamic track. 326 argumentation framework instances were selected among those used in previous competitions as well as 2 new benchmarks submitted for ICCMA'19[13].

The ranking of solvers for a track was based on the sum of scores over all tasks of the track. Ties were broken by the total time it took the solver to return correct results. Note that in order to make sure that each task had the same impact on the evaluation of the track, all tasks for one semantics had the same number of instances. Each solver participating in a task was queried with a fixed number of instances corresponding to the task with a timeout of 10 minutes each. For each instance, a solver got $(0, 1]$ points if it delivered the correct result (it may have been incomplete); -5 points if it delivered an incorrect result; 0 points if the result was empty (e.g., the timeout was reached without answer) or if it was not parsable (e.g., some unexpected error message). In case of testing **SE**, **DC**, and **DS**, the assigned score was 1 if the solver returned the correct answer (respectively "yes", "no", or just an extension). In case of **EE**, a solver received a $(0, 1]$ fraction of points depending on the percentage of found enumerated extensions (1 if it returned all of them).

6.3 Evaluation

The timeout to compute an answer for dynamic tracks was 5 minutes for each framework/change (half of the time in the classical track for a single instance). For the solvers participating in the dynamic tracks, a result was considered correct and complete if, for n changes, $n + 1$ correct and complete results were given. The score for a correct and complete result was 1. A partial (incomplete) result was considered correct if it gave less than $n + 1$ answers, but each of the given answers was correct and complete (with respect to the corresponding static tasks). These rules hold for all the problems (**SE**, **DC**, **DS**, **EE**) in the dynamic tracks. A correct but incomplete result scored a value in $(0, 1]$, depending on the rate of correct sub-solutions given. There was an exception in the case the considered dynamic task involved enumeration (i.e., **EE**): if the last solution a solver provided was correct but partial, then the whole answer was evaluated as the last problem was not solved at all, considering the answer as partial and correct, and fraction of $1/n$ points, depending on the percentage of returned enumerated extensions was assigned. If any of the sub-solution was incorrect, then the overall output was considered incorrect (-5 points). Otherwise, in case no answer was given, 0 points were assigned (for sintance, due to a timeout). In the final ranking, ties were broken by the total time it took the solver to return correct results for all the considered frameworks (initial framework and successive changes).

[13] ICCMA'19 benchmarks: `https://iccma2019.dmi.unipg.it/submissions.html`.

7 Conclusion

The third International Competition on Computational Models of Argumentation (ICCMA'19) focuses on reasoning tasks in abstract argumentation frameworks. Its aim is to provide a forum for empirical comparison of solvers, to highlight challenges to the community, to propose new directions for research and to provide a core of common benchmark instances and a representation formalism that can aid in the comparison and evaluation of solvers.

We have described the environment in which we performed ICCMA19. Using Docker made the competition easy to recompute, allowing submitters and independent parties to easily reproduce our results and build on them to advance the state of the art. As a second improvement, we also organized a track completely dedicated to dynamic solvers, where previous results can be used to rapidly reach a solution on a slightly modified AF, instead of solving the whole problem from scratch.

As future work, we will provide insights on the results obtained by solvers; for a first summary of final solver-rankings, the interested reader can check them from the main Website of ICCMA.[14] In the non-dynamic tracks, μ-toksia is the overall winner, followed by CoQuiAAs and Aspartix19, while in the dynamic track, the first two positions are the same, and then we have Pyglaf.

Acknowledgements

The first and third author have been supported by projects *Argumentation 360* ("Ricerca di Base" 2017–2019) and *Rappresentazione della Conoscenza e Apprendimento Automatico (RACRA)* ("Ricerca di base" 2018–2020).

References

1. Alfano, G., Greco, S., Parisi, F.: Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI. pp. 49–55. ijcai.org (2017)
2. Alfano, G., Greco, S., Parisi, F.: An efficient algorithm for skeptical preferred acceptance in dynamic argumentation frameworks. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI. pp. 18–24. ijcai.org (2019)
3. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. The Knowledge Engineering Review **26**(4), 365–410 (2011)
4. Baroni, P., Giacomin, M., Liao, B.: On topology-related properties of abstract argumentation semantics. A correction and extension to dynamics of argumentation systems: A division-based method. Artif. Intell. **212**, 104–115 (2014)

[14] Rankings: `http://argumentationcompetition.org/2019/results.html`.

5. Baumann, R., Brewka, G.: Expanding argumentation frameworks: Enforcing and monotonicity results. In: Computational Models of Argument (COMMA). Frontiers in Artificial Intelligence and Applications, vol. 216, pp. 75–86. IOS Press (2010)
6. Bistarelli, S., Santini, F., Taticchi, C.: On looking for invariant operators in argumentation semantics. In: Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS. pp. 537–540. AAAI Press (2018)
7. Bistarelli, S., Kotthoff, L., Santini, F., Taticchi, C.: Containerisation and dynamic frameworks in iccma'19. In: Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018) co-located with the 7th International Conference on Computational Models of Argument (COMMA 2018). CEUR Workshop Proceedings, vol. 2171, pp. 4–9. CEUR-WS.org (2018)
8. Bistarelli, S., Rossi, F., Santini, F.: Not only size, but also shape counts: abstract argumentation solvers are benchmark-sensitive. J. Log. Comput. **28**(1), 85–117 (2018)
9. Boella, G., Kaci, S., van der Torre, L.W.N.: Dynamics in argumentation with single extensions: Abstraction principles and the grounded extension. In: Symbolic and Quantitative Approaches to Reasoning with Uncertainty ECSQARU. LNCS, vol. 5590, pp. 107–118. Springer (2009)
10. Caminada, M., Carnielli, W.A., Dunne, P.E.: Semi-stable semantics. J. Log. Comput. **22**(5), 1207–1254 (2012)
11. Cayrol, C., de Saint-Cyr, F., Lagasquie-Schiex, M.: Change in abstract argumentation frameworks: Adding an argument. J. Artif. Intell. Res. **38**, 49–84 (2010)
12. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–358 (1995)
13. Dung, P.M., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation. Artif. Intell. **171**(10-15), 642–674 (2007)
14. Egly, U., Gaggl, S.A., Woltran, S.: Answer-set programming encodings for argumentation frameworks. Argument & Computation **1**(2), 147–177 (2010)
15. Gaggl, S.A., Linsbichler, T., Maratea, M., Woltran, S.: Design and results of the second international competition on computational models of argumentation. Artif. Intell. **279** (2020)
16. Kökciyan, N., Yaglikci, N., Yolum, P.: Argumentation for resolving privacy disputes in online social networks: (extended abstract). In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. pp. 1361–1362. ACM (2016)
17. Thimm, M., Villata, S.: The first international competition on computational models of argumentation: Results and analysis. Artif. Intell. **252**, 267–294 (2017)
18. Verheij, B.: Two approaches to dialectical argumentation: Admissible sets and argumentation stages. In: In Proceedings of the biannual International Conference on Formal and Applied Practical Reasoning (FAPR) workshop. pp. 357–368. Universiteit (1996)
19. Xu, Y., Cayrol, C.: The matrix approach for abstract argumentation frameworks. In: Theory and Applications of Formal Argumentation TAFA. LNCS, vol. 9524, pp. 243–259. Springer (2015)