ICCMA 2023: 5th International Competition on Computational Models of Argumentation

Matti Järvisalo[a], Tuomo Lehtonen[a,b], Andreas Niskanen[a]

[a] *Department of Computer Science, University of Helsinki, Finland*
[b] *Department of Computer Science, Aalto University, Finland*

Abstract

The study of computational models of argumentation and the development of practical automated approaches to reasoning over the models has developed into a vibrant area of artificial intelligence research in recent years. The series of International Competitions on Computational Models of Argumentation (ICCMA) aims at nurturing research and development of practical reasoning algorithms for models of argumentation. Organized biennially, the ICCMA competitions provide a snapshot of the current state of the art in algorithm implementations for central fundamental reasoning tasks over models of argumentation. The year 2023 marked the 5th instantiation of International Competitions on Computational Models of Argumentation, ICCMA 2023. We provide a comprehensive overview of ICCMA 2023, including details on the various new developments introduced in 2023, overview of the participating solvers, extensive details on the competition benchmarks and results, as well as lessons learned.

Keywords: algorithm competitions, formal argumentation, abstract argumentation, structured argumentation, empirical evaluation

---

1. Introduction

The study of computational aspects of argumentation is a topical and vibrant area of artificial intelligence research, aiming to capture rational reasoning when faced with conflicting claims [1, 2, 3, 4]. Motivated by a range of practical settings, including legal reasoning [5, 6, 7, 8, 9, 10, 11], medical diagnostics [12, 13, 14, 15, 16, 17, 18] and other decision support scenarios [19, 20, 21, 22], as well as multi-agent systems [23, 24, 25, 26] and explainability [27, 28, 29, 30, 31], various formal models have been proposed for representing different argumentative scenarios. The types of formal models are traditionally divided into abstract formalisms [32, 33, 34] and structured formalisms [35, 36, 37, 38]. In abstract argumentation, where Dung's abstract argumentation frameworks [32] constitute arguably the most central formalism, arguments and conflicts between arguments are assumed as given, and acceptance statuses of arguments are identified solely based on an attack relation between arguments. In contrast, structured argumentation formalisms allow for a more fine-grained modeling of argumentative settings by representing the underlying rules and premises supporting claims as fundamental building blocks for deriving arguments, conflict information between arguments, and conclusions on the acceptance of claims. The development of practical algorithmic techniques for reasoning over formal models of argumentation—through central reasoning tasks, including credulous and skeptical acceptance of claims—has been identified as an important research direction [39].

By providing further incentives for the research community at large to invest resources towards developing increasingly efficient practical system implementations of algorithms for reasoning over formal models of argumentation, the series of International Competitions on Computational Models of Argumentation (ICCMA, `http://argumentationcompetition.org`) was first established in 2015 [40] and subsequently organized in 2017 [41], 2019 [42, 43], 2021 [44, 43], and most recently—as described in this article—in 2023. Organized biennially, the ICCMA competitions provide a timely snapshot of the current state of the art in algorithm implementations for central reasoning tasks—a great majority of which are computationally difficult, namely, NP/coNP-hard [45]—over formal models of argumentation. At the same time, the competitions provide standard benchmark sets for the use of researchers working on systems implementations and new algorithmic approaches to reasoning over models of argumentation.

The ICCMA competitions invite via open calls for participation the research community at large to submit both (i) system implementations (or solvers, for short) for participating in the competition and (ii) interesting new benchmark instances and generators on which the empirical runtime performance of the submitted solvers are to be evaluated. In terms of models of argumentation, the competition series has focused mainly on abstract argumentation frameworks (AFs) and on key acceptance and related reasoning problems over AFs. Recently—as also detailed in this article—the scope of the competition has widened to cover reasoning over both dynamically changing AFs and structured argumentation.

The year 2023 marked the fifth instantiation of the International Competitions on Computational Models of Argumentation, ICCMA 2023 (`https://iccma2023.github.io`). The main aim of this article is to provide a comprehensive overview of ICCMA 2023. Interestingly, ICCMA 2023 brought on several new developments. These include bringing to fruition for the first time a competition track for system implementations developed for reasoning about acceptance in the structured formalism of assumption-based argumentation (ABA) [46, 38]; changes in the input formats towards more clean and simplistic representation of instances; a new API for developing both benchmarks and solvers developed for reasoning over dynamically changing AFs [47, 48, 49, 50, 51, 52]; as well as witness checking for positive answers reported by competing solvers. All in all, ICCMA 2023 included four tracks: one "Main track" on classical reasoning problems over AFs; one for approximate solvers for some of the Main track problems (the "Approximate track"); one on reasoning over dynamically changing AFs (the "Dynamic track"), and one for reasoning in ABA (the "ABA track"). Our main motivations in this article are to provide a historical record, a detailed account of the various design choices made and the results obtained, and lessons learned from organizing the 2023 instantiation of ICCMA.

For completeness, we start with background on the argumentation formalisms considered in ICCMA 2023 (Section 2). We then proceed with an overview of ICCMA 2023, including detailed descriptions of the competition tracks (Section 3), and rules of the competition and the computing environment the competition was executed on (Section 4). After the overview, we describe the input-output specification (Section 5), the competition benchmarks and witness verification (Section 6), and provide a high-level overview of the solvers submitted to the competition (Section 7). The competition results are provided in Section 8, with fur-

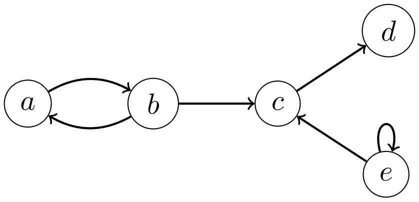


Figure 1: An abstract argumentation framework.

ther details on the empirical data obtained presented in Section 9. Before conclusions, we discuss lessons learned from the competition with a selection of recommendations for future instantiations of ICCMA in Section 10.

2. Argumentation Formalisms in ICCMA 2023

In this section we recall abstract argumentation frameworks (AFs) [32, 33] and assumption-based argumentation frameworks [46, 38], their semantics, and the computational problems focused on in ICCMA 2023.

2.1. Abstract Argumentation

Definition 1. An argumentation framework (AF for short) is a pair $F = (A, R)$, where A is a set of arguments and $R \subseteq A \times A$ is an attack relation. If $(a, b) \in R$, argument a attacks argument b. An argument $a \in A$ is defended by a set $S \subseteq A$ if, for every $(b, a) \in R$, we have $(c, b) \in R$ for some $c \in S$.

Example 1. The AF $F = (A, R)$ with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, a), (b, c), (c, d), (e, c), (e, e)\}$ is illustrated as a directed graph in Figure 1. Each argument is represented as a node, and each attack as a directed edge between the associated arguments. In F, the argument d is defended by $\{b\}$ (as well as any $S \subseteq A$ with $b \in S$), as the only attack (c, d) on d is countered by the attack (b, c).

Semantics for AFs are defined as functions mapping an AF to a collection of jointly acceptable sets of arguments called extensions [33]. In ICCMA, we consider the widely-studied complete [32] (CO), preferred [32] (PR), stable [32] (ST), semi-stable [53] (SST), stage [54] (STG), and ideal [55] (ID) semantics.

Definition 2. Given an AF $F = (A, R)$ and a subset of arguments $S \subseteq A$, the range of S is $S_R^{\oplus} = S \cup \{a \in A \mid (b, a) \in R, b \in S\}$. A set $S \subseteq A$ is conflict-free ($S \in \mathrm{CF}(F)$) if there is no $(a, b) \in R$ with $a, b \in S$, and admissible ($S \in \mathrm{AD}(F)$) if $S \in \mathrm{CF}(F)$ and every $a \in S$ is defended by S. Now, $S \in \mathrm{CF}(F)$ is an extension under

- complete semantics ($S \in \mathrm{CO}(F)$) if $S \in \mathrm{AD}(F)$ and every $a \in A$ defended by S is included in S;
- preferred semantics ($S \in \mathrm{PR}(F)$) if $S \in \mathrm{AD}(F)$ and there is no $T \in \mathrm{AD}(F)$ with $S \subset T$;
- stable semantics ($S \in \mathrm{ST}(F)$) if $S \in \mathrm{CF}(F)$ and $S_R^{\oplus} = A$;

- semi-stable semantics ($S \in \mathrm{SST}(F)$) if $S \in \mathrm{AD}(F)$ and there is no $T \in \mathrm{AD}(F)$ with $S_R^{\oplus} \subset T_R^{\oplus}$;
- stage semantics ($S \in \mathrm{STG}(F)$) if $S \in \mathrm{CF}(F)$ and there is no $T \in \mathrm{CF}(F)$ with $S_R^{\oplus} \subset T_R^{\oplus}$;
- ideal semantics ($S \in \mathrm{ID}(F)$) if $S \in \mathrm{AD}(F)$, $S \subseteq \bigcap \mathrm{PR}(F)$, and there is no $T \in \mathrm{AD}(F)$, $T \subseteq \bigcap \mathrm{PR}(F)$, with $S \subset T$.

Example 2. For the AF F in Example 1 we have

$$\begin{aligned}
\mathrm{CO}(F) &= \{\emptyset, \{a\}, \{b, d\}\}, \\
\mathrm{PR}(F) &= \{\{a\}, \{b, d\}\}, \\
\mathrm{ST}(F) &= \emptyset, \\
\mathrm{SST}(F) &= \{\{b, d\}\}, \\
\mathrm{STG}(F) &= \{\{b, d\}, \{a, c\}\}, and \\
\mathrm{ID}(F) &= \{\emptyset\}.
\end{aligned}$$

Deciding whether a given argument is credulously or skeptically accepted in a given AF constitute two central reasoning problems in abstract argumentation [45].

Definition 3. Let $F = (A, R)$ be an AF and $\sigma \in \{\mathrm{CO}, \mathrm{PR}, \mathrm{ST}, \mathrm{SST}, \mathrm{STG}, \mathrm{ID}\}$ a semantics. A query argument $a \in A$ is (i) credulously accepted under σ in F if $a \in E$ for some $E \in \sigma(F)$ with $a \in E$; (ii) skeptically accepted under σ in F if $a \in E$ for all $E \in \sigma(F)$.

Example 3. Consider the AF F in Example 1 and its extensions (see Example 2). The arguments a, b, and d are credulously accepted under CO and PR. Since there is no stable extension, no argument is credulously accepted under ST, while all arguments are skeptically accepted under ST. Arguments b and d are both credulously and skeptically accepted under SST. Under STG, the arguments a, b, c, and d are credulously accepted, but no argument is skeptically accepted. Finally, since $\emptyset$ is the unique ideal extension, no argument is credulously or skeptically accepted under ID.

The reasoning problems over abstract argumentation frameworks considered in ICCMA 2023 were credulous acceptance, skeptical acceptance, and the problem of finding a single extension under a given semantics.

Definition 4. Consider an AF $F = (A, R)$ and semantics $\sigma = \{\mathrm{CO}, \mathrm{PR}, \mathrm{ST}, \mathrm{SST}, \mathrm{STG}, \mathrm{ID}\}$.

- DC-σ: Is a given query argument $a \in A$ contained in a σ-extension of F?
- DS-σ: Is a given query argument $a \in A$ contained in all σ-extensions of F?
- SE-σ: Return a σ-extension of F or report that one does not exist.

The computational complexity of the decision problems DC and DS is well-established for the various argumentation semantics [45, 56, 57, 58]: DS-CO is in P; DC-CO, DC-PR, DC-ST are NP-complete [56]; DS-ST is coNP-complete [56]; DC-ID and DS-ID are Θ_2^{P}-complete [59]; DC-SST and DC-STG are Σ_2^{P}-complete [60, 61]; and DS-PR, DS-SST and DS-STG are Π_2^{P}-complete [60, 61, 62]. As the ideal extension is unique [55], DC-ID and DS-ID coincide, and both problems can be decided by computing the ideal extension. Furthermore, since every complete extension is contained in a preferred extension, DC-CO coincides with DC-PR. Finally, DS-CO and SE-CO are solvable in polynomial time, and are reducible to the problem of determining the grounded extension.

2.2. Assumption-Based Argumentation

We turn to the structured formalism of assumption-based argumentation (ABA), and specifically the logic programming fragment of ABA [46, 38], which is considered in ICCMA 2023.

Assume a deductive system $(\mathcal{L}, \mathcal{R})$, where $\mathcal{L}$ is a set of atomic sentences, and $\mathcal{R}$ a set of inference rules over $\mathcal{L}$. A rule $r \in \mathcal{R}$ has the form $a_0 \leftarrow a_1, \ldots, a_n$ with $a_i \in \mathcal{L}$ for $0 \leq i \leq n$ and $a_0 \notin \mathcal{A}$. We denote the head of rule r by $head(r) = \{a_0\}$ and the (possibly empty) body of r with $body(r) = \{a_1, \ldots, a_n\}$. An ABA framework consists of a deductive system together with a subset of the atoms being specified as assumptions (which can be provisionally assumed to hold) and which atoms are contrary to assumptions (inducing a conflict).

Definition 5. An ABA framework is a tuple $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, where $(\mathcal{L}, \mathcal{R})$ is a deductive system, $\mathcal{A} \subseteq \mathcal{L}$ a non-empty set of assumptions, and $^-$ a function mapping assumptions $\mathcal{A}$ to sentences $\mathcal{L}$.

We assume the sets $\mathcal{L}$, $\mathcal{R}$ and $\mathcal{A}$ to be finite, and that assumptions do not occur as heads of rules.

Given an ABA framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, the derivability of an atom $a \in \mathcal{L}$ from a set of assumptions $X \subseteq \mathcal{A}$, denoted by $X \vdash a$, is defined as follows. It holds that $X \vdash a$, if either (i) $a \in X$ or (ii) there is a sequence of rules $(r_1, \ldots, r_n)$ in $\mathcal{R}$ with $head(r_n) = a$ such that for reach rule r_i, each atom in the body of r_i either is in X or is the head of a rule earlier in the sequence, i.e., $body(r_i) \subseteq X \cup \bigcup_{j<i} head(r_j)$.

Attacks in ABA are defined between assumption sets.

Definition 6. Let $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an ABA framework, and $X, Y \subseteq \mathcal{A}$ two sets of assumptions. Assumption set X attacks assumption set Y in F if $X \vdash \overline{b}$ for some $b \in Y$.

In words, an assumption set X attacks another assumption set Y if the contrary of an assumption in Y can be derived using $\mathcal{R}$ and X.

Example 4. Consider the ABA framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, with

$$\begin{aligned} &\mathcal{L} = \{a, b, c, x, y, z, w\}, \\ &\mathcal{R} = \{(x \leftarrow y, a), (y \leftarrow c), (z \leftarrow b, c)\}, \\ &\mathcal{A} = \{a, b, c\}, and \\ &\overline{a} = z, \overline{b} = x, \overline{c} = w. \end{aligned}$$

Here $\{c\} \vdash y$, $\{a, c\} \vdash y$, $\{a, c\} \vdash x$, $\{b, c\} \vdash z$ and $\{b, c\} \vdash z$. Since $\overline{a} = z$, we have that $\{b, c\}$ attacks $\{a\}$. Since $\overline{b} = x$, $\{a, c\}$ attacks $\{b\}$ and $\{b, c\}$.

The semantics of ABA are based on the notions of conflict-freeness and defense for assumptions sets.

Definition 7. Let $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{\ })$ be an ABA framework. An assumption set $X \subseteq \mathcal{A}$ is conflict-free if X does not attack itself. The assumption set X defends an assumption set $Y \subseteq \mathcal{A}$ if X attacks all assumption sets $Z \subseteq \mathcal{A}$ that attack Y. Assumption set $X \subseteq \mathcal{A}$ is *admissible* if X defends itself.

We now recall the ABA semantics considered in the ABA track of ICCMA 2023, namely the complete, preferred and stable semantics. For simplicity, we call a set of assumptions that satisfies a given semantics an extension under the semantics.

Definition 8. Let $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{\ })$ be an ABA framework and $X \subseteq \mathcal{A}$ be a conflict-free set of assumptions. Then X is an extension under

- complete semantics ($X \in \mathrm{CO}(F)$) if X is admissible and contains every assumption it defends;
- preferred semantics ($X \in \mathrm{PR}(F)$) if X is complete and there is no complete set of assumptions $Y \subseteq \mathcal{L}$ with $Y \supsetneq X$; and
- stable semantics ($X \in \mathrm{ST}(X)$) if each $\{x\} \subseteq \mathcal{A} \setminus X$ is attacked by X.

Analogously as for AFs, for simplicity we call a set of assumptions $X \in \sigma(F)$ for a given semantics σ and ABA framework F an extension of F under σ. Further, in analogy with the ICCMA 2023 Main track focusing on abstract argumentation, the problems considered in the ICCMA 2023 ABA track were deciding credulous acceptance, deciding skeptical acceptance, and the problem of finding a single extension. In ABA, the acceptance problems are defined in terms of derivability of atoms from extensions.

Definition 9. Consider an ABA framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{\ })$ and semantics $\sigma = \{\mathrm{CO}, \mathrm{PR}, \mathrm{ST}\}$.

- DC-σ: Is a given atom $s \in \mathcal{L}$ derivable from a σ-extension of F?
- DS-σ: Is a given atom $s \in \mathcal{L}$ derivable from all σ-extensions of F?
- SE-σ: Return a σ-extension of F or report that one does not exist.

Example 5. Consider again the ABA framework F from Example 4. The admissible sets of F are $\emptyset$, $\{c\}$, $\{b, c\}$, and $\{a, c\}$, since $\{a, c\}$ and $\{b, c\}$ attack each other and $\{c\}$ is not attacked by any set (as the contrary of c is not derivable). Out of these, $\emptyset$ is not complete, since it defends but does not include $\{c\}$. Finally, $\{a, c\}$ and $\{b, c\}$ are stable and preferred, as they both attack the remaining assumption (b and a, respectively). Thus, under complete, preferred and stable semantics y, x, z, a, b and c are credulously accepted, while only y and c are skeptically accepted.

Similarly as for AFs, the computational complexity of the acceptance problems is well-established for ABA [63, 64, 45] and coincides with the complexity of their abstract argumentation counterpart: DS-CO is in P; DC-CO, DC-PR, DC-ST are NP-complete; DS-ST is coNP-complete; and DS-PR is Π_2^{P}-complete. Similarly as for AFs, credulous acceptance under complete and preferred semantics coincide also in ABA.

3. Competition Tracks

ICCMA 2023 consisted of four competition tracks: the *Main track* and the special *Approximate*, *Dynamic*, and *ABA* tracks. Each of the tracks is composed of multiple subtracks, defined by a combination of a reasoning problem and an argumentation semantics. Solvers could be submitted for evaluation into any choice of subtracks. In other words, no requirements were enforced to require that solvers should support, e.g., all semantics for a specific reasoning problem, or all reasoning problems for a specific semantics.

3.1. Main Track

The Main track concerns solvers for computing extensions and, most centrally, skeptical and credulous reasoning in abstract argumentation frameworks. Abstract argumentation is a unifying approach to argumentation and non-monotonic reasoning, abstracting away the structure of arguments and focusing on the resolution of conflicts between arguments [32]. Abstract argumentation frameworks form the formal basis for much of modern research in computational argumentation [33], underlining the importance of these main reasoning tasks the Main track focuses on. Highlighting their importance, solvers for reasoning in abstract argumentation constitute core reasoning engines for a wide range of argumentative scenarios; see e.g. [8, 9, 10, 11, 16, 19, 23, 27, 28, 29, 30, 65, 66, 67, 68].

In particular, the focus of the ICCMA 2023 Main track was to evaluate single core argumentation reasoning solvers available in open source. Solvers combining different existing solvers in portfolio-style techniques, solvers employing parallel computations via the use of multiple processor cores, as well as solvers which will not be made available in open source were invited to a special No-limits track which consists of the same subtracks as the Main track. This distinction was made in order to allow for separately evaluating sequential solvers and solvers building on top of sequential solvers. The following combinations of reasoning modes and semantics constituted the Main and No-limits subtracks in ICCMA 2023.

- Subtracks concerning credulous reasoning: DC-CO, DC-ST, DC-SST, DC-STG
- Subtracks concerning skeptical reasoning: DS-PR, DS-ST, DS-SST, DS-STG
- Subtracks concerning computing a single extension: SE-PR, SE-ST, SE-SST, SE-STG, SE-ID

Note that DS-CO, SE-CO, DC-PR, DC-ID and DS-ID are not included as subtracks, since each of these either coincides with one of the problems included in the track, or is solvable in polynomial time (recall Section 2.1).

3.2. Approximate Track

Organized since ICCMA 2021, the Approximate track concerns inexact solvers developed for abstract argumentation, i.e., solvers which may in some cases provide incorrect YES/NO answers to credulous/skeptical acceptance. The main motivation behind the Approximate track is to provide incentives to develop practical algorithmic solutions which are more scalable than exact solvers in the Main track by relaxing the requirement of correctness to being correct as often as possible in terms of providing correct YES/NO answers.

The subtracks in the Approximate track were the following.

- Subtracks concerning credulous reasoning: DC-CO, DC-ST, DC-SST, DC-STG, DC-ID
- Subtracks concerning skeptical reasoning: DS-PR, DS-ST, DS-SST, DS-STG

3.3. Dynamic Track

Organized for the first time in ICCMA 2019, the Dynamic track concerns solvers for answering credulous/skeptical acceptance queries in a dynamically evolving AF as a form of argumentation dynamics, motivated by the fact that in various application scenarios such as disputes between agents in online social networks [69] the attack relation is subject to temporal changes, arising from e.g. the fact that disputes may change (be retracted or added) due to new available knowledge [49]. The goal of the Dynamic track is to provide incentives for developing argumentation solvers supporting efficient reasoning under such dynamic settings.

An instance of the Dynamic track consists of an initial AF, as well as a sequence of operations corresponding to

- changes in the structure of the current AF, i.e., additions or deletions of arguments or attacks,
- declarations of query arguments for acceptance tasks, and
- solve calls, in which case a solver must report either YES or NO according to the task-semantics combination corresponding to the subtrack.

The subtracks in the ICCMA 2023 Dynamic track were DC-CO, DC-ST, and DS-ST, which correspond to all NP-complete acceptance problems considered in the Main track. The DS-PR subtrack was also included in the call for participation, but as only one solver supported it, DS-PR was excluded as a subtrack from the competition.

The sequence of operations is issued to a participating solver via an API which we detail in Section 5.4. Using this API, we developed a simple benchmark application which takes an AF and a query argument as input, extracts a subgraph to construct an initial AF, iteratively changes the structure of the AF as well as the query arguments for acceptance tasks. We detail this application in Section 6.2.

3.4. ABA Track

The ABA track, realized for the first time in ICCMA 2023, concerns solvers developed for reasoning in the structured argumentation formalism of assumption-based argumentation (ABA). The ABA track was originally proposed for ICCMA 2021. However, the track was cancelled in 2021 due to lack of solver submissions. For 2023, we opted to re-organize the ABA track, building on the well-thought-out plans for an ABA track from 2021. Regarding the choice of ABA in particular among the various existing structured argumentation formalisms as the focus of a structured argumentation track at ICCMA, we note that there has recently been active development of algorithms for reasoning in ABA [70, 71, 72, 73, 74, 75, 76, 77, 78] as well as work on applying ABA in application settings [12, 13, 14, 15, 30, 24, 79, 80, 81] which suggested that an ABA track might be feasible for 2023.

The subtracks in the ABA track are DC-CO, DC-ST, DS-PR, DS-ST, SE-PR, and SE-ST. Similarly as for the Main track, DS-CO, SE-CO and DC-PR were not considered due to either coinciding with one of the organized subtracks or due to being solvable in polynomial time (recall Section 2.2).

4. Organizational Details

We continue with an overview of central organizational details of ICCMA 2023, including key rules and requirements, correctness specifications for solvers and solver testing employed by the organizers, the schemes used for ranking solvers in the tracks, and the specification of the computing cluster used to run ICCMA 2023.

4.1. Organizers and Steering Committee

The authors of this article constitute the organizers of ICCMA 2023. The Steering Committee of the ICCMA competition series made the decision to invite the organizers. Furthermore, all central decisions (such as the organized competition tracks and rules) were made with the approval of the 2021–2023 ICCMA Steering Committee.[1]

4.2. Open Source Requirement

For all tracks apart from No-Limits, solver source code originating from the authors (including modifications to third-party source code as part of a solver) had to be submitted together with a corresponding solver binary. The source code package of each participating solver were to be made available at the time when the results of ICCMA 2023 were presented at the KR 2023 conference. This rule was enforced to ensure that the research community at large would be able to access, make use of, and potentially study ways of further improving the state-of-the-art solvers after the competition. In case bug fixes were submitted during the evaluation (i.e., if requested by the evaluation organizers), the bug-fixed source package had to also be submitted together with a bug-fixed solver binary. Solvers were allowed to use external binaries or unmodified third-party libraries. However, if a solver implementation used non-standard libraries (beyond STL, Boost, etc.) its identity and usage in the solver had to be clearly specified in the solver description.

4.3. Use of Processor Cores and Eligibility of Portfolio Solvers

In all tracks except for the No-Limits track, solvers were required to use only a single core of the processor on the computing node it was run on. Solvers making use of multiple cores would be disqualified. This limitation did not concern the No-Limits track, where parallel computations were allowed. Portfolio solvers, which combine several existing argumentation solvers were not allowed to participate in the competition apart from the No-Limits track, where portfolios were allowed. The organizers reserved the right to may move a solver from the Main track to the No-Limits track if a solver was deemed to violate these conditions. As a results, one solver was moved from Main to No-Limits track due to multithreading.

4.4. Correctness Requirements

We applied the following definition of correct solvers. A solver is correct in a subtrack if it fulfills the following requirements for every instance executed during the evaluation. For every subtrack, the output of the solver must conform to the I/O requirements (see Section 5). No additional output was allowed. If the solver terminates without running out of time or memory, it must exit without any errors and fulfill the following requirements.

---

[1] The composition of the 2021–2023 ICCMA Steering Committee was as follows: Sarah A. Gaggl (President), Johannes P. Wallner (Vice-President), Jean-Guy Mailly (Secretary), Andrea Cohen, Jean-Marie Lagniez, Matthias Thimm, and Serena Villata.

- Main track and No-Limits track:
  - DC-σ: If the query argument is credulously accepted, the solver outputs YES along with a certificate, i.e., a σ-extension containing the query. Otherwise, the solver outputs NO.
  - DS-σ: If the query argument is not skeptically accepted, the solver outputs NO along with a certificate, i.e., a σ-extension not containing the query. Otherwise, the solver outputs YES.
  - SE-σ: The solver outputs a σ-extension.
- Approximate track: The solver must output YES or NO; certificates are not required.
- Dynamic track: Every function specified in the API (as detailed later on in Section 5) employed by the benchmark application is correctly implemented. A benchmark instance is solved when the benchmark application terminates correctly. If the benchmark application issues a sequence of API calls which are not supported by the solver, the solver must enter state ERROR. If this state is encountered, the solver will be excluded from the corresponding subtrack, but will not be disqualified.
- Assumption-based Argumentation track: Same requirements as in Main track, but without the need to produce a certificate.

A solver would not be immediately disqualified if it outputs a wrong solution during the execution of the evaluation. The organizers provided all participants a fair chance of submitting bug fixes to their solvers in a timely manner based on feedback from the organizers regarding incorrect results. If the bug resulting in incorrect behavior could not be resolved by the solver developers, the submitters were still given the option of having the solver's correct results tabulated and reported among the ICCMA 2023 results. However, the results would be marked to indicate that the solver also generated some incorrect results.

4.5. Testing Solvers for Correctness

To detect bugs in solvers early, the organizers fuzz tested all submitted solvers before the evaluation was performed and reported any erroneous results to the submitters. For each solver and subtrack, hundreds of random frameworks were generated and tested against a track-specific reference solver. Multiple rounds of such feedback were provided to the solver developers in case the solver continued to not pass the tests.

For the Main track, the following fuzz testing procedure was implemented to test the correctness of solver outputs, including witnesses. We generated AFs according to a random model inspired by [82]. For each AF $F = (A, R)$, we first sampled the number of arguments n uniformly at random from an interval $[10, 50]$. To construct R, for each pair of arguments $a_1, a_2 \in A$ we let an attack (a_1, a_2) exist with probability 0.1. If an attack exists, it is symmetric with probability 0.05. Finally, for each $a \in A$, a self-attack (a, a) exists with probability 0.02. For acceptance tasks (DC and DS), a query argument was generated uniformly at random.

For each semantics (CO, PR, ST, SST, STG, ID), we called a reference solver to enumerate all extensions to be able to verify witnesses reported by the solvers (in addition to YES/NO answers). As a reference solver, we used μ-TOKSIA (ICCMA 2019 version) due to its success in ICCMA 2019 [43]. For each subtrack supported by a participating solver, we called the participating solver to obtain a YES/NO answer and, depending on the task and answer, a witness extension. We checked the following properties against the extensions reported by the reference solver.

- DC: If the answer is YES, a witness was reported and the witness is an extension under the considered semantics and *contains* the query argument. If the answer is NO, *no extension contains* the query argument.
- DS: If the answer is NO, a witness was reported and the witness is an extension under the considered semantics and *does not contain* the query argument. If the answer is YES, *all extensions contain* the query argument.
- SE: If the answer is NO, no extension exists. Otherwise, a witness exists and is an extension under the considered semantics.

This procedure was repeated for 100 iterations for each participating solver. Using fuzz testing, we determined that one solver submitted to the Main track reported incorrect extensions for the STG semantics. The corresponding input AFs and tasks were reported to the developers of the solver. After obtaining bug fixes from the developers of the solver, all solvers submitted to the Main track passed this procedure.

For the Dynamic track, we employed a similar fuzz testing procedure as for the Main track. For every iteration of the procedure (repeated 100 times), we generated an initial AF using the same random model as used for fuzzing in the Main track. A task-semantics combination (DC-CO, DC-ST, DS-ST) was chosen uniformly at random. A participating solver was initialized with the initial AF and the chosen semantics via IPAFAIR. For the next 500 iterations, we applied a dynamic change (addition or deletion of an argument or attack; chosen uniformly at random from all available combinations) to the current AF, and issued the change to the participating solver via IPAFAIR. Then, similarly to the Main track, we enumerated all extensions of the AF under the current semantics using a reference solver (μ-TOKSIA, ICCMA 2019 version), and verified that the result reported by the participating solver agrees with the extensions reported by the reference solver, and that the state of the solver is not erroneous. We observed both incorrect results and crashes on two solvers submitted to the Dynamic track. These observations were reported to the developers by providing the corresponding initial AF and a Python script containing the trace of IPAFAIR calls issued during fuzzing. After obtaining bug fixes from the developers, all solvers passed the fuzz testing procedure.

For the ABA track, the following fuzz testing procedure was implemented. We generated ABA frameworks with a simplified version of a random model used in [73], with 25 atoms and 8 assumptions. Assumptions were randomly assigned a contrary from the set of all atoms. For each non-assumption atom, 1–5 rules were generated with 1–5 atoms in the rule body. Both values were selected separately and uniformly at random, as were the atoms occurring in a given rule body. As a reference solver we used ASPforABA, a mature solver that has been publicly available and empirically evaluated prior to the competition [75]. We checked that the reference solver and each tested solver agreed on their answer (YES or NO) on the acceptance subtracks DC-CO, DC-ST, DS-PR, and DS-ST. When an answer reported by a submitted solver differed from the answer given by the reference solver, the framework was inspected by hand to verify that the error was in the tested solver. Instances on which a solver reported an erroneous result were reported to the submitters of the solver. The correctness of the solvers in the actual evaluation corresponded to testing: each solver that passed the fuzz testing reported only correct answers in the actual evaluation and each solver that did not in the end pass the fuzz testing reported some erroneous results in the actual evaluation.

4.6. Ranking Schemes

For the Main, Dynamic, and ABA tracks, the score of a solver on a subtrack is the sum of the so-called PAR-2 scores [83]—as a standard ranking scheme employed in various other solver competitions (see e.g. [84])—of the solver over all instances of a subtrack. The PAR-2 score assigned to a solver on an individual instance is the CPU time used by the solver on the instance if the solver solved the instance within resource limits, and 2× the per-instance time limit otherwise. In other words, the PAR-2 score penalizes a solver timeout by double the time limit. For the No-limits track, PAR-2 based on wall-clock time, that is, elapsed time as measured by the internal clock of the computer, instead of CPU time was used in order to account for parallel processing. The winner of a subtrack is the solver with the lowest score. For the Approximate track, due to the inexactness of solvers, incorrect solutions are discarded, and the solver with the largest number of correctly solved instances wins. If needed, cumulative CPU running time over solved instances was used as a tie-breaker.

4.7. Computing Environment and Resource Limits

ICCMA 2023 was executed on a computing cluster of the University of Helsinki, Finland, with the following homogeneous node specification: 2.60-GHz Intel Xeon E5-2670 CPUs and 57GB RAM under AlmaLinux 8.4, including GCC 12.2.0, Clang 12.0.1, Boost 1.76.0, GLib 2.68.2, Rust 1.70.0, Java 17.0.4, and Python 3.9.5. The memory limit of 16 GB per instance was enforced in all tracks. The per-instance time limit for all but the Approximate and No-limit tracks was set to 1200 seconds CPU time per instance. For the Approximate track, the time limit was 60 seconds CPU time per instance, and for the No-limits track 1200 seconds wall-clock time per instance. For the Dynamic track, the resource limits were applied to each benchmark instance as a whole (including multiple changes to the AF and acceptance queries).

4.8. Further Rules

Mandatory Solver Descriptions. Each solver entrant to ICCMA 2023 was required to accompanied by a short, 1–2 page written description of the system. The description needed to include a list of all authors of the system and their present institutional affiliations, provide details of any non-standard algorithmic techniques (e.g., heuristics, simplification/learning techniques, etc.) and data structures in the solver, as well as references to relevant literature (be they by the authors themselves or by others). The solver descriptions were compiled by the organizers into a technical report [85] which was made openly available so that the research community can for future purposes refer to the individual solver descriptions.

Number of Submissions and Withdrawal. As a further rule, a solver could be withdrawn from ICCMA 2023 only before the deadline for the submission of the final versions. After this deadline no further changes or withdrawals of the solvers are possible. This rule was enforced in order to avoid potential late withdrawals of solvers that might have not reached top positions. The option to withdraw was not in fact used by any participant. The evaluation organizers also reserved the right to restrict the number of solver submissions originating from the same author(s) based on computation resource limitations. The application of this rule did not turn out to be necessary.

Participation of Organizers. The organizers were allowed to enter their own solvers into the evaluation. The steering committee approved this decision. The reason for this decision was that the organizers have contributed to various argumentation solvers, and it was not considered beneficial for the community to leave out these solvers from the evaluation. That said, strict measures were enforced to avoid providing them with advantage over other participants. In particular, benchmark selection was done using a random seed—811543731122527—concatenated from numbers sent separately to the organizers by each ICCMA steering committee member (this procedure also aimed to ensure that no specific steering committee member could single-handedly decide the random seed). The seed and benchmark selection scripts are openly available on the ICCMA 2023 website. Furthermore, hashes of solver source codes (in the form of git commit IDs) originating from the organizers were communicated to the steering committee before benchmark selection to confirm that the organizers did not modify their solvers after benchmark selection.

5. I/O Requirements and Processing

ICCMA 2023 brought on new developments in the ways in which benchmark instances are input to the participating argumentation solvers.

In previous instantiations of ICCMA, two input formats were supported for representing abstract argumentation frameworks (see, e.g., [40]): the Trivial Graph Format (TGF, `https://en.wikipedia.org/wiki/Trivial_Graph_Format`) and the so-called ASPARTIX format. TGF is a standard format for representing directed graphs using indices for nodes (arguments), with directed edges (attacks) listed line-by-line as pairs of indices. The ASPARTIX format, named after the answer set programming (ASP) [86, 87] based argumentation solver ASPARTIX [88], is an answer set programming style input format, listing the existence of arguments (using the predicate `arg/1`) and attacks (using the predicate `att/2`) as grounded facts. In 2021, an extension of the ASPARTIX format to representing ABA frameworks was also proposed (although the then-planned ABA track did not unfortunately come to existence due to lack of participants). For the Dynamic track, in 2021 the TGF and ASPARTIX formats were used and extended to include information on the initial AF as well as all the changes the AF would be subject to in the same file (which means that that solvers in the Dynamic track were made aware of all future changes already at the time of reading in the initial AF).

As detailed next, for ICCMA 2023 a single indexing-based numerical input format for AFs was enforced for the Main, Dynamic and Approximate tracks, based on a proposal originating from the community[2] to move to such an input format We also extended the format for use in the ABA track by beginning-of-line identifiers for distinguishing between assumptions, rules and contraries. The main motivations were to move from more verbose and non-unique formats to a single more compact format which is simple to parse and which directly provides

---

[2] We acknowledge Matthias Thimm for proposing this to the ICCMA 2023 organizers.

indexing of the basic elements over which AF and ABA frameworks are defined. A single format also avoids possible issues related to diverging inputs when allowing a choice of multiple formats, and also makes the execution of solvers in the competition more straightforward.

5.1. Input Format for Abstract Argumentation Frameworks

The following AF input file format was used in the Main, No-Limits, Dynamic, and Approximate tracks.

The arguments of an AF with n arguments are indexed consecutively with positive integers from 1 to n. The first line of the input file is the unique "p-line" of the form

```
p af <n>
```

where `<n>` is the number of arguments n, ending with the newline character. An attack $a \to b$, where a has index i and b has index j, is expressed as the line

```
i j
```

ending with the newline character. In addition to the p-line and lines expressing attacks, lines starting with the character `#` are allowed. These lines are interpreted as comment lines unrelated to the syntax of the input AF, and end with the newline character. No other lines are allowed.

Example 6. Consider the AF in Example 1 over the arguments $\{a,b,c,d,e\}$ with the attacks (a, b), (b, a), (b, c), (c, d), (e, c), $(e, e)\}$. Assuming the indexing $a = 1$, $b = 2$, $c = 3$, $d = 4$, $e = 5$, this AF is specified as follows.

```
p af 5
# this is a comment
1 2
2 1
2 3
3 4
5 3
5 5
```

5.2. Input Format for ABA Frameworks

In the ABA track, the following ABA input file format was used. The atoms of an ABA framework with n atoms are indexed consecutively with positive integers from 1 to n. The first line of the input file is of the unique "p-line" of the form

```
p aba <n>
```

where `<n>` is the number of atoms `n`, ending with the newline character. Assumptions, rules and contraries are specified on individual lines. A line starting with `a`, followed by an index between 1 and n, specifies that the atom with the index is an assumption. A line starting with `c`, followed by two space-separated indices between 1 and n, specified that the atom corresponding to the second index is a contrary of the atom corresponding to the first index. A line starting with `r` followed by a space-separated list of indices between 1 and n specify a rule whose head is the atom corresponding to the first index in the list and whose body consists of the atoms corresponding to the subsequent indices in the list. Each line starting with `a`, `c` or `r` ends with the newline character. In addition to the p-line and lines starting with `a`, `c` or `r`, lines starting with the character `#` are allowed. These lines are interpreted as comment lines unrelated to the syntax of the input ABA framework, and end with the newline character. No other lines are allowed.

Example 7. Consider the ABA framework from Example 4 with rules $(p \leftarrow q, a)$, $(q \leftarrow)$, $(r \leftarrow b, c)$, assumptions $\{a,b,c\}$, and contraries $\overline{a} = r$, $\overline{b} = s, \overline{c} = t$ is specified as follows, assuming the atom-indexing $a = 1$, $b = 2$, $c = 3$, $p = 4$, $q = 5$, $r = 6$, $s = 7$, $t = 8$.

```
p aba 8
# this is a comment
a 1
a 2
a 3
c 1 6
c 2 7
c 3 8
r 4 5 1
r 5
r 6 2 3
```

5.3. Output Formats

In all tracks except for the Dynamic track, the solvers were required to output their results through standard output in the following format.

5.3.1. Credulous Reasoning (DC)

Main and No-Limits tracks. If the query argument is determined to be credulously accepted, the solver should output the line "YES" followed by a line specifying a witness, i.e., a σ-extension containing the query. For example, if the solver finds the σ-extension $\{1, 2, 5\}$ containing the query argument 1, the solver output should be the following.

```
YES
w 1 2 5
```

If the query argument is determined not to be credulously accepted, the solver should output the single line "NO".

Approximate and ABA tracks. If the query argument is determined to be credulously accepted, the solver output should be the single line "YES". If the query argument is determined not to be credulously accepted, the solver should output the single line "NO".

5.3.2. Skeptical Reasoning (DS)

Main and No-Limits tracks. If the query argument is determined not to be skeptically accepted, the solver should output the line "NO" followed by a line specifying a counterexample, i.e., a σ-extension not containing the query. For example, if the solver finds the σ-extension $\{1,4\}$ not containing the query argument 2, the solver output should be the following.

```
// Semantics supported by IPAFAIR.
typedef enum { AD, CO, PR, ST, SST, STG, ID } semantics;
// Construct a new AF solver and return a pointer to it.
void * ipafair_init ();
// Release the solver, i.e., all its resources and allocated memory.
void ipafair_release (void * solver);
// Set the argumentation semantics for the next calls of 'ipafair_solve'.
void ipafair_set_semantics (void * solver, semantics sem);
// Add the given argument to the current argumentation framework.
void ipafair_add_argument (void * solver, int32_t arg);
// Delete the given argument from the current argumentation framework.
void ipafair_del_argument (void * solver, int32_t arg);
// Add the given attack (s,t) to the current argumentation framework.
void ipafair_add_attack (void * solver, int32_t s, int32_t t);
// Delete the given attack (s,t) from the current argumentation framework.
void ipafair_del_attack (void * solver, int32_t s, int32_t t);
// Add an assumption for the next call of 'ipafair_solve'.
void ipafair_assume (void * solver, int32_t arg);
// Solve the current instance in the credulous reasoning mode.
int32_t ipafair_solve_cred (void * solver);
// Solve the current instance in the skeptical reasoning mode.
int32_t ipafair_solve_skept (void * solver);
// Determine whether the given argument is contained in a solution or counterexample.
int32_t ipafair_val (void * solver, int32_t arg);
```

Figure 2: Functions declared in the IPAFAIR header.

```
NO
w 1 4
```

If the query argument is determined to be skeptically accepted, the solver should output the single line “YES”.

Approximate and ABA tracks. If the query argument is determined to be credulously accepted, the solver output should be the single line “YES”. If the query argument is determined not to be credulously accepted, the solver output should be the single line “NO”.

5.3.3. Computing a Single Extension (SE)

If a σ-extension is identified, the solver should output a line specifying such an extension. For example, if the solver finds the σ-extension $\{3, 7\}$, the solver should must be the following.

```
w 3 7
```

If it is determined that there is no σ-extension, the solver output should output the line “NO”.

5.4. IPAFAIR: API for the Dynamic Track

In the Dynamic track, I/O is implemented via an API titled IPAFAIR (Re-entrant Incremental Argumentation Framework solver API), an incremental API for reasoning in AFs. We designed IPAFAIR in the style of IPASIR [89], a standard API for incremental Boolean satisfiability (SAT) [90, 91] solving. IPAFAIR is available in open source under `https://bitbucket.org/coreo-group/ipafair`, with both C and Python versions available. The repository also contains an example C-to-Python wrapper and examples of its usage.

The functions declared in the IPAFAIR header are listed in Figure 2. Using IPAFAIR, an external program can initialize a solver with an input AF and semantics, modify a current AF, and make credulous and skeptical acceptance queries. In analogy to IPASIR [89], `ipafair_init` and `ipafair_release` are used to initialize and release a solver. Specific to argumentation solvers, an argumentation semantics (an `enum` type) can be set using `ipafair_set_semantics`. A current AF is specified using calls to `ipafair_add_argument`, `ipafair_del_argument`, `ipafair_add_attack`, and `ipafair_del_attack`, which add/delete arguments/attacks. Note that arguments are simply positive integers (in line with the input format of ICCMA 2023). Acceptance queries over arguments defined via `ipafair_add_argument` are defined using `ipafair_assume`. Note that in contrast to the DC and DS tasks in the Main track, IPAFAIR allows for setting multiple query arguments.

Two function declarations—`ipafair_solve_cred` and `ipafair_solve_skept` for the credulous and skeptical reasoning modes, respectively—are provided for solving a current instance (consisting of a specified AF, semantics and acceptance query/queries). In the credulous (resp. skeptical) reasoning mode, the task is to decide whether *all arguments* assumed via `ipafair_assume` are contained in *some extension* (resp. *all extensions*) of the current AF under the current semantics. If the answer is *YES*, the function returns 10, and the state is changed to SAT. If the answer is *NO*, the function returns 20, and the state is changed to UNSAT. If the solver does not support the sequence of API calls performed, the function returns -1 and the state of the solver is changed to ERROR. To retrieve the witnessing extension (in credulous reasoning mode) or counterexample extension (in skeptical reasoning mode), given an argument in the current AF, the function `ipafair_val` returns a positive value if the argument is contained in the extension, and a negative value if it is not contained in the extension. This function can only be used if `ipafair_solve_cred` has returned 10, or `ipafair_solve_skept` has returned 20, and the state of the solver has not changed. Note that the "single extension" task can be solved without specifying any assumptions in the credulous reasoning mode.

The Python version of IPAFAIR contains similar functions to the C header. As the main differences, the AF semantics is specified in the constructor, and the constructor includes an input AF filename as an optional argument for specifying the initial AF. Further, assumptions are provided as optional arguments to `solve_cred` and `solve_skept` as lists of integers. Finally, a function `extract_witness` returns the witness or counterexample extension. For the competition, a solver must implement the Python version of the API. Alternatively, an example C-to-Python wrapper is provided in the repository, as well as the fuzz testing tool developed for the dynamic track (recall Section 4.5).

5.5. Witness Checking

For the Main track, all extensions (including witnesses and counterexamples for DC and DS tasks) returned by solvers were checked using an external routine. The routine takes the input instance—consisting of a reasoning task, an AF, and a query argument (for DC and DS tasks)—and the output file returned by a participating solver as command-line arguments. First, we check that an extension is contained in the output file (on a "`w`-line") in the required cases, namely SE (with the exception of stable semantics where a "NO" answer suffices when an extension does not exist), DC in the case of a "YES" answer, and DS in the case of a "NO" answer. We also check that the query argument is contained in the witness extension in the credulous case, and that it is not contained in the counterexample extension in the skeptical case.

We implemented the checking of the provided extension by making use of an external Boolean satisfiability (SAT) [91] solver and standard encodings for complete and stable semantics [92]. For a given AF and semantics, the encoding produces a formula in propositional logic the satisfying assignments of which correspond to extensions of the AF under the given semantics.

To check a given extension under complete and stable semantics, we assign in the encodings the truth values of variables that correspond to arguments in the extension (i.e. assign variables corresponding to arguments within the extension to true and variables corresponding to other arguments to false), and check with a SAT solver that the resulting formula is satisfiable. For preferred, semi-stable, and stage semantics, we begin by similarly verifying that the extension extends to a satisfiable assignment of the encoding corresponding to the base semantics: conflict-free for stage, and complete for preferred and semi-stable. For preferred semantics, we continue by asking the SAT solver for a complete extension which is a superset of the given extension (in the style of [93]), to confirm this call returns "unsatisfiable". In this case the extension reported by the participating solver is preferred. Similarly, for semi-stable (resp. stage) semantics, we ask for a complete extension (resp. conflict-free set) whose range is a superset of the range of the witness extension, and verify that the result is "unsatisfiable". For further guarantees on correctness, we recorded the proofs of unsatisfiability [94] obtained from the SAT solver (Glucose [95] version 4.1 via PySAT [96]) in the DRUP format, and used an external proof checker (DRAT-trim [97]) to verify that the proofs were correct.

Finally, for ideal semantics (the SE-ID task), since the ideal extension is unique, we verified that for each input AF instance the extensions reported by all Main and No-limits track solvers are the same. This approach is also motivated by the fact that in contrast to other semantics considered in ICCMA 2023, under standard complexity-theoretic assumptions verifying an ideal extension—a Θ_2^p-complete task [59, 45]—is not possible using a single NP oracle call.

All witnesses were successfully verified, apart from the following exceptions. On the SE-PR and DS-PR tasks, the verification procedure timed out for a single input instance; the corresponding extensions were afterwards successfully verified using a longer time limit. On the SE-SST task, two timeouts were observed, and with a longer time limit the procedure ran out of memory due to the size of the proof of unsatisfiability under construction. Finally, 14 additional memory-outs occurred due proof construction: 10 on SE-PR, and 1 on DC-SST, DC-STG, DS-SST, and DS-STG.

6. Benchmarks

We continue by detailing the construction of the ICCMA 2023 benchmark sets. For AFs, we made use of existing benchmark AFs from which the benchmark sets of previous ICCMA competitions were sampled from. These benchmark domains are briefly outlined in the following. In addition, a dedicated call for benchmarks was issued in conjunction to a call for solvers, where the argumentation community was invited to submit new and challenging AFs and ABA frameworks in the specified input format. We also welcomed submissions of benchmark generators, that is, software for generating AFs or ABA frameworks together with suggestions for suitable parameter values for generating interesting benchmark instances. Especially for the Dynamic track, the community was invited to submit Python programs employing IPAFAIR (see Section 5.4). As a result, we obtained (only) one benchmark submission, namely a new benchmark generator for AFs.

6.1. Main and Approximate Tracks

For ICCMA 2015 [40], three different graph generators were implemented [98] for generating hard AF instances.

- **GroundedGenerator** is a generator for AFs with a large grounded extension to test whether solvers can exploit reasoning under grounded semantics.
- **SccGenerator** is a generator for AFs with many strongly connected components in order to test whether solvers can exploit techniques based on decomposing AFs.
- **StableGenerator** is a generator for AFs with many stable extensions (and hence, many preferred and complete extensions) in order to penalize solvers which decide acceptance based on enumerating extensions.

Since ICCMA 2015 benchmark AFs were also featured in subsequent iterations of ICCMA [41, 42], for ICCMA 2023 we decided to use a new set of benchmark AFs using these generators [98] with similar parameters as in ICCMA 2015 [40, 41], resulting in 100 AFs for each generator.[3]

In ICCMA 2017, in addition to benchmark AFs generated with the ICCMA 2015 generators, benchmarks from six new benchmark domains were included in the ICCMA 2017 benchmark set [41].

- **ABA2AF** is a set of 426 AFs resulting from a translation of ABA frameworks to abstract argumentation [71].
- **AdmBuster** is a crafted set of 15 benchmark AFs for strong admissibility [99].
- **Barabasi-Albert** is a set of 500 AFs generated using AFBenchGen2 [100] according to the Barabasi-Albert graph model [101].
- **Erdös-Rényi** is a set of 500 AFs generated using AFBenchGen2 [100] according to the Erdős-Rényi graph model [102].
- **Planning2AF** is a set of 385 planning instances translated first to a propositional formula [103] and then to AFs [104].
- **SemBuster** is a crafted set of benchmarks for semi-stable semantics [105], consisting of 16 instances.
- **Traffic** is a set of 600 AFs obtained from real-world traffic networks.
- **Watts-Strogatz** is a set of 400 AFs generated using AFBenchGen2 [100] according to the Watts-Strogatz graph model [106].

---

[3] The authors thank Matthias Thimm for generating these AFs.

Table 1: Benchmark statistics per domain.

Domain	Number of AFs	Min. $\|A\|$	Avg. $\|A\|$	Max. $\|A\|$
ABA2AF	426	8	112	1449
AdmBuster	15	1000	526733	2500000
AFGen	17	100	231	512
Barabasi-Albert	500	21	111	201
crusti_g2io	450	3875	46889	89425
Datalog	134	2	1434	11775
Erdös-Rényi	500	101	301	502
GrdGenerator	100	1034	2240	3801
Planning2AF	385	86	765	5660
SCCGenerator	100	219	4470	9976
SemBuster	16	60	2713	7500
StableGenerator	100	400	942	1497
Traffic	600	2	1562	15605
Watts-Strogatz	400	100	300	500
ICCMA 2023	329	100	29791	2500000

We consider all of the corresponding benchmark AFs for the construction of the ICCMA 2023 benchmark set.

In ICCMA 2019, two new benchmark submissions were received and used for the ICCMA 2019 benchmark set [42, 43]. We considered all of the submitted benchmark AFs for constructing the ICCMA 2023 benchmark set.

- **AFGen** is a benchmark generator based on a random graph model [82]. The benchmark set consists of 17 sample AFs.
- **Datalog**$^{\pm}$ is a benchmark set of 134 AFs built from knowledge bases expressed in Datalog$^{\pm}$ [107].

For this edition of ICCMA, we received a benchmark generator called **crusti_g2io** based on an inner-outer random graph model [108]. In ICCMA 2021, all new benchmark AFs were obtained by using a similar generator [43]. Therefore we did not include benchmark AFs from ICCMA 2021, and instead generated new AFs using suggested parameter choices (see [85] for the parameters) for crusti_g2io. In particular, for each of the nine suggested parameter combinations, we generated 50 AFs using the random seed (recall Section 4.8) provided by the ICCMA steering committee for benchmark generation and sampling. This procedure resulted in a total of 450 AFs.

The 14 benchmark domains used for constructing the ICCMA 2023, the respective number of AFs, as well as statistics on the number of arguments in these AFs, are reported in Table 1. We excluded all AFs with less than 100 arguments from consideration. For each domain, we sampled 25 benchmark AFs, except for the novel crusti_g2io domain for which we sampled 25% more (i.e. a total of 32 AFs) If the domain contains less than 25 benchmark AFs (i.e., the AdmBuster, AFGen, and SemBuster domains), we included all of the AFs in that domain. This procedure resulted in a total of 329 benchmark AFs. For each of the benchmark AFs, a

Algorithm 1 Algorithm used for benchmarks in the Dynamic track of ICCMA 2023.
Input: AF $F = (A, R)$, query $q \in A$, subtrack (DC-CO, DC-ST, DS-ST).
Parameters: $p_{\text{fixed}} = 0.333$, $p_{\text{added}} = 0.333$, $n_{\text{iter}} = 64$, $n_{\text{changes}} = 32$, $n_{\text{queries}} = 16$

```
Perform breadth-first search starting from $q$, constructing a mapping *depth* from arguments to their distance to $q$.
Starting from arguments with lowest *depth*, mark $p_{\text{fixed}} \cdot |A|$ arguments as *fixed*.
Continuing similarly, mark the next $p_{\text{added}} \cdot |A|$ arguments as *existing*.
Initialize the set of arguments $A' = \{a \in A \mid a \text{ is existing}\}$ and attacks $R' = \{(a, b) \in R \mid a, b \text{ are existing}\}$.
Set $Q = \{q\}$, and sample $n_{\text{queries}} - 1$ additional fixed arguments to $Q$.
for $i = 1, \dots, n_{\text{iter}}$ do
    for $q \in Q$ do
        Decide acceptance status of $q$ in $F' = (A', R')$.
    for $k = 1, \dots, n_{\text{changes}}$ do
        Set $p =$ number of existing non-fixed arguments/number of non-fixed arguments.
        With probability $p$, delete an existing non-fixed argument from $A'$ along with its incident attacks from $R'$.
        Otherwise (with probability $1 - p$), add a non-existing argument to $A'$ along with its incident attacks to $R'$.
```

query argument was sampled uniformly at random from the set of arguments which are not self-attacking nor have zero indegree, in order to avoid trivial acceptance queries. This set of benchmark AFs and queries was used directly for all subtracks of the Main and Approximate tracks.

6.2. Dynamic Track

Recall that in the Dynamic track, a benchmark instance consists of a sequence of calls issued via IPAFAIR (Section 5.4). As a base for issuing these calls, we used all of the 329 benchmark AFs and corresponding query arguments of the benchmark set of the Main and Approximate tracks. To issue this sequence of IPAFAIR calls, we implemented the following procedure, outlined as Algorithm 1. An AF $F = (A, R)$, a query argument $q \in A$, and subtrack specification is obtained as input. Starting from the query argument, we label $p_{\text{fixed}} \cdot |A|$ (with $p_{\text{fixed}} = 0.333$) of the arguments in the AF as *fixed*, i.e., included in the initial AF and not subject to deletion. By continuing the search, we label the next $p_{\text{added}} \cdot |A|$ (with $p_{\text{added}} = 0.333$) of arguments in the AF as the initial *existing* arguments in the current (first) AF (which are, however, subject to deletion). We sample a total of $n_{\text{queries}} = 16$ query arguments, including the original query argument, from the set of fixed arguments. Then, for a total of $n_{\text{iter}} = 64$ iterations, first, the acceptance status of each query argument is decided using an IPAFAIR call corresponding to the subtrack. In each iteration, we then perform $n_{\text{changes}} = 32$ changes to the current AF, each change being an addition or deletion of a non-fixed argument along with its incident attacks from F. The choice between an addition or deletion is made at random, using a probability based on the current number of existing arguments, so that the number of arguments remains balanced throughout the execution of the algorithm.

6.3. ABA Track

In contrast to the more heterogeneous set of abstract argumentation frameworks employed in the Main track, benchmarks for the ABA track were in this first instantiation of the track generated with a simple random instance generator. The varying parameters are the number of atoms (25, 100, 500, 2000 or 5000), the proportion of atoms that are assumptions (10% or 30%), the maximum number of rules deriving each non-assumption atom (5 or 10), and the maximum size of each rule body (5 or 10). The number of rules deriving any given atom was selected uniformly at random from the interval $[1, n]$ for $n \in \{5, 10\}$, and similarly for the size of each rule (i.e., number of atoms in the body of a rule). Additionally, for each assumption, a contrary was selected uniformly at random from the set of all atoms. Ten instances with each combination of the four parameters were generated for a total of 400 benchmark instances. For acceptance problems, the query for each instance was selected at random from the atoms for which there the ABA framework at hand includes at least one derivation.

7. Participants

We continue with an overview of the solvers submitted to ICCMA 2023. An overview of the participating teams and the tracks their solvers participated in is shown in Table 2. In total, there were two participants in the Main and No-limits tracks, three in the Dynamic track, and five in both the Approximate track and the ABA track. Our overview of the individual solvers is based on the system descriptions submitted by the respective solver authors in conjunction with their solver. For more details, we refer the reader to the separate technical report containing all the solver descriptions [85]. The source codes of all submitted solvers are available at `https://iccma2023.github.io/solvers.html`.

Many of the submitted solvers share similarities. Out of the 14 solvers, 8 are based on the declarative approach, i.e., on translating a given argumentation problem to a constraint modelling language, using a constraint solver for the modelling language to obtain a solution to the argumentation problem. The eight solver based on the declarative approach are each based on either Boolean satisfiability (SAT) [91] or answer set programming (ASP) [86, 87]. Notably, apart from the Approximate track, only one submitted solver did not use SAT or ASP.

AFGCN v2 [109] (by Lars Malmqvist) is written in Python and employs graph convolutional neural networks (utilizing the libraries PyTorch [110] and Deep Graph Library [111]) and supports all subtracks of the Approximate track. AFGCN v2 uses a neural network trained to approximate acceptance of claims with various graph properties, such as PageRank, in-degrees and out-degrees, as input features.

ARIPOTER-DEGREES [112] (by Jérôme Delobelle, Jean-Guy Mailly and Julien Rossit) is written in Java and supports all subtracks of the Approximate track. ARIPOTER-DEGREES computes the grounded extension and accepts the query argument if it either is included in the grounded extensions or its out-degree (number of arguments the query attacks) is sufficiently high compared to its in-degree (number of arguments that attack the query).

ARIPOTER-HCAT [112] (by Jérôme Delobelle, Jean-Guy Mailly and Julien Rossit) is written in Java and supports all subtracks of the Approximate track. ARIPOTER-HCAT accepts arguments primarily based on whether they belong to or are attacked by the grounded

Table 2: Overview of the participating solvers and authors. Solvers participating in the No-limits track are marked with ✓*.

Solver	**Authors**	**Main**	**Approximate**	**Dynamic**	**ABA**
AFGCN v2	Lars Malmqvist (University of York)		✓		
ARIPOTER-Degrees	Jérôme Delobelle (Paris Cité University) Jean-Guy Mailly (Paris Cité University) Julien Rossit (Paris Cité University)		✓		
ARIPOTER-HCAT	Jérôme Delobelle (Paris Cité University) Jean-Guy Mailly (Paris Cité University) Julien Rossit (Paris Cité University)		✓		
ASTRA	Andrei Popescu (TU Graz) Johannes P. Wallner (TU Graz)				✓
AcbAr	Tuomo Lehtonen (University of Helsinki) Anna Rapberger (TU Wien) Markus Ulbricht (Leipzig University) Johannes P. Wallner (TU Graz)				✓
ASPforABA	Tuomo Lehtonen (University of Helsinki) Matti Järvisalo (University of Helsinki) Johannes P. Wallner (TU Graz)				✓
Crustabri	Jean-Marie Lagniez (University of Artois) Emmanuel Lonca (University of Artois) Jean-Guy Mailly (Paris Cité University)	✓		✓	✓
fargo-limited	Matthias Thimm (University of Hagen)		✓		
flexAble	Martin Diller (TU Dresden) Sarah Alice Gaggl (TU Dresden) Piotr Gorczyca (TU Dresden)				✓
Fudge	Matthias Thimm (University of Hagen) Federico Cerutti (University of Brescia) Mauro Vallati (University of Huddersfield)	✓*			
harper++	Matthias Thimm (University of Hagen)		✓		
κ-solutions	Christian Pasero (TU Graz) Johannes P. Wallner (TU Graz)			✓	
μ-toksia	Andreas Niskanen (University of Helsinki) Matti Järvisalo (University of Helsinki)	✓		✓	
PORTSAT	Sylvain Declercq (Paris Cité University) Quentin Januel Capellini (Sorbonne University) Christophe Yang (Paris Cité University) Jérôme Delobelle (Paris Cité University) Jean-Guy Mailly (Paris Cité University)	✓*			

extension (accepting in the former and rejecting in the latter case), and secondarily based on whether their hcat score, adapted from the h-Categorizer gradual semantic [113], is sufficiently high.

ASTRA [78] (by Andrei Popescu and Johannes P. Wallner) is written in Python and supports DC-CO, DC-ST, DS-ST and SE-ST in the ABA track. ASTRA employs D-FLAT [114, 115], an ASP-based tool for dynamic programming, exploiting tree-decompositions. In this context, ASTRA uses a tree-decomposition of the graph given by the atoms and rules of an ABA framework as nodes and edges.

AcbAr [70] (by Tuomo Lehtonen, Anna Rapberger, Markus Ulbricht and Johannes P. Wallner) is written in Python and supports all subtracks of the ABA track. AcbAr implements a polynomially-bounded reduction from ABA to AF and employs the AF solver μ-toksia [116] on the resulting AF.

ASPforABA [75, 117] (by Tuomo Lehtonen, Matti Järvisalo and Johannes P. Wallner) is written in Python and supports all subtracks of the ABA track. ASPforABA is based on answer set programming (ASP), utilizing encodings of ABA semantics in terms of assumption sets, without explicit construction of arguments.

Crustabri (by Jean-Marie Lagniez, Emmanuel Lonca and Jean-Guy Mailly) SAT-based solver, supporting all subtracks in the Main track and ABA track, and DC-CO, DC-ST, and DS-ST in the Dynamic track. Crustabri is a revised version of CoQuiAAS [118] for ICCMA 2023, rewritten in Rust and using CaDiCaL [119] as the SAT solver. For the Dynamic track, Crustabri uses the SAT solver incrementally [90], activating and deactivating parts of the SAT encoding related to attacks and arguments between the SAT solver invocations. For ABA, Crustabri generates an AF and uses its own AF reasoning to solve the given task.

fargo-limited (by Matthias Thimm) is written in C++, supporting all subtracks of the Approximate track. fargo-limited implements a depth-bounded depth-first search algorithm for admissible sets. For credulous acceptance, fargo-limited answers YES if an admissible extension containing the query is found, and similarly YES for skeptical acceptance if in addition no attacker of the query is contained in an admissible extension. Thereby, each YES answer should be correct, but the solver may report NO incorrectly.

flexAble [76, 77] (by Martin Diller, Sarah Alice Gaggl, Piotr Gorczyca) is written in Scala, supporting DC-CO and DC-ST in the ABA track. flexAble implements specialized reasoning algorithms for ABA, namely, flexible dispute derivations constructing dialectical (tree-like) justifications for accepted claims.

Fudge [120] (by Matthias Thimm, Federico Cerutti and Mauro Vallati) is a SAT-based solver written in C++, supporting all subtracks of the No-limits track. Beyond direct SAT encodings, Fudge implements iterative SAT-based approaches for preferred and ideal semantics, using CaDiCaL [119] as the SAT solver.

harper++ (by Matthias Thimm) is written in C++, supporting all subtracks of the Approximate track. harper++ is based on approximating acceptance via the grounded extension; the solver outputs YES to credulous acceptance if the query argument is contained in the grounded extension or not attacked by an argument in the grounded extension, and YES to skeptical acceptance if the query argument is contained in the grounded extension.

κ-solutions (by Christian Pasero and Johannes P. Wallner) is written in Python, supporting for all subtracks of the Dynamic track. κ-solutions implements a SAT-based approach, using Z3 [121] as the SAT solver. The solver computes up to $k = 3$ witnesses ($k = 3$) with several calls to the SAT solver. Upon changes to the AF, κ-solutions first checks if any precomputed witness is a witness for the new AF, and only computes new witnesses if this is not the case.

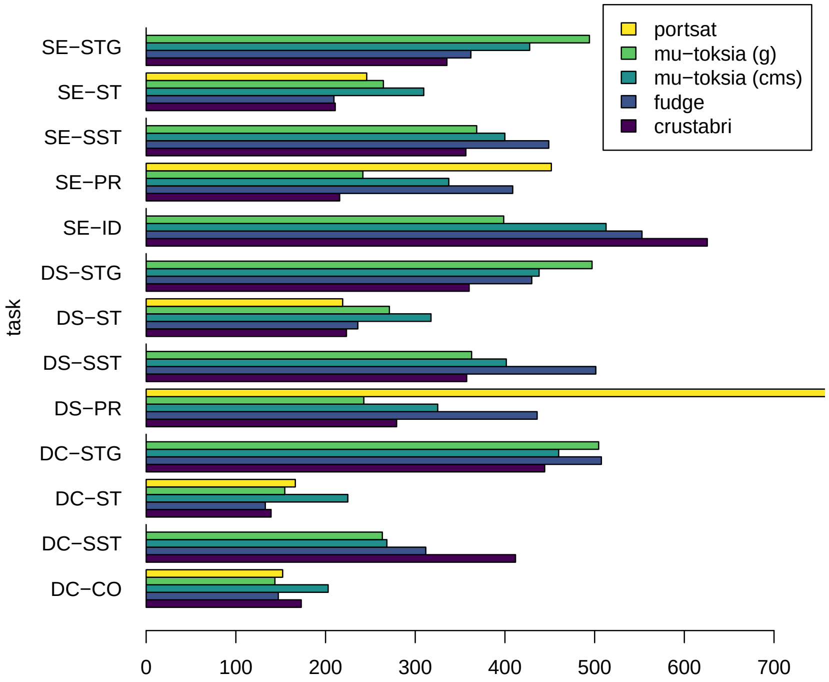


Figure 3: Main and No-limits tracks: PAR-2 scores. Note that PORTSAT and FUDGE are No-limits solvers, while the other solvers competed in the Main track.

μ-TOKSIA [49, 116] (by Andreas Niskanen and Matti Järvisalo) is a SAT-based solver written in C++ with support for all subtracks of the Main and Dynamic tracks. For the Main track, two configurations were submitted, one using GLUCOSE [95] and one using CRYPTOMINISAT [122] as the SAT solver. For the Dynamic track, configurations with and without incremental SAT solving were submitted, both using GLUCOSE as the SAT solver. The incremental configuration uses the SAT solver incrementally, activating and deactivating attacks and arguments via assumptions. The static configuration encodes the AF from scratch at each iteration.

PORTSAT (by Sylvain Declercq, Quentin Januel Capellini, Christophe Yang, Jérôme Delobelle and Jean-Guy Mailly) is written in Rust supporting for DC-CO, DC-ST, DS-PR, DS-ST, SE-PR, and SE-ST subtracks of the No-limits track. PORTSAT is a SAT-based parallel portfolio approach, invoking a set of SAT solvers in parallel (MINISAT [123], MANYSAT [124], MAPLESAT [125], and GLUCOSE [95]).

As agreed with the ICCMA steering committee, for transparency, access to the implementations of all solver submissions involving any of the organizers of ICCMA 2023 was provided to the ICCMA steering committee before the submission deadline, before the steering committee provided the random seed used for benchmark selection.

8. Overview of Competition Results

In this section we provide an overview of the results of ICCMA 2023, as presented in Tables 3a–6 and Figures 3–6. Beyond this overview, further analysis of the empirical data is provided later in Section 9.

Table 3: Rankings and PAR-2 scores for the Main (white) and No-limits (gray) tracks.

(a) Credulous acceptance (DC) subtrack.

	Rank (PAR-2 score)			
Solver	DC-CO	DC-SST	DC-ST	DC-STG
Crustabri	2 (172.92)	3 (411.80)	**1 (139.29)**	**1 (444.33)**
Fudge	- (147.31)	- (311.79)	**- (132.86)**	- (507.53)
μ-toksia (cmsat)	3 (202.88)	2 (268.39)	3 (224.83)	2 (459.92)
μ-toksia (glucose)	**1 (143.56)**	**1 (263.32)**	2 (154.56)	3 (504.51)
PORTSAT	- (152.20)	-	- (166.32)	-

(b) Skeptical acceptance (DS) subtrack.

	Rank (PAR-2 score)			
Solver	DS-PR	DS-SST	DS-ST	DS-STG
Crustabri	2 (279.27)	**1 (357.38)**	**1 (223.34)**	**1 (360.12)**
Fudge	- (435.91)	- (501.33)	- (236.00)	- (429.91)
μ-toksia (cmsat)	3 (325.07)	3 (401.54)	3 (317.58)	2 (438.09)
μ-toksia (glucose)	**1 (242.69)**	2 (362.83)	2 (271.21)	3 (497.12)
PORTSAT	- (1151.41)	-	**- (219.11)**	-

(c) Single extension (SE) subtrack.

	Rank (PAR-2 score)				
Solver	SE-ID	SE-PR	SE-SST	SE-ST	SE-STG
Crustabri	3 (625.59)	**1 (215.76)**	**1 (356.48)**	**1 (210.83)**	**1 (335.33)**
Fudge	- (552.81)	- (408.70)	- (448.90)	**- (209.39)**	- (362.04)
μ-toksia (cmsat)	2 (512.76)	3 (337.43)	3 (399.93)	3 (309.49)	2 (427.59)
μ-toksia (glucose)	**1 (398.65)**	2 (241.65)	2 (368.52)	2 (264.52)	3 (494.24)
PORTSAT	-	- (451.73)	-	- (245.92)	-

8.1. Main Track

Starting with the Main track, the PAR-2 scores of all solvers in each subtrack of the Main track, including the No-limits solvers, are shown in Figure 3. The relative rankings of the solvers and PAR-2 scores of each solver are listed in Table 3a for the problem of credulous acceptance (DC), in Table 3b for skeptical acceptance (DS), and Table 3c for finding a single extension (SE). Here μ-toksia (cmsat) and μ-toksia (glucose) stand for the versions of μ-toksia with CryptoMiniSat and Glucose as the SAT solver, respectively. Overall, Crustabri had the lowest PAR-2 score and thus ranked first in most tracks (nine): DC-ST, DC-STG, DS-SST, DS-ST, DS-STG, SE-PR, SE-SST, SE-ST, and SE-STG. μ-toksia, using Glucose as the SAT solver, ranked first in the remaining four subtrack: DC-CO, DC-SST, DS-PR, and SE-ID. In the No-limits track, the PAR-2 score of Fudge (based on wall-clock time) was lower than the PAR-2 score (based on CPU time) of the best solver in the subtracks DC-ST and SE-ST. However, as No-limits solvers were allowed to utilize multiple CPU cores and employ a portfolio of solvers, it may be considered surprising that the No-limits solvers Fudge and PORTSAT did not significantly outperform the Main track solvers overall. We note here that Fudge was not originally submitted to the No-limits track but was moved

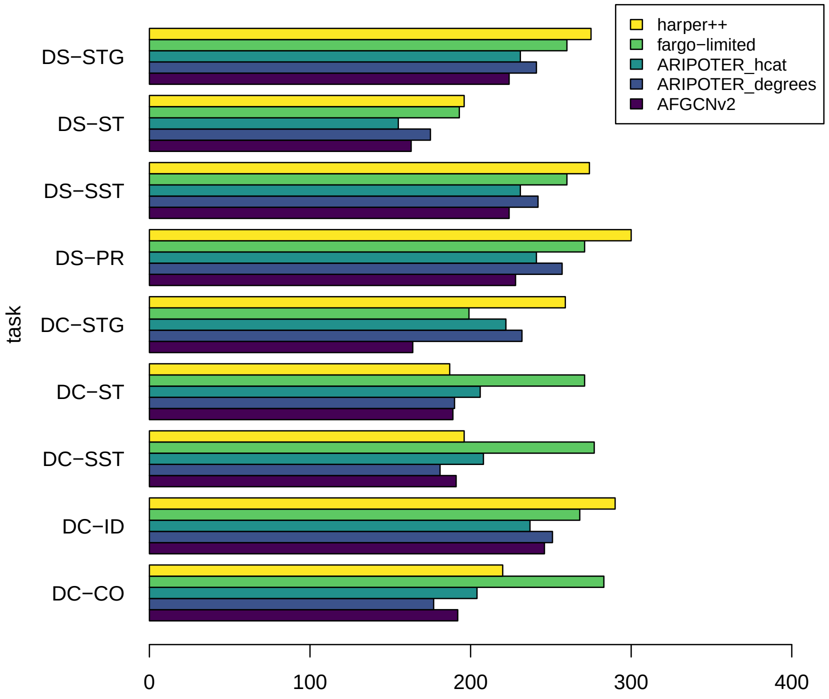


Figure 4: Approximate track: Number of solved instances

there by the organizers in agreements with the solver authors after realizing that the solver actually did make at least in cases light use of multithreading (the author of the solver opted not to revise the solver to avoid this).

8.2. Approximate Track

The results of the Approximate track are summarized in Figure 4, showing the number of solved instances by each solver, with the PAR-2 scores and ranks of the solvers shown in Table 4a and Table 4b for the problems of credulous acceptance and skeptical acceptance, respectively. Note here for the Approximate track "solved instance" means that a solver reported the correct answer to the instance at hand. HARPER++ ranked first most often, namely six times. This interestingly includes all skeptical subtracks: DC-ID, DC-STG, PS-PR, DS-SST, DS-ST, and DS-STG. FARGO-LIMITED ranked first in the other three subtracks, DC-CO, DC-SST, and DC-ST. Interestingly, the winning margins to the rank-2 solver were particularly high in the individual subtracks in which FARGO-LIMITED ranked first in: 283 vs 220 in DC-CO, 277 vs 208 in DC-SST, and 271 vs 206 solved instances in DC-ST. In the other subtracks, the winning margins between the first and second ranking solver was less than 30 solved instances.

8.3. Dynamic Track

Results of the Dynamic track are summarized in Figure 5, showing the PAR-2 scores of each participating solver, with the rankings and PAR-2 scores of the solvers also shown in Table 5. CRUSTABRI dominated the Dynamic track overall, ranking first in each of the subtracks DC-CO, DC-ST, and DS-ST. The winning margins are particularly high in the

Table 4: Rankings and number of solved instances for the Approximate track.

(a) Credulous acceptance (DC) subtrack.

	Rank (# solved)				
Solver	DC-CO	DC-ID	DC-SST	DC-ST	DC-STG
AFGCNv2	4 (192)	4 (246)	4 (191)	4 (189)	5 (164)
ARIPOTER (DEGREES)	5 (177)	3 (251)	5 (181)	3 (190)	2 (232)
ARIPOTER (HCAT)	3 (204)	5 (237)	2 (208)	2 (206)	3 (222)
FARGO-LIMITED	**1 (283)**	2 (268)	**1 (277)**	**1 (271)**	4 (199)
HARPER++	2 (220)	**1 (290)**	3 (196)	5 (187)	**1 (259)**

(b) Skeptical acceptance (DS) subtrack.

	Rank (# solved)			
Solver	DS-PR	DS-SST	DS-ST	DS-STG
AFGCNv2	5 (228)	5 (224)	4 (163)	5 (224)
ARIPOTER (DEGREES)	3 (257)	3 (242)	3 (175)	3 (241)
ARIPOTER (HCAT)	4 (241)	4 (231)	5 (155)	4 (231)
FARGO-LIMITED	2 (271)	2 (260)	2 (193)	2 (260)
HARPER++	**1 (300)**	**1 (274)**	**1 (196)**	**1 (275)**

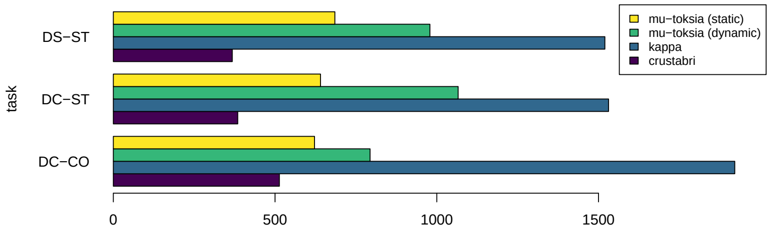


Figure 5: Dynamic track: PAR-2 scores

Table 5: Dynamic track: Rankings and PAR-2 scores

	Rank (PAR-2 score)		
Solver	DC-CO	DC-ST	DS-ST
CRUSTABRI	**1 (513.37)**	**1 (384.68)**	**1 (367.82)**
μ-TOKSIA (STATIC)	2 (622.01)	2 (640.56)	2 (684.92)
μ-TOKSIA (DYNAMIC)	3 (793.80)	3 (1066.09)	3 (978.76)
κ-SOLUTIONS	4 (1921.25)	4 (1531.09)	4 (1519.69)

DC-ST and DS-ST subtracks. In DC-ST, CRUSTABRI scored 384.68 compared to the 640.56 of the rank-2 solver μ-TOKSIA; for DS-ST, the scores are 367.82 against 684.92.

8.4. ABA Track

Finally, results of the ABA track are summarized in Figure 6 in terms of the PAR-2 scores of each solver, with the rankings and PAR-2 scores also shown in Table 6. In the ABA track, CRUSTABRI was disqualified in each subtrack due to producing erroneous output. We include it here in gray, with PAR-2 scores computed by treating the instances with erro-

Table 6: ABA track: Rankings and PAR-2 scores. (Crustabri shown in gray due to disqualification.)

	Rank (PAR-2 score)					
Solver	DC-CO	DC-ST	DS-PR	DS-ST	SE-PR	SE-ST
AcbAr	2 (1087.05)	2 (1060.93)	2 (1120.31)	2 (1053.63)	2 (1104.19)	2 (1067.90)
ASPforABA	**1 (120.61)**	**1 (105.08)**	**1 (156.52)**	**1 (118.24)**	**1 (147.79)**	**1 (119.11)**
ASTRA	4 (2382.00)	4 (2371.69)	-	3 (2400.00)	-	3 (2400.00)
Crustabri	- (1087.65)	- (1081.38)	- (1081.64)	- (1075.51)	- (1182.66)	- (1105.09)
flexAble	3 (1643.71)	3 (1917.29)	-	-	-	-

neous answers as if the resource limits were reached. ASPforABA dominates the ranking, ranking first in all of the subtracks DC-CO, DC-ST, DS-PR, DS-ST, SE-PR, and SE-ST. The competition between the first and second ranking solver was the tightest in the DS-PR subtrack, where the PAR-2 score of ASPforABA was approximately 1/7 of the PAR-2 score of the second-ranking solver AcbAr (156.52 vs 1120.31). We note that, hypothetically, without disqualification (and treating erroneous answers as exceeding resource limits instead), Crustabri might have ranked third in all tracks but DS-PR, where it might have ranked second—assuming that the error-producing issues in Crustabri would not affect its performance on the other benchmark instances.

9. Further Analysis of the Competition Data

Before turning to lessons learned from ICCMA 2023, recommendations for future competitions, and conclusions, we report on further analysis of the competition data.

9.1. Distribution of Positive and Negative Answers among Solved Benchmarks

We start by analyzing the balance between YES and NO answers among benchmark instances that at least one solver managed to solve. Table 7 show the YES/NO distribution for the Main and ABA tracks, together with the number of benchmark instances that were not solved by any participating solver. Note that instances in the Dynamic track have a sequence of YES and NO answers, and the solutions to instances in the Approximate track correspond to those of the Main track. For SE subtracks, "YES" means here that an extension was found, and "NO" that the nonexistence of an extension was reported. In DC and DC subtracks, the ratio between YES and NO answers was quite balanced, with some exceptions, notably DS-PR, DS-SST and DS-STG in the Main track, and DS-ST in the ABA track. In all SE subtracks except ST, an extension always exists, so all instances either have YES as an answer or are not solved. For SE-ST in the Main track, note that a stable extension exists in 206 out of 329 instances; this property can be used as a shortcut for second-level-complete SST and STG semantics. In both Main and ABA tracks, there are also more YES instances in DS-ST than in DC-ST. This is due to the fact that if a stable extension does not exist, any query is skeptically accepted.

9.2. Virtual Best Solver Performance in the Main and ABA Tracks

Table 8 (first row) shows how many instances were solved by the virtual best solver (VBS), i.e., how many instances were solved by at least one solver in each subtrack of the Main track, including the No-limits solvers. We observe that for the DC-CO and DC-ST subtracks, almost all instances are solved by the VBS (with 314/329 and 315/329 instances solved, respectively). Subtracks involving problems complete for the second level of the polynomial hierarchy are clearly harder, as witnessed by a lower number of instances solved in the DC-SST, DC-STG, DS-SST, DS-STG, SE-ID, SE-SST, and SE-STG tracks, respectively (with 287–297 instances solved out of 329, depending on the subtrack).

We also show the number of contributions of each solver to the VBS for each subtrack in Table 8 (after first row), where we define that for an instance in a subtrack, a solver contributes to the VBS if its runtime is at most 0.01 seconds less than the runtime of the VBS. Interestingly, in all subtracks, μ-TOKSIA (GLUCOSE) contributes most to the VBS (160–219 instances), followed by the No-limits solver PORTSAT on tasks involving ST semantics (137–146 instances), and μ-TOKSIA (CMSAT) in the rest of the subtracks (114-153 instances). This is in contrast to the PAR-2 ranking of the solvers in the Main track, where CRUSTABRI ranked first in nine subtracks (DC-ST, DC-STG, DS-SST, DS-ST, DS-STG, SE-PR, SE-SST, SE-ST, and SE-STG).

The full list of instances not solved in at least one subtrack of the Main track by any participating solver, together with the number of arguments and attacks in these instances, is provided in Table A.15. Note that while several relatively large crusti_g2io instances (containing 6975–89425 arguments) were not solved by any solver in the second-level DC-STG, DC-SST, DS-STG, DS-SST, SE-SST, and SE-STG subtracks, there were also a number of not-solved instances from other domains (namely Erdős-Rényi, Watts-Strogatz, and StbGenerator) which are considerably smaller (containing 301–1489 arguments). Interestingly, for the other subtracks, all crusti_g2io instances were solved by at least one solver, but there were not-solved instances within the other benchmark domains.

Table 10 shows the number of instances per subtrack that were solved by at least one solver and the contributions of each solver to the VBS in the ABA track[4]. Analogously to the Main track, fewer instances were solved in computationally harder subtracks: 376 and 377 out of 400 in DS-PR and SE-PR compared to over 380 for the other tracks. In terms of contributions, the picture is simple: the overall best-performing solver ASPforABA contributed the most to VBS (at least 346 in each subtrack), with some instances contributed by AcbAr (up to 38) and two by ASTRA. In addition, ASPforABA solved all instances with 25, 100 and 500 atoms under all semantics and all but three instances with 2000 atoms, but reached the resource limits on 118 out of 480 instances with 5000 atoms. Those three instances with 2000 atoms (an instance with 30% assumptions and $rph = rs = 10$ in DS-PR and SE-PR, and an instance with 30% assumptions and $rph = rs = 5$ in SE-PR) were also not solved by any of the other solvers, and thus they were the smallest instances that none of the solvers were able to solve.

---

[4] We include only the solvers that did not report incorrect results.

Table 7: Ratio of YES/NO answers and instances that were not solved by any solver for each subtrack of the Main and ABA tracks.

		instance count		
Track	**Subtrack**	YES	NO	not solved
Main	DC-CO	153	161	15
	DC-SST	129	168	32
	DC-ST	120	195	14
	DC-STG	201	84	44
	DS-PR	32	271	26
	DS-SST	42	245	42
	DS-ST	140	167	22
	DS-STG	42	246	41
	SE-ID	294	—	35
	SE-PR	305	—	24
	SE-SST	288	—	41
	SE-ST	206	101	22
	SE-STG	294	—	35
ABA	DC-CO	212	169	19
	DC-ST	153	230	17
	DS-PR	207	169	24
	DS-ST	317	64	19
	SE-PR	377	—	23
	SE-ST	217	164	19

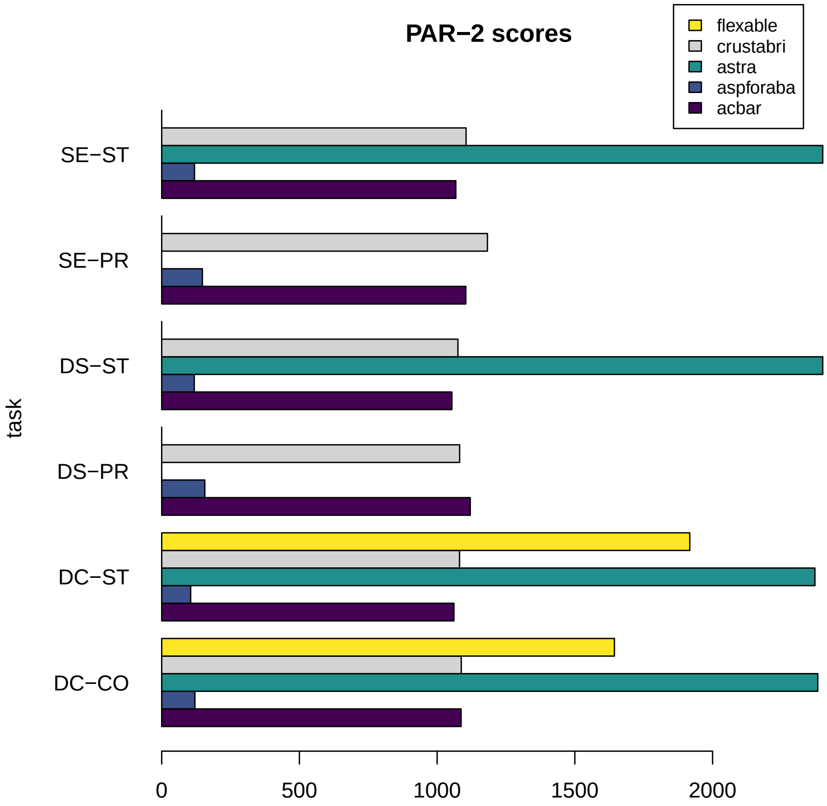


Figure 6: ABA track: PAR-2 scores. (Crustabri is disqualified due to erroneous output.)

Table 8: Number of instances solved by the virtual best solver (VBS) and the contribution of solvers to the VBS in the Main track, including No-limits solvers.

Solver	DC-CO	DC-SST	DC-ST	DC-STG	DS-PR	DS-SST	DS-ST	DS-STG	SE-ID	SE-PR	SE-SST	SE-ST	SE-STG
VBS	314	297	315	285	303	287	307	288	294	305	288	307	294
Crustabri	10	34	11	41	28	50	22	59	32	39	41	19	49
Fudge	64	69	59	92	61	67	40	62	58	58	73	35	87
μ-toksia (c)	153	135	114	142	127	122	115	129	120	117	119	118	125
μ-toksia (g)	**203**	**217**	**163**	**185**	**214**	**207**	**160**	**189**	**219**	**212**	**209**	**174**	**189**
PORTSAT	60	–	137	–	29	–	146	–	–	34	–	143	–

Table 9: Main and No-limits tracks: Number of instances solved by each solver (PAR-2-based rank in parentheses).

Solver	DC-CO	DC-SST	DC-ST	DC-STG	DS-PR	DS-SST	DS-ST	DS-STG	SE-ID	SE-PR	SE-SST	SE-ST	SE-STG
Crustabri	310 (2)	275 (3)	313 (1)	**272 (1)**	296 (2)	**284 (1)**	**303 (1)**	**285 (1)**	249 (3)	**304 (1)**	**284 (1)**	**305 (1)**	**289 (1)**
Fudge	**312 (-)**	289 (-)	**314 (-)**	264 (-)	276 (-)	263 (-)	301 (-)	276 (-)	257 (-)	280 (-)	270 (-)	**305 (-)**	285 (-)
μ-toksia (c)	306 (3)	**295 (2)**	302 (3)	268 (2)	290 (3)	279 (3)	290 (3)	271 (2)	262 (2)	288 (3)	278 (3)	292 (2)	273 (2)
μ-toksia (g)	**312 (1)**	**295 (1)**	311 (2)	262 (3)	**299 (1)**	283 (2)	296 (2)	263 (3)	**280 (1)**	299 (2)	282 (2)	297 (3)	264 (3)
PORTSAT	310 (-)	—	309 (-)	—	175 (-)	—	302 (-)	—	—	273 (-)	—	299 (-)	—

Table 10: Number of instances solved by the virtual best solver (VBS) and the contribution of solvers to the VBS in the ABA track.

Solver	DC-CO	DC-ST	DS-PR	DS-ST	SE-PR	SE-ST
VBS	381	383	376	381	377	381
AcbAr	38	35	25	30	24	24
ASPforABA	**346**	**351**	**354**	**355**	**355**	**360**
ASTRA	2	0	0	0	0	0
flexAble	0	0	0	0	0	0

Table 11: ABA track: Number of instances solved by each solver (PAR-2-based rank in parentheses).

Solver	DC-CO	DC-ST	DS-PR	DS-ST	SE-PR	SE-ST
AcbAr	221 (2)	225 (2)	217 (2)	227 (2)	219 (2)	224 (2)
ASPforABA	**381 (1)**	**383 (1)**	**376 (1)**	**381 (1)**	**377 (1)**	**381 (1)**
ASTRA	3 (4)	5 (4)	—	0 (3)	—	0 (3)
Crustabri	219 (-)	220 (-)	220 (-)	221 (-)	203 (-)	216 (-)
flexAble	131 (3)	81 (3)	—	—	—	—

9.3. Contrasting Number of Solved Instances and PAR-2 based Ranking

Next, we consider the number of solved instances by solvers in the Main track (including No-limits solvers) and the ABA track. In particular we show how the choice of the ranking scheme (PAR-2 vs number of solved instances) affects the ranking of the solvers.

Table 9 shows the number of solved instances by each solver in each Main subtrack, together with their PAR-2 based ranking in the competition given in parenthesis. We observe that the two ranking schemes would provide very similar rankings, with only a few exceptions. In particular, PORTSAT solved one instance less than Crustabri in DS-ST, whereas in terms of PAR-2 scores, PORTSAT won the subtrack. In terms of number of solved instances, Fudge and μ-toksia (cmsat) are tied in DC-CO; μ-toksia (cmsat) and μ-toksia (glucose) are tied in DC-SST; and Crustabri and Fudge are tied in SE-ST. PAR-2 scoring breaks these ties in favour of μ-toksia (cmsat) twice and in favour of Fudge once. (Note here that Fudge and PORTSAT are in the No-limits track and the other mentioned solvers in the Main track.) Table 11 shows the number of solved instances in the ABA subtracks with their PAR-2 based ranking in parentheses. Here the relative ranks of the solvers are identical to the PAR-2 ranking, with ASPforABA dominating by solving more than 375 out of the 400 instances in each subtrack.[5]

9.4. Solver Similarity in the Main Track

Figure 7 visualizes the runtime distribution of solvers in the DC-CO, DS-ST, DS-PR, and SE-ID subtracks of the Main track (including the No-limits solvers and the VBS) showing the number of instances solved (y-axis) within a given time (x-axis). Furthermore, the

---

[5] We note that the number of erroneous results Crustabri produced was 6 in DC-CO, 5 in DC-ST and DS-PR, 4 in DS-ST, 10 in SE-ST and 16 in SE-PR. These are not included as solved instances in Table 11. Treating erroneous results as timeouts, Crustabri would, similarly to under PAR-2 scoring, place third in all subtracks except DS-PR, where it would be second.

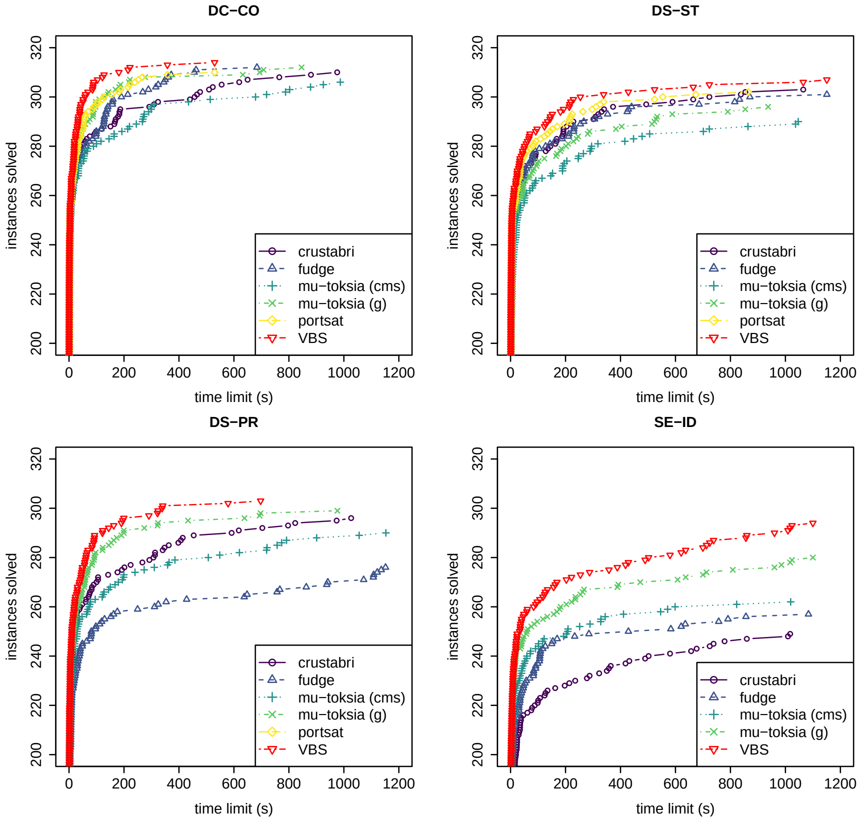
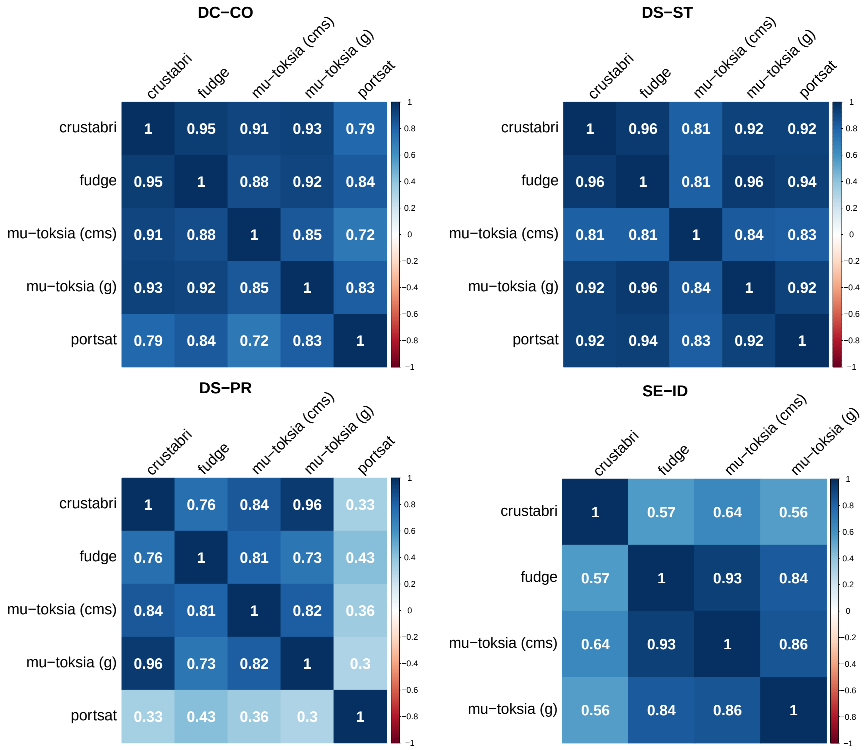


Figure 7: Number of instances solved by each solver under given a per-instance time limit in the DC-CO, DS-ST, DS-PR, and SE-ID subtracks of the Main track, including No-limits solvers.

pairwise Pearson correlation coefficients for solver runtimes are visualized in Figure 8. In the DC-CO and DS-ST subtracks, the runtime distributions of all solvers are similar, and the correlations are significantly high. This can be explained by the fact that all solvers are SAT-based, employing similar SAT encoding and solver techniques. Interestingly, in the DS-PR and SE-ID subtracks, the runtime distributions of solvers are more different and runtime correlations lower. Furthermore, in the SE-ID subtrack the VBS outperforms the winning solver by a large margin. Therefore it seems that in these tracks the solvers are internally different, which also suggests that investigating these differences and combining the strengths of different approaches might lead to further improved solvers. Runtime distributions and pairwise correlations for all other subtracks of the Main track are provided in Appendix A, Figures A.9 and A.10, respectively.

9.5. False Positives and Negatives in the Approximate Track

We move on to analyse the Approximate track data on how often the participating solvers provided wrong answers, which is a particular feature of this track. Table 12 provides statistics on the frequency of true and false positive and negative answers reported by the participating solver. The same statistics are shown for each subtrack separately in Table 13. Specifically, we report the number of true positive (correct solution is YES and solver reports YES), false positive (correct solution is NO and solver reports YES), true negative (correct solution is



DC−CO

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)	portsat
crustabri	1	0.95	0.91	0.93	0.79
fudge	0.95	1	0.88	0.92	0.84
mu−toksia (cms)	0.91	0.88	1	0.85	0.72
mu−toksia (g)	0.93	0.92	0.85	1	0.83
portsat	0.79	0.84	0.72	0.83	1

DS−ST

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)	portsat
crustabri	1	0.96	0.81	0.92	0.92
fudge	0.96	1	0.81	0.96	0.94
mu−toksia (cms)	0.81	0.81	1	0.84	0.83
mu−toksia (g)	0.92	0.96	0.84	1	0.92
portsat	0.92	0.94	0.83	0.92	1

DS−PR

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)	portsat
crustabri	1	0.76	0.84	0.96	0.33
fudge	0.76	1	0.81	0.73	0.43
mu−toksia (cms)	0.84	0.81	1	0.82	0.36
mu−toksia (g)	0.96	0.73	0.82	1	0.3
portsat	0.33	0.43	0.36	0.3	1

SE−ID

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.57	0.64	0.56
fudge	0.57	1	0.93	0.84
mu−toksia (cms)	0.64	0.93	1	0.86
mu−toksia (g)	0.56	0.84	0.86	1

Figure 8: Pairwise Pearson correlation coefficients of solver runtimes in the DC-CO, DS-ST, DS-PR, and SE-ID subtracks of the Main track, including No-limits solvers.

NO and solver reports NO) and false negative (correct solution is YES and solver reports NO) answers. We refer to these as TP, FP, TN and FN, respectively. We moreover refer to the number of instances for which the solution is known to be YES (respectively, NO) as P (respectively, N). In addition to the raw numbers, we report the rate of correct positive answers over all instances for which the answer is known to be positive (TP/P), and similarly for negative answers (TN/N). These measures reflect how likely a solver is to report the correct answer on an instance, as opposed to either reporting the wrong answer or not reporting an answer. We also report the rate of correct YES answers over the instances that the given solver gave an answer on (TP/(TP+FN)), and similarly for NO answers (TN/(TN+FP)). These reflect the likelihood that an answer output by a solver is correct, given that the solver has output an answer.

HARPER++ is the only solver for which TP+FN equals P and TN+FP equals N, indicating that it output an answer for each instance (that any solver in the Main track gave an answer on). This likely had an effect on its success—indeed, HARPER++ ranked first in most subtracks—as its rate of correct NO answers was relatively low when comparing to TN+FP, but high when comparing to N. Additionally, HARPER++ has clearly the best true positive rate, with the highest rate on both true positive measures. For all solvers other than HARPER++, the true positive rates are significantly lower than the true negative rates, indicating a bias towards answering NO.

Table 12: True positive (TP), false positive (FP), true negative (TN) and false negative (FN), as well as rates of correct answers over all instances with known correct solutions (N and P) or all instances that the solver gave an answer on (TP+FN and TN+FP) in Approximate track per solver over all subtracks.

Solver	TP/P	TP/(TP+FN)	TN/N	TN/(TN+FP)	TP	FP	TN	FN
AFGCN v2	0.53	0.68	0.75	0.91	472	137	1350	221
ARIPOTER-Degrees	0.51	0.60	0.83	0.93	451	108	1495	295
ARIPOTER-HCAT	0.61	0.77	0.77	0.92	542	127	1393	166
fargo-limited	0.73	0.78	0.90	0.98	655	25	1627	180
harper++	0.84	0.84	0.81	0.81	747	348	1450	145

Table 13: True positive (TP), false positive (FP), true negative (TN) and false negative (FN), as well as rates of correct answers over all instances with known correct answer (N and P) or all instances that the solver gave an answer on (TP+FN and TN+FP) in Approximate track per subtrack and solver.

Subtrack	**Solver**	TP/P	TP/(TP+FN)	TN/N	TN/(TN+FP)	TP	FP	TN	FN
DC-CO	AFGCN v2	0.48	0.70	0.74	0.83	73	24	119	32
	ARIPOTER-Degrees	0.15	0.20	0.96	1.00	23	0	154	94
	ARIPOTER-HCAT	0.41	0.57	0.88	0.97	62	4	142	47
	fargo-limited	0.84	0.98	0.96	1.00	129	0	154	3
	harper++	1.00	1.00	0.42	0.42	153	94	67	0
DC-ID	AFGCN v2	0.76	0.86	0.85	1.00	25	0	221	4
	ARIPOTER-Degrees	0.70	0.85	0.87	1.00	23	0	228	4
	ARIPOTER-HCAT	0.70	0.85	0.82	1.00	23	0	214	4
	fargo-limited	1.00	1.00	0.90	1.00	33	0	235	0
	harper++	0.88	0.88	1.00	1.00	29	0	261	4
DC-SST	AFGCN v2	0.57	0.70	0.70	0.81	73	27	118	31
	ARIPOTER-Degrees	0.18	0.21	0.94	1.00	23	0	158	87
	ARIPOTER-HCAT	0.48	0.58	0.87	0.97	62	4	146	44
	fargo-limited	0.95	0.97	0.92	0.96	123	6	154	4
	harper++	1.00	1.00	0.40	0.40	129	101	67	0
DC-ST	AFGCN v2	0.54	0.68	0.64	0.82	65	27	125	31
	ARIPOTER-Degrees	0.18	0.21	0.87	0.99	21	2	169	80
	ARIPOTER-HCAT	0.48	0.58	0.76	0.94	57	9	149	41
	fargo-limited	0.97	0.97	0.79	0.92	116	13	155	3
	harper++	1.00	1.00	0.34	0.34	120	128	67	0
DC-STG	AFGCN v2	0.61	0.75	0.50	0.57	122	32	42	41
	ARIPOTER-Degrees	0.88	0.99	0.65	0.72	177	21	55	1
	ARIPOTER-HCAT	0.83	0.99	0.65	0.72	167	21	55	1
	fargo-limited	0.61	0.64	0.90	0.93	123	6	76	70
	harper++	1.00	1.00	0.70	0.70	200	25	59	1
DS-PR	AFGCN v2	0.78	0.89	0.75	0.98	25	5	203	3
	ARIPOTER-Degrees	0.72	0.88	0.86	1.00	23	0	234	3
	ARIPOTER-HCAT	0.72	0.88	0.80	1.00	23	0	218	3
	fargo-limited	1.00	1.00	0.88	1.00	32	0	239	0
	harper++	0.91	0.91	1.00	1.00	29	0	271	3
DS-SST	AFGCN v2	0.69	0.88	0.80	0.97	29	7	195	4
	ARIPOTER-Degrees	0.55	0.68	0.89	1.00	23	0	219	11
	ARIPOTER-HCAT	0.55	0.68	0.85	1.00	23	0	208	11
	fargo-limited	0.79	0.79	0.93	1.00	33	0	227	9
	harper++	0.69	0.69	1.00	1.00	29	0	245	13
DS-ST	AFGCN v2	0.21	0.29	0.80	0.96	30	6	133	72
	ARIPOTER-Degrees	0.82	0.97	0.36	0.41	115	85	60	4
	ARIPOTER-HCAT	0.73	0.96	0.32	0.37	102	89	53	4
	fargo-limited	0.24	0.29	0.96	1.00	33	0	160	82
	harper++	0.21	0.21	1.00	1.00	29	0	167	111
DS-STG	AFGCN v2	0.71	0.91	0.79	0.96	30	9	194	3
	ARIPOTER-Degrees	0.55	0.68	0.89	1.00	23	0	218	11
	ARIPOTER-HCAT	0.55	0.68	0.85	1.00	23	0	208	11
	fargo-limited	0.79	0.79	0.92	1.00	33	0	227	9
	harper++	0.69	0.69	1.00	1.00	29	0	246	13

9.6. Impact of Benchmark Parameters on Solver Performance in ABA Track

We next consider the impact that parameter values used for generating the ABA track benchmarks had on solver performance. Recall that, in contrast to the more heterogeneous sets of abstract argumentation frameworks standardly employed in the Main track, benchmarks for the ABA track were in this first instantiation of the ABA track generated with a simple random instance generator as detailed in Section 6.3. While this makes the ABA track benchmark set less heterogeneous, it on the other hand allows for a more fine-grained investigation into the impact of generator parameter values on the performance of individual solvers.

Table 14 provides the PAR-2 scores and numbers of solved instances (in parentheses) for different combinations of the benchmark parameters (i) maximum number of rules per head (*rph*, i.e. rules per non-assumption atom) and (ii) maximum rule size (*rs*). The data for the different subtracks turned out to be quite similar, and hence for simplicity we here focus on DC-CO.

Interestingly, the solvers behave somewhat differently with respect to their performance on these parameter families. Instances with $rph = 10, rs = 5$ and $rph = 5, rs = 10$ are easier to solve for ASPforABA than instances arising from using the other benchmark parameter combinations. The combination $rph = 5, rs = 10$ results in instances that were easier compared to other combinations for all solvers, but instances resulting from the combination $rph = 10, rs = 5$ turned out to be hard to solve for the other solvers than ASPforABA.

Furthermore, we note that a greater proportion of assumptions out of all atoms made instances harder to solve for each of the participating solvers. ASPforABA solved 1191 of the 1200 instances (across all subtracks) with 10% assumptions, but only 1088 of the instances with 30% assumptions. For Crustabri the corresponding numbers are 827 and 472; for flexABle 116 and 96; and for AcbAr 673 and 660. The number of atoms, as can be expected as the primary parameter for scaling the size of the instances, also had a very significant impact on runtimes (recall Section 9.2).

10. Further Discussion, Lessons Learned and Recommendations for Future Competitions

Finally, we discuss some of the lessons learned from organizing ICCMA 2023 and further observations.

10.1. New Developments and Potential Ideas for New/Revised Competition Tracks

The 2023 instantiation of ICCMA brought on several new developments. One major aspect was the **ABA track**, which came to fruition for the first time by drawing in a necessary number of solver submissions. Due to several recent developments in practical algorithms for reasoning in structured argumentation formalisms [126, 127, 70, 71, 128, 129, 72, 130, 131, 132, 73, 74, 75, 76, 77, 78], we hope that future ICCMA instantiations will also feature a track (or even several tracks) focusing on reasoning in structured formalisms.

Another development were **changes to the input/output formats**, moving to a single, more compact numerical format. As pointed out earlier, the proposal for this change came from the community and was motivated by the fact that essentially all argumentation solvers in any case need to internally indexing the building blocks of argumentation frameworks (arguments, attacks, etc.), and providing this already at input allows solvers to directly employ the input indexing. While such a change might have potentially discouraged submitting already existing solvers to the competition, the organizers viewed this change worthwhile to make as it also only required quite minor changes restricted to the input processing routines of existing solvers. Furthermore, no complaints on making this change were received from

Table 14: PAR-2 scores and number of solved instances under different parameters in the DC-CO subtrack of the ABA track.

		PAR-2 score (#solved)			
Solver	Subtrack	$rph = 10, rs = 10$	$rph = 10, rs = 5$	$rph = 5, rs = 10$	$rph = 5, rs = 5$
DC-CO	AcbAr	1166.89 (52)	1160.62 (52)	968.20 (60)	1052.48 (57)
	ASPforABA	201.00 (92)	4.28 (100)	1.56 (100)	275.61 (89)
	ASTRA	2376.00 (1)	2376.00 (1)	2400 (0)	2376.00 (1)
	Crustabri	1032.95 (57)	2137.43 (11)	25.03 (99)	1155.20 (52)
	flexAble	1686.16 (30)	2026.56 (16)	1137.31 (56)	1724.81 (29)
DC-ST	AcbAr	1094.4 (55)	1159.8 (52)	968.4 (60)	1021.0 (58)
	ASPforABA	196.4 (92)	2.3 (100)	1.4 (100)	220.3 (91)
	ASTRA	2376.0 (1)	2376.0 (1)	2358.8 (2)	2376.0 (1)
	Crustabri	1033.2 (57)	2065.3 (14)	25.0 (99)	1202.1 (50)
	flexAble	1921.0 (20)	2026.8 (16)	1800.8 (25)	1920.5 (20)
DS-PR	AcbAr	1197.4 (51)	1213.1 (51)	968.0 (60)	1102.8 (55)
	ASPforABA	270.7 (89)	29.0 (99)	1.5 (100)	324.9 (88)
	ASTRA	—	—	—	—
	Crustabri	1032.8 (57)	2113.5 (12)	25.0 (99)	1155.3 (52)
	flexAble	—	—	—	—
DS-ST	AcbAr	1081.4 (56)	1145.4 (53)	968.4 (60)	1019.4 (58)
	ASPforABA	247.1 (90)	2.4 (100)	1.3 (100)	222.1 (91)
	ASTRA	2400 (0)	2400 (0)	2400 (0)	2400 (0)
	Crustabri	1033.1 (57)	2089.3 (13)	24.9 (99)	1154.7 (52)
	flexAble	—	—	—	—
SE-PR	AcbAr	1186.4 (51)	1176.1 (52)	967.9 (60)	1086.4 (56)
	ASPforABA	271.0 (89)	3.3 (100)	1.6 (100)	315.3 (88)
	ASTRA	—	—	—	—
	Crustabri	1128.7 (53)	2184.1 (9)	25.0 (99)	1392.9 (42)
	flexAble	—	—	—	—
SE-ST	AcbAr	1116.9 (54)	1160.9 (52)	968.6 (60)	1025.2 (58)
	ASPforABA	247.0 (90)	2.4 (100)	1.4 (100)	225.7 (91)
	ASTRA	2400.0 (0)	2400.0 (0)	2400.0 (0)	2400.0 (0)
	Crustabri	1033.3 (57)	2137.3 (11)	25.0 (99)	1224.8 (49)
	flexAble	—	—	—	—

the community. With these considerations, we would recommend keeping the now-introduced numerical format also for forthcoming ICCMA instantiations.

A further input/output related change was the introduction of the **IPAFAIR interface for the Dynamic track**, resulting in adjusting the specification of the track so that changes to the argumentation framework were communicated to the solver iteratively rather than at initialization. It should be noted that the current Dynamic track focuses on a very specific form of dynamics. A wide range of different types of dynamics in argumentation—both in abstract [133, 134, 135, 47, 136, 137, 49, 138, 139, 52] and structured formalisms [140, 141, 142, 143, 144, 145, 146]—has recently received considerable attention. For future ICCMA instantiations, it might be interesting to consider other specific types of dynamics also as the basis of a Dynamic track in ICCMA, and potentially also in structured argumentations formalisms. More generally, considering new challenging computational tasks, in addition to the more classical skeptical and credulous decision problems, has the potential of keeping ICCMA vibrant and forward-looking.

A major development in 2023 was the introduction of **witness checking**. In particular, all "positive" witnesses (i.e., witnessing extensions reported by solvers for credulous acceptance, witnessing counterexample-extensions reported by solvers for skeptical acceptance, as will as reported witnesses for the problem of finding a single extension) in the main track were checked. We find this an important development towards ensuring the correctness of implementations of argumentation solvers. For future ICCMA instantiations, we believe witness checking should also be introduced for the ABA (or similar structured argumentation) track as well. For ICCMA 2023, we did not enforce witness checking in the ABA track because our main goal was to realize the track for the first time. A further non-trivial next step in potential future ICCMA competitions would be to introduce ways of checking "negative" answers (NO for credulous acceptance, YES for skeptical acceptance) reported by solvers. By standard complexity assumptions, however, no short witnesses exist in these cases. Thus such an extension would require the development of proof certificates and proof checkers, in analogy to e.g. recent developments in the realm of SAT solving [94].

To allow for separately evaluating sequential solvers and solvers building on top of sequential solvers e.g. by combining different existing solvers in portfolio-style techniques, solvers employing parallel computations via the use of multiple processor cores, as well as solvers which will not be made available in open source were invited to a special **No-limits track** which consists of the same subtracks as the Main track. Only two solvers turned out to fit the No-limits description, due to making use of parallel computations. The performance gains for these solvers when compared to Main track solvers were relatively modest. We believe there may be various reasons for this. It is well-acknowledged e.g. in the realm of parallel SAT solving that it can be surprisingly difficult at times to obtain massive gains from non-trivial parallelization of solvers. Regarding portfolios, it may be the case that the relatively high similarity of current argumentation solvers (as empirically observed and discussed in the article) hinders making large performance gains through portfolios. As a further consideration, to our best understanding there are only relatively few works so far (including [147, 148, 149, 150, 151, 152]) on developing highly effective parallel or portfolio solvers for argumentation, and there could be further potential that could be harnessed in the future. With this in mind, we would recommend future ICCMA organizers to consider organizing special tracks specifically for parallel argumentation solvers in order to more clearly encourage pushing the state of the art in parallel approaches to argumentative reasoning forward.

10.2. Similarity of Main Track Solvers

The use of SAT solvers appears to be—at least currently—the dominating approach to developing systems to reasoning in abstract argumentation. More generally, declarative approaches (based on SAT or ASP) appear to be dominating in all tracks except for the Approximate track. On one hand, the identification of the success of the declarative approaches for argumentative reasoning is something to be celebrated. On the other hand, in particular in the Main and Dynamic tracks, the solvers mostly implement very similar ideas, relying on SAT solvers, to the extent that it is not entirely clear whether the somewhat limited performance differences between the AF solvers is more due to the choice of the underlying SAT solver; it should be noted that there has already been some work on the impact of the choice of SAT solving techniques of the efficiency of SAT-based argumentation solvers [153, 154]. For future competitions, it would be worthwhile to consider whether a specific SAT solver should be enforced to be used by the competition organizers, potentially via offering an API to interface with a pre-determined SAT solver. This would allow for a more scientific evaluation of the actual algorithmic ideas each AF solver is based on, discounting the impact of the choice of a SAT solver. Naturally, such a decision should be made in discussion with the community. Furthermore, the development of non-SAT-based AF solvers should be encouraged to ensure algorithmic diversity.

10.3. Diversity of Competition Benchmarks

The number of new benchmarks and benchmark generators submitted to ICCMA 2023 was markedly low. Notably, the ABA track—realized for the first time—received no benchmark submissions and so the benchmarks were generated by a single random generation model implemented by the organizers. Random general models can be considered interesting due to allowing for a tight control over the parameter space of generated benchmark instances. However, we consider it increasingly important for the argumentation community at large to develop, generate and submit benchmarks arising from different real-world use cases of argumentative reasoning to the ICCMA competition, especially benchmarks which would be at the same time challenging for current state-of-the-art argumentation solvers. This would provide an avenue for showcasing the practical importance of developing increasingly capable argumentation solvers and motivate organizing future instantiations of ICCMA. New and diverse benchmark instances are also important both for the ICCMA competitions and for subsequent use in research works, in order to avoid potential overfitting of solver techniques to solve a relatively fixed and limited set of benchmarks.

11. Conclusions

In this article we provided a comprehensive overview of the 2023 ICCMA competition, the 5th instantiation of the series of International Competition on Computational Models of Argumentation. We explained new changes to the competitions, including revised input-output formats, the IPAFAIR API for the Dynamic track, the new structured argumentation track, and witness checking. We gave a description of the generation and selection of benchmarks and an overview of the solvers that participated in the competition. We detailed the results of the competition with additional analysis of the empirical data obtained from the competition. Furthermore, we discussed some of the key lessons learned from organizing ICCMA 2023 with potential considerations for future instantiations of the competition.

Acknowledgements

This work has been financially supported by Research Council of Finland (under grants 347588 and 356046), University of Helsinki Doctoral Programme in Computer Science and Helsinki Institute for Information Technology HIIT. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources. Finally, the authors thank everyone who participated in or otherwise contributed to ICCMA 2023, as well as the ICCMA Steering Committee for their support.

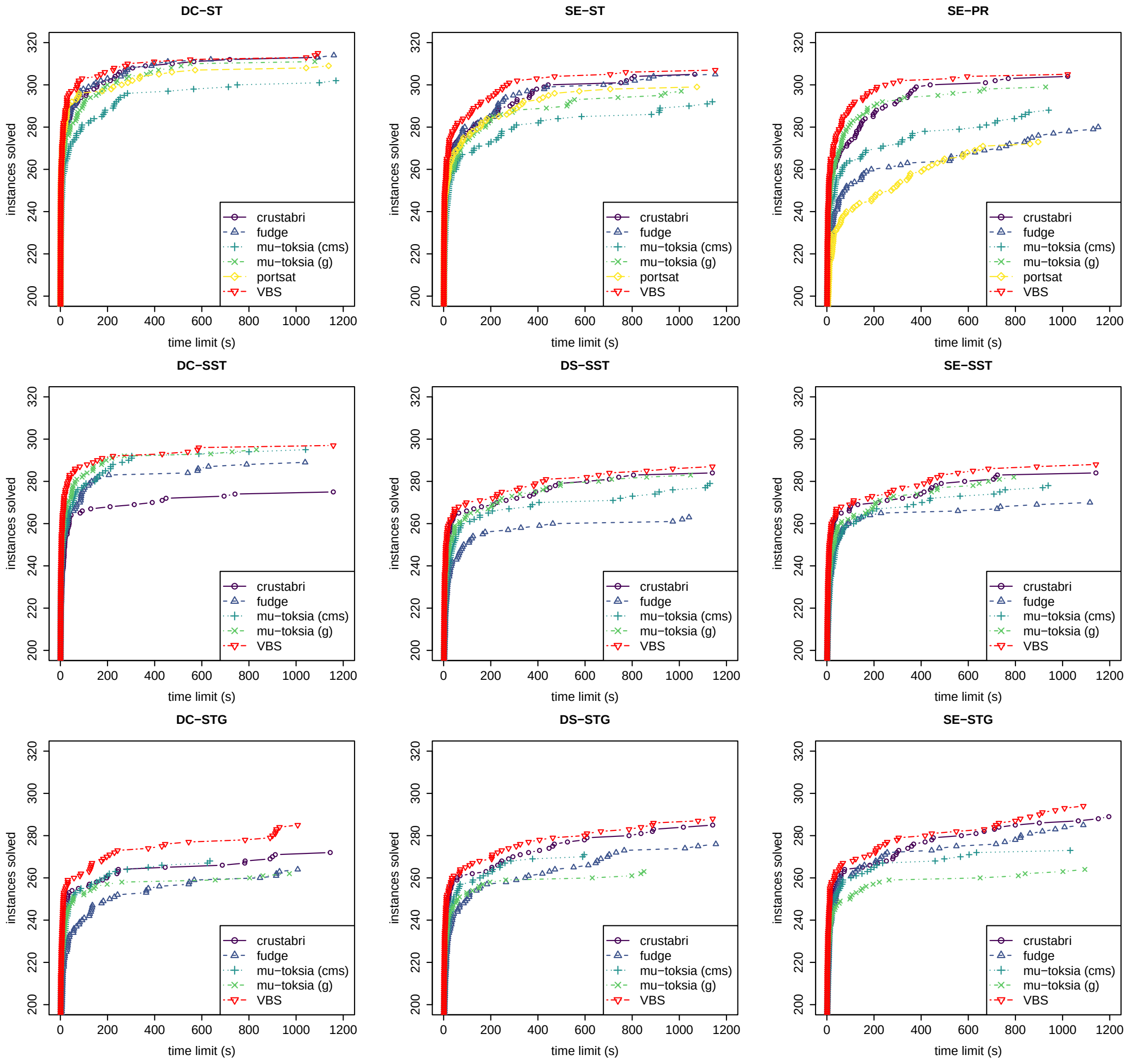
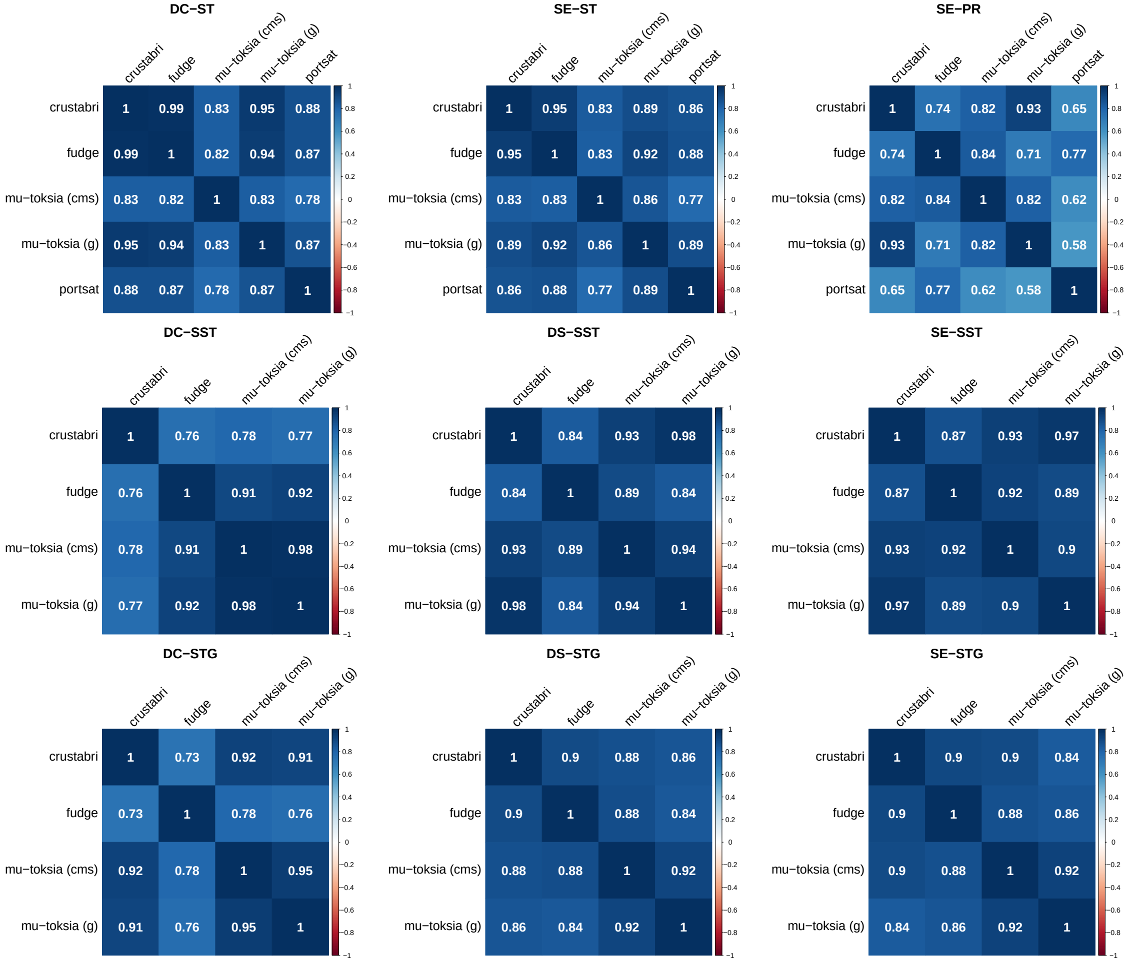


Figure A.9: Number of instances solved by a participating solver given a per-instance time limit for DC-ST, SE-ST, SE-PR, DC-SST, DS-SST, SE-SST, DC-STG, DS-STG, and SE-STG subtracks of the Main and No-limits tracks.

Appendix A. Additional Empirical Data

Figure A.9 visualizes the runtime distributions of solvers in the DC-ST, SE-ST, SE-PR, DC-SST, DS-SST, SE-SST, DC-STG, DS-STG, and SE-STG subtracks of the Main and No-limits tracks. Pairwise correlation coefficients for solver runtimes are visualized in Figure A.10.



DC−ST

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)	portsat
crustabri	1	0.99	0.83	0.95	0.88
fudge	0.99	1	0.82	0.94	0.87
mu−toksia (cms)	0.83	0.82	1	0.83	0.78
mu−toksia (g)	0.95	0.94	0.83	1	0.87
portsat	0.88	0.87	0.78	0.87	1

SE−ST

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)	portsat
crustabri	1	0.95	0.83	0.89	0.86
fudge	0.95	1	0.83	0.92	0.88
mu−toksia (cms)	0.83	0.83	1	0.86	0.77
mu−toksia (g)	0.89	0.92	0.86	1	0.89
portsat	0.86	0.88	0.77	0.89	1

SE−PR

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)	portsat
crustabri	1	0.74	0.82	0.93	0.65
fudge	0.74	1	0.84	0.71	0.77
mu−toksia (cms)	0.82	0.84	1	0.82	0.62
mu−toksia (g)	0.93	0.71	0.82	1	0.58
portsat	0.65	0.77	0.62	0.58	1

DC−SST

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.76	0.78	0.77
fudge	0.76	1	0.91	0.92
mu−toksia (cms)	0.78	0.91	1	0.98
mu−toksia (g)	0.77	0.92	0.98	1

DS−SST

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.84	0.93	0.98
fudge	0.84	1	0.89	0.84
mu−toksia (cms)	0.93	0.89	1	0.94
mu−toksia (g)	0.98	0.84	0.94	1

SE−SST

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.87	0.93	0.97
fudge	0.87	1	0.92	0.89
mu−toksia (cms)	0.93	0.92	1	0.9
mu−toksia (g)	0.97	0.89	0.9	1

DC−STG

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.73	0.92	0.91
fudge	0.73	1	0.78	0.76
mu−toksia (cms)	0.92	0.78	1	0.95
mu−toksia (g)	0.91	0.76	0.95	1

DS−STG

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.9	0.88	0.86
fudge	0.9	1	0.88	0.84
mu−toksia (cms)	0.88	0.88	1	0.92
mu−toksia (g)	0.86	0.84	0.92	1

SE−STG

	crustabri	fudge	mu−toksia (cms)	mu−toksia (g)
crustabri	1	0.9	0.9	0.84
fudge	0.9	1	0.88	0.86
mu−toksia (cms)	0.9	0.88	1	0.92
mu−toksia (g)	0.84	0.86	0.92	1

Figure A.10: Pairwise Pearson correlation coefficients of solving times for DC-ST, SE-ST, SE-PR, DC-SST, DS-SST, SE-SST, DC-STG, DS-STG, and SE-STG subtracks of the Main and No-limits tracks.

instance	$\|A\|$	$\|R\|$	DC-CO	DC-SST	DC-ST	DC-STG	DS-PR	DS-SST	DS-ST	DS-STG	SE-ID	SE-PR	SE-SST	SE-ST	SE-STG
ER_300_10_9.af	301	4692						✗							
ER_300_20_2.af	301	9196					✗	✗		✗	✗		✗		✗
ER_400_20_9.af	401	16493	✗	✗		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ER_400_30_9.af	401	24236				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ER_400_40_7.af	401	32676					✗	✗		✗	✗	✗	✗		
WS_400_24_50_10.af	400	5200				✗		✗		✗	✗		✗		✗
WS_400_32_70_70.af	400	6800	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
WS_500_16_50_50.af	500	4500								✗					✗
WS_500_16_70_50.af	500	4500				✗					✗				
crusti_g2io_175_0.2_511_10.af	89425	6234732		✗		✗									
crusti_g2io_175_0.2_511_13.af	89425	6233136		✗		✗									
crusti_g2io_175_0.2_511_32.af	89425	6235617		✗		✗				✗					
crusti_g2io_175_0.2_511_36.af	89425	6238768				✗									
crusti_g2io_175_0.2_511_48.af	89425	6238580		✗		✗				✗					
crusti_g2io_200_0.1_127_12.af	25400	1512884				✗				✗					✗
crusti_g2io_200_0.1_127_19.af	25400	1514482		✗		✗		✗		✗			✗		✗
crusti_g2io_200_0.1_127_38.af	25400	1513686		✗		✗		✗		✗			✗		✗
crusti_g2io_200_0.1_127_46.af	25400	1512941				✗		✗		✗	✗		✗		✗
crusti_g2io_200_0.1_127_6.af	25400	1511839		✗		✗		✗		✗	✗		✗		✗
crusti_g2io_200_0.1_127_8.af	25400	1514585		✗		✗		✗		✗			✗		✗
crusti_g2io_225_0.1_31_25.af	6975	459475				✗		✗		✗			✗		
crusti_g2io_225_0.2_127_41.af	28575	2553211		✗		✗									
crusti_g2io_250_0.2_255_12.af	63750	6353317		✗		✗		✗					✗		✗
crusti_g2io_250_0.2_255_15.af	63750	6347738		✗		✗		✗		✗			✗		
crusti_g2io_250_0.2_255_18.af	63750	6350676		✗				✗					✗		
crusti_g2io_250_0.2_255_31.af	63750	6350548		✗		✗		✗		✗			✗		
crusti_g2io_250_0.2_255_43.af	63750	6349882		✗				✗					✗		
crusti_g2io_300_0.2_255_11.af	76500	9143840		✗		✗		✗		✗	✗		✗		
crusti_g2io_300_0.2_255_17.af	76500	9146812		✗		✗		✗		✗	✗		✗		✗
crusti_g2io_300_0.2_255_26.af	76500	9143633		✗		✗		✗		✗	✗		✗		✗
crusti_g2io_350_0.5_255_40.af	89250	21800790		✗				✗			✗		✗		
st_1015_36_27_3440.af	1015	18721	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1037_97_34_647.af	1037	49883	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1230_64_16_373.af	1230	39043	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1234_99_25_3756.af	1234	59468				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1244_87_28_3669.af	1244	52974				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1276_43_25_1938.af	1276	28248				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1350_55_31_149.af	1350	37677	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1352_53_23_3737.af	1352	35891	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1391_70_12_1674.af	1391	49069	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1400_85_28_2113.af	1400	59392	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1412_95_15_3263.af	1412	66072				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_1489_41_39_1070.af	1489	31575	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_496_75_39_1354.af	496	18048									✗				
st_521_43_14_3157.af	521	11104									✗				
st_659_37_25_686.af	659	12492	✗		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_704_68_9_3183.af	704	23216				✗	✗	✗		✗	✗	✗	✗		✗
st_815_74_9_2860.af	815	29439				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_826_34_8_3910.af	826	14306	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_883_28_16_1144.af	883	13083	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_890_86_9_572.af	890	38015	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_902_67_36_2711.af	902	29682				✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
st_955_26_12_3941.af	955	12938	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
stanislaus_ca_2015-12-02.gml.20.af	319	816					✗								

Table A.15: Instances not solved in at least one subtrack of the Main track by any solver (marked by ✗ for each subtrack), and the number of arguments $|A|$ and attacks $|R|$ in each instance.

References

[1] K. Atkinson, P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G. R. Simari, M. Thimm, S. Villata, Towards artificial argumentation, AI Mag. 38 (3) (2017) 25–36.

[2] P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018.

[3] H. Prakken, Historical overview of formal argumentation, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 2, pp. 73–141.

[4] T. J. M. Bench-Capon, P. E. Dunne, Argumentation in artificial intelligence, Artif. Intell. 171 (10-15) (2007) 619–641.

[5] T. J. M. Bench-Capon, Argument in artificial intelligence and law, Artif. Intell. Law 5 (4) (1997) 249–261.

[6] D. N. Walton, Argumentation methods for artificial intelligence in law, Springer, 2005.

[7] T. J. M. Bench-Capon, H. Prakken, G. Sartor, Argumentation in legal reasoning, in: G. R. Simari, I. Rahwan (Eds.), Argumentation in Artificial Intelligence, Springer, 2009, pp. 363–382.

[8] H. Prakken, G. Sartor, Law and logic: A review from an argumentation perspective, Artif. Intell. 227 (2015) 214–245.

[9] H. Prakken, A. Z. Wyner, T. J. M. Bench-Capon, K. Atkinson, A formalization of argumentation schemes for legal case-based reasoning in ASPIC+, J. Log. Comput. 25 (5) (2015) 1141–1166.

[10] T. J. Bench-Capon, Representation of case law as an argumentation framework, Proc. Jurix (2002) 103–112.

[11] T. J. M. Bench-Capon, Before and after Dung: Argumentation in AI and law, Argument Comput. 11 (1-2) (2020) 221–238.

[12] J. Domínguez, D. Prociuk, B. Marović, K. Čyras, O. Cocarascu, F. Ruiz, E. Mi, E. Mi, C. Ramtale, A. Rago, A. Darzi, F. Toni, V. Curcin, B. Delaney, ROAD2H: Development and evaluation of an open-source explainable artificial intelligence approach for managing co-morbidity and clinical guidelines, Learn. Health Syst. (2023).

[13] R. Craven, F. Toni, C. Cadar, A. Hadad, M. Williams, Efficient argumentation for medical decision-making, in: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), Proc. KR, AAAI Press, 2012, pp. 598–602.

[14] K. Cyras, T. Oliveira, A. Karamlou, F. Toni, Assumption-based argumentation with preferences and goals for patient-centric reasoning with interacting clinical guidelines, Argument Comput. 12 (2) (2021) 149–189.

[15] Z. Zeng, Z. Shen, J. J. Chin, C. Leung, Y. Wang, Y. Chi, C. Miao, Explainable and contextual preferences based decision making with assumption-based argumentation for diagnostics and prognostics of alzheimer's disease, in: A. E. F. Seghrouchni, G. Sukthankar, B. An, N. Yorke-Smith (Eds.), Proc. AAMAS, IFAAMAS, 2020, pp. 2071–2073.

[16] A. Hunter, M. Williams, Aggregating evidence about the positive and negative effects of treatments, Artif. Intell. Medicine 56 (3) (2012) 173–190.

[17] I. Sassoon, N. Kökciyan, S. Modgil, S. Parsons, Argumentation schemes for clinical decision support, Argument Comput. 12 (3) (2021) 329–355.

[18] M. Chapman, P. Balatsoukas, M. Ashworth, V. Curcin, N. Kökciyan, K. Essers, I. Sassoon, S. Modgil, S. Parsons, E. I. Sklar, Computational argumentation-based clinical decision support, in: E. Elkind, M. Veloso, N. Agmon, M. E. Taylor (Eds.), Proc. AAMAS, IFAAMAS, 2019, pp. 2345–2347.

[19] K. Atkinson, T. J. M. Bench-Capon, S. Modgil, Argumentation for decision support, in: S. Bressan, J. Küng, R. R. Wagner (Eds.), Proc. DEXA, Vol. 4080 of LNCS, Springer, 2006, pp. 822–831.

[20] A. Rago, F. Toni, M. Aurisicchio, P. Baroni, Discontinuity-free decision support with quantitative argumentation debates, in: C. Baral, J. P. Delgrande, F. Wolter (Eds.), Proc. KR, AAAI Press, 2016, pp. 63–73.

[21] A. Rago, O. Cocarascu, F. Toni, Argumentation-based recommendations: Fantastic explanations and how to find them, in: J. Lang (Ed.), Proc. IJCAI, ijcai.org, 2018, pp. 1949–1955.

[22] Q. Zhong, X. Fan, X. Luo, F. Toni, An explainable multi-attribute decision model based on argumentation, Expert Syst. Appl. 117 (2019) 42–61.

[23] Á. Carrera, C. A. Iglesias, A systematic review of argumentation techniques for multi-agent systems research, Artif. Intell. Rev. 44 (4) (2015) 509–535.

[24] X. Fan, F. Toni, A. Mocanu, M. Williams, Dialogical two-agent decision making with assumption-based argumentation, in: A. L. C. Bazzan, M. N. Huhns, A. Lomuscio, P. Scerri (Eds.), Proc. AAMAS, IFAAMAS/ACM, 2014, pp. 533–540.

[25] A. R. Panisson, P. McBurney, R. H. Bordini, A computational model of argumentation schemes for multi-agent systems, Argument Comput. 12 (3) (2021) 357–395.

[26] C. da Costa Pereira, B. Liao, A. Malerba, A. Rotolo, A. G. B. Tettamanzi, L. W. N. van der Torre, S. Villata, Handling norms in multi-agent systems by means of formal argumentation, FLAP 4 (9) (2017) 3039–3073.

[27] K. Cyras, A. Rago, E. Albini, P. Baroni, F. Toni, Argumentative XAI: A survey, in: Z. Zhou (Ed.), Proc. IJCAI, ijcai.org, 2021, pp. 4392–4399.

[28] A. Vassiliades, N. Bassiliades, T. Patkos, Argumentation and explainable artificial intelligence: a survey, Knowl. Eng. Rev. 36 (2021) e5.

[29] K. Cyras, A. Karamlou, M. Lee, D. Letsios, R. Misener, F. Toni, AI-assisted schedule explainer for nurse rostering, in: A. E. F. Seghrouchni, G. Sukthankar, B. An, N. Yorke-Smith (Eds.), Proc. AAMAS, IFAAMAS, 2020, pp. 2101–2103.

[30] A. Rago, O. Cocarascu, C. Bechlivanidis, D. A. Lagnado, F. Toni, Argumentative explanations for interactive recommendations, Artif. Intell. 296 (2021) 103506.

[31] N. Potyka, Interpreting neural networks as quantitative argumentation frameworks, in: Proc. AAAI, AAAI Press, 2021, pp. 6463–6470.

[32] 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.

[33] P. Baroni, M. Caminada, M. Giacomin, Abstract argumentation frameworks and their semantics, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 4, pp. 159–236.

[34] G. Brewka, S. Woltran, Abstract dialectical frameworks, in: F. Lin, U. Sattler, M. Truszczynski (Eds.), Proc. KR, AAAI Press, 2010, pp. 102–111.

[35] P. Besnard, A. J. García, A. Hunter, S. Modgil, H. Prakken, G. R. Simari, F. Toni, Introduction to structured argumentation, Argument Comput. 5 (1) (2014) 1–4.

[36] S. Modgil, H. Prakken, Abstract rule-based argumentation, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 6, pp. 287–364.

[37] P. Besnard, A. Hunter, A review of argumentation based on deductive arguments, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 9, pp. 437–484.

[38] K. Čyras, X. Fan, C. Schulz, F. Toni, Assumption-based argumentation: Disputes, explanations, preferences, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 7, pp. 365–408.

[39] F. Cerutti, S. A. Gaggl, M. Thimm, J. P. Wallner, Foundations of implementations for formal argumentation, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 14, pp. 689–767.

[40] M. Thimm, S. Villata, The first international competition on computational models of argumentation: Results and analysis, Artif. Intell. 252 (2017) 267–294.

[41] S. A. Gaggl, T. Linsbichler, M. Maratea, S. Woltran, Design and results of the second international competition on computational models of argumentation, Artif. Intell. 279 (2020).

[42] S. Bistarelli, L. Kotthoff, F. Santini, C. Taticchi, Summary report for the third international competition on computational models of argumentation, AI Mag. 42 (3) (2021) 70–73.

[43] S. Bistarelli, L. Kotthoff, J.-M. Lagniez, E. Lonca, J.-G. Mailly, J. Rossit, F. Santini, C. Taticchi, The third and fourth international competitions on computational models of argumentation: Design, results and analysis, Argument Comput. Pre-press (Pre-press) (2024) 1–73.

[44] J. Lagniez, E. Lonca, J. Mailly, J. Rossit, Introducing the fourth international competition on computational models of argumentation, in: S. A. Gaggl, M. Thimm, M. Vallati (Eds.), Proc. SAFA, Vol. 2672 of CEUR Workshop Proceedings, CEUR-WS.org, 2020, pp. 80–85.

[45] W. Dvorák, P. E. Dunne, Computational problems in formal argumentation and their complexity, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 13, pp. 631–687.

[46] A. Bondarenko, P. M. Dung, R. A. Kowalski, F. Toni, An abstract, argumentation-theoretic approach to default reasoning, Artif. Intell. 93 (1997) 63–101.

[47] G. Alfano, S. Greco, F. Parisi, Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach, in: C. Sierra (Ed.), Proc. IJCAI, ijcai.org, 2017, pp. 49–55.

[48] G. Alfano, S. Greco, F. Parisi, An efficient algorithm for skeptical preferred acceptance in dynamic argumentation frameworks, in: S. Kraus (Ed.), Proc. IJCAI, ijcai.org, 2019, pp. 18–24.

[49] A. Niskanen, M. Järvisalo, Algorithms for dynamic argumentation frameworks: An incremental SAT-based approach, in: G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), Proc. ECAI, Vol. 325 of FAIA, IOS Press, 2020, pp. 849–856.

[50] G. Alfano, S. Greco, F. Parisi, Incremental computation in dynamic argumentation frameworks, IEEE Intell. Syst. 36 (6) (2021) 80–86.

[51] G. Alfano, S. Greco, Incremental skeptical preferred acceptance in dynamic argumentation frameworks, IEEE Intell. Syst. 36 (2) (2021) 6–12.

[52] J. Lagniez, E. Lonca, J. Mailly, A sat-based approach for argumentation dynamics, in: M. Dastani, J. S. Sichman, N. Alechina, V. Dignum (Eds.), Proc. AAMAS, ACM, 2024, pp. 2351–2353.

[53] M. Caminada, Semi-stable semantics, in: P. E. Dunne, T. J. M. Bench-Capon (Eds.), Proc. COMMA, Vol. 144 of FAIA, IOS Press, 2006, pp. 121–130.

[54] B. Verheij, Two approaches to dialectical argumentation: admissible sets and argumentation stages, in: J.-J. Meyer, L. van der Gaag (Eds.), Proc. NAIC, Utrecht University, 1996, pp. 357–368.

[55] P. M. Dung, P. Mancarella, F. Toni, Computing ideal sceptical argumentation, Artif. Intell. 171 (10-15) (2007) 642–674.

[56] Y. Dimopoulos, A. Torres, Graph theoretical structures in logic programs and default theories, Theor. Comput. Sci. 170 (1-2) (1996) 209–244.

[57] P. E. Dunne, M. J. Wooldridge, Complexity of abstract argumentation, in: G. R. Simari, I. Rahwan (Eds.), Argumentation in Artificial Intelligence, Springer, 2009, pp. 85–104.

[58] S. Coste-Marquis, C. Devred, P. Marquis, Symmetric argumentation frameworks, in: L. Godo (Ed.), Proc. ECSQARU, Vol. 3571 of LNCS, Springer, 2005, pp. 317–328.

[59] P. E. Dunne, The computational complexity of ideal semantics, Artif. Intell. 173 (18) (2009) 1559–1591.

[60] W. Dvorák, S. Woltran, Complexity of semi-stable and stage semantics in argumentation frameworks, Inf. Process. Lett. 110 (11) (2010) 425–430.

[61] M. W. A. Caminada, W. A. Carnielli, P. E. Dunne, Semi-stable semantics, J. Log. Comput. 22 (5) (2012) 1207–1254.

[62] P. E. Dunne, T. J. M. Bench-Capon, Coherence in finite argument systems, Artif. Intell. 141 (1/2) (2002) 187–203.

[63] Y. Dimopoulos, B. Nebel, F. Toni, On the computational complexity of assumption-based argumentation for default reasoning, Artif. Intell. 141 (1/2) (2002) 57–78.

[64] K. Cyras, Q. Heinrich, F. Toni, Computational complexity of flat and generic assumption-based argumentation, with and without probabilities, Artif. Intell. 293 (2021) 103449.

[65] B. Fazzinga, S. Flesca, F. Furfaro, L. Pontieri, Process mining meets argumentation: Explainable interpretations of low-level event logs via abstract argumentation, Inf. Syst. 107 (2022) 101987.

[66] A. Raymond, M. Malencia, G. Paulino-Passos, A. Prorok, Agree to disagree: Subjective fairness in privacy-restricted decentralised conflict resolution, Frontiers Robotics AI 9 (2022) 733876.

[67] M. Bernreiter, J. Maly, O. Nardi, S. Woltran, Combining voting and abstract argumentation to understand online discussions, in: M. Dastani, J. S. Sichman, N. Alechina, V. Dignum (Eds.), Proc. AAMAS, IFAAMAS/ACM, 2024, pp. 170–179.

[68] Y. Dimopoulos, J. Mailly, P. Moraitis, Arguing and negotiating using incomplete negotiators profiles, Auton. Agents Multi Agent Syst. 35 (2) (2021) 18.

[69] N. Kökciyan, N. Yaglikci, P. Yolum, An argumentation approach for resolving privacy disputes in online social networks, ACM Trans. Internet Techn. 17 (3) (2017) 27:1–27:22.

[70] T. Lehtonen, A. Rapberger, M. Ulbricht, J. P. Wallner, Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity, in: P. Marquis, T. C. Son, G. Kern-Isberner (Eds.), Proc. KR, 2023, pp. 440–450.

[71] T. Lehtonen, J. P. Wallner, M. Järvisalo, From structured to abstract argumentation: Assumption-based acceptance via AF reasoning, in: A. Antonucci, L. Cholvy, O. Papini (Eds.), Proc. ECSQARU, Vol. 10369 of LNCS, Springer, 2017, pp. 57–68.

[72] T. Lehtonen, J. P. Wallner, M. Järvisalo, Algorithms for reasoning in a default logic instantiation of assumption-based argumentation, in: F. Toni, S. Polberg, R. Booth, M. Caminada, H. Kido (Eds.), Proc. COMMA, Vol. 353 of FAIA, IOS Press, 2022, pp. 236–247.

[73] R. Craven, F. Toni, Argument graphs and assumption-based argumentation, Artif. Intell. 233 (2016) 1–59.

[74] Z. Bao, K. Cyras, F. Toni, ABAplus: attack reversal in abstract and structured argumentation with preferences, in: B. An, A. L. C. Bazzan, J. Leite, S. Villata, L. W. N. van der Torre (Eds.), Proc. PRIMA, Vol. 10621 of LNCS, Springer, 2017, pp. 420–437.

[75] T. Lehtonen, J. P. Wallner, M. Järvisalo, Declarative algorithms and complexity results for assumption-based argumentation, J. Artif. Intell. Res. 71 (2021) 265–318.

[76] M. Diller, S. A. Gaggl, P. Gorczyca, Flexible dispute derivations with forward and backward arguments for assumption-based argumentation, in: P. Baroni, C. Benzmüller, Y. N. Wáng (Eds.), Proc. CLAR, Vol. 13040 of LNCS, Springer, 2021, pp. 147–168.

[77] M. Diller, S. A. Gaggl, P. Gorczyca, Strategies in flexible dispute derivations for assumption-based argumentation, in: S. A. Gaggl, J. Mailly, M. Thimm, J. P. Wallner (Eds.), Proc. SAFA, Vol. 3236 of CEUR Workshop Proceedings, CEUR-WS.org, 2022, pp. 59–72.

[78] A. Popescu, J. P. Wallner, Reasoning in assumption-based argumentation using tree-decompositions, in: S. A. Gaggl, M. V. Martinez, M. Ortiz (Eds.), Proc. JELIA, Vol. 14281 of LNCS, Springer, 2023, pp. 192–208.

[79] P. M. Dung, P. M. Thang, N. D. Hung, Modular argumentation for modelling legal doctrines of performance relief, Argument Comput. 1 (1) (2010) 47–69.

[80] X. Fan, S. Liu, H. Zhang, C. Leung, C. Miao, Explained activity recognition with computational assumption-based argumentation, in: G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), Proc. ECAI, Vol. 285 of FAIA, IOS Press, 2016, pp. 1590–1591.

[81] X. Fan, F. Toni, A general framework for sound assumption-based argumentation dialogues, Artif. Intell. 216 (2014) 20–54.

[82] Y. Gao, A random model for argumentation framework: Phase transitions, empirical hardness, and heuristics, in: C. Sierra (Ed.), Proc. IJCAI, ijcai.org, 2017, pp. 503–509.

[83] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: An automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (2009) 267–306.

[84] A. Balint, A. Belov, M. Järvisalo, C. Sinz, Overview and analysis of the SAT Challenge 2012 solver competition, Artif. Intell. 223 (2015) 120–155.

[85] M. Järvisalo, T. Lehtonen, A. Niskanen (Eds.), Solver and Benchmark Descriptions of ICCMA 2023: 5th International Competition on Computational Models of Argumentation, Vol. B-2023-3 of Department of Computer Science Series of Publications B, University of Helsinki, Finland, 2023.

[86] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. A. Kowalski, K. A. Bowen (Eds.), Proc. ICLP/SLP, MIT Press, 1988, pp. 1070–1080.

[87] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (3-4) (1999) 241–273.

[88] U. Egly, S. A. Gaggl, S. Woltran, ASPARTIX: Implementing argumentation frameworks using answer-set programming, in: M. G. de la Banda, E. Pontelli (Eds.), Proc. ICLP, Vol. 5366 of LNCS, Springer, 2008, pp. 734–738.

[89] T. Balyo, A. Biere, M. Iser, C. Sinz, SAT Race 2015, Artif. Intell. 241 (2016) 45–65.

[90] N. Eén, N. Sörensson, Temporal induction by incremental SAT solving, Electronic Notes in Theoretical Computer Science 89 (4) (2003) 543–560.

[91] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, Vol. 336 of FAIA, IOS Press, 2021.

[92] P. Besnard, S. Doutre, Checking the acceptability of a set of arguments, in: J. P. Delgrande, T. Schaub (Eds.), Proc. NMR, 2004, pp. 59–64.

[93] W. Dvorák, M. Järvisalo, J. P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, Artif. Intell. 206 (2014) 53–78.

[94] M. J. H. Heule, Proofs of unsatisfiability, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, Vol. 336 of FAIA, IOS Press, 2021, pp. 635–668.

[95] G. Audemard, L. Simon, On the Glucose SAT solver, Int. J. Artif. Intell. Tools 27 (1) (2018) 1840001:1–1840001:25.

[96] A. Ignatiev, A. Morgado, J. Marques-Silva, PySAT: A Python toolkit for prototyping with SAT oracles, in: O. Beyersdorff, C. M. Wintersteiger (Eds.), Proc. SAT, Vol. 10929 of LNCS, Springer, 2018, pp. 428–437.

[97] N. Wetzler, M. Heule, W. A. H. Jr., DRAT-trim: Efficient checking and trimming using expressive clausal proofs, in: C. Sinz, U. Egly (Eds.), Proc. SAT, Vol. 8561 of LNCS, Springer, 2014, pp. 422–429.

[98] F. Cerutti, N. Oren, H. Strass, M. Thimm, M. Vallati, A benchmark framework for a computational argumentation competition, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Proc. COMMA, Vol. 266 of FAIA, IOS Press, 2014, pp. 459–460.

[99] M. Caminada, P. E. Dunne, Strong admissibility revisited: Theory and applications, Argument Comput. 10 (3) (2019) 277–300.

[100] F. Cerutti, M. Giacomin, M. Vallati, Generating structured argumentation frameworks: AFBenchGen2, in: P. Baroni, T. F. Gordon, T. Scheffler, M. Stede (Eds.), Proc. COMMA, Vol. 287 of FAIA, IOS Press, 2016, pp. 467–468.

[101] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.

[102] P. Erdős, A. Rényi, On random graphs I, Publ. Math. Debr. 6 (290-297) (1959) 18.

[103] A. Sideris, Y. Dimopoulos, Constraint propagation in propositional planning, in: R. I. Brafman, H. Geffner, J. Hoffmann, H. A. Kautz (Eds.), Proc. ICAPS, AAAI, 2010, pp. 153–160.

[104] A. Z. Wyner, T. J. M. Bench-Capon, P. E. Dunne, F. Cerutti, Senses of 'argument' in instantiated argumentation frameworks, Argument Comput. 6 (1) (2015) 50–72.

[105] M. Caminada, B. Verheij, On the existence of semi-stable extensions, in: Proc. BNAIC, 2010.

[106] D. J. Watts, S. H. Strogatz, Collective dynamics of "small-world" networks, Nature 393 (6684) (1998) 440–442.

[107] B. Yun, M. Croitoru, S. Vesic, P. Bisquert, DAGGER: Datalog+/- argumentation graph generator, in: E. André, S. Koenig, M. Dastani, G. Sukthankar (Eds.), Proc. AAMAS, IFAAMAS, 2018, pp. 1841–1843.

[108] J. Lagniez, E. Lonca, J. Mailly, J. Rossit, A new evolutive generator for graphs with communities and its application to abstract argumentation, in: O. Cocarascu, S. Doutre, J. Mailly, A. Rago (Eds.), Proceedings of the First International Workshop on Argumentation and Applications, Vol. 3472 of CEUR Workshop Proceedings, CEUR-WS.org, 2023, pp. 52–64.

[109] L. Malmqvist, Approximate solutions to abstract argumentation problems using graph neural networks., Ph.D. thesis, University of York (2022).

[110] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Proc. NeurIPS, 2019, pp. 8024–8035.

[111] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, Z. Zhang, Deep graph library: Towards efficient and scalable deep learning on graphs, CoRR abs/1909.01315 (2019). arXiv:1909.01315.

[112] J. Delobelle, J. Mailly, J. Rossit, Revisiting approximate reasoning based on grounded semantics, in: Z. Bouraoui, S. Vesic (Eds.), Proc. ECSQARU, Vol. 14294 of LNCS, Springer, 2023, pp. 71–83.

[113] P. Besnard, A. Hunter, A logic-based theory of deductive arguments, Artif. Intell. 128 (1-2) (2001) 203–235.

[114] B. Bliem, M. Morak, S. Woltran, D-FLAT: Declarative problem solving using tree decompositions and answer-set programming, Theory Pract. Log. Program. 12 (4-5) (2012) 445–464.

[115] B. Bliem, G. Charwat, M. Hecher, S. Woltran, D-FLAT^2: Subset minimization in dynamic programming on tree decompositions made easy, Fundam. Informaticae 147 (1) (2016) 27–61.

[116] A. Niskanen, M. Järvisalo, μ-toksia: An efficient abstract argumentation reasoner, in: D. Calvanese, E. Erdem, M. Thielscher (Eds.), Proc. KR, 2020, pp. 800–804.

[117] T. Lehtonen, J. P. Wallner, M. Järvisalo, Harnessing incremental answer set solving for reasoning in assumption-based argumentation, Theory Pract. Log. Program. 21 (6) (2021) 717–734.

[118] J. Lagniez, E. Lonca, J. Mailly, CoQuiAAS: A constraint-based quick abstract argumentation solver, in: Proc. ICTAI, IEEE, 2015, pp. 928–935.

[119] A. Biere, K. Fazekas, M. Fleury, M. Heisinger, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions, Vol. B-2020-1 of Department of Computer Science Report Series B, Department of Computer Science, University of Helsinki, 2020, pp. 50–53.

[120] M. Thimm, F. Cerutti, M. Vallati, Skeptical reasoning with preferred semantics in abstract argumentation without computing preferred extensions, in: Z. Zhou (Ed.), Proc. IJCAI, ijcai.org, 2021, pp. 2069–2075.

[121] L. M. de Moura, N. S. Bjørner, Z3: an efficient SMT solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), Proc. TACAS, Vol. 4963 of LNCS, Springer, 2008, pp. 337–340.

[122] M. Soos, K. Nohl, C. Castelluccia, Extending SAT solvers to cryptographic problems, in: O. Kullmann (Ed.), Proc. SAT, Vol. 5584 of LNCS, Springer, 2009, pp. 244–257.

[123] N. Eén, N. Sörensson, An extensible SAT-solver, in: E. Giunchiglia, A. Tacchella (Eds.), Proc. SAT, Vol. 2919 of LNCS, Springer, 2003, pp. 502–518.

[124] Y. Hamadi, S. Jabbour, L. Sais, ManySAT: a parallel SAT solver, J. Satisf. Boolean Model. Comput. 6 (4) (2009) 245–262.

[125] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki, Learning rate based branching heuristic for SAT solvers, in: N. Creignou, D. L. Berre (Eds.), Proc. SAT, Vol. 9710 of LNCS, Springer, 2016, pp. 123–140.

[126] D. Odekerken, F. Bex, A. Borg, B. Testerink, Approximating stability for applied argument-based inquiry, Intell. Syst. Appl. 16 (2022) 200110.

[127] T. Lehtonen, J. P. Wallner, M. Järvisalo, An answer set programming approach to argumentative reasoning in the ASPIC+ framework, in: D. Calvanese, E. Erdem, M. Thielscher (Eds.), Proc. KR, 2020, pp. 636–646.

[128] D. Odekerken, T. Lehtonen, A. Borg, J. P. Wallner, M. Järvisalo, Argumentative reasoning in ASPIC+ under incomplete information, in: P. Marquis, T. C. Son, G. Kern-Isberner (Eds.), Proc. KR, 2023, pp. 531–541.

[129] T. Lehtonen, J. P. Wallner, M. Järvisalo, Computing stable conclusions under the weakest-link principle in the ASPIC+ argumentation formalism, in: G. Kern-Isberner, G. Lakemeyer, T. Meyer (Eds.), Proc. KR, 2022, pp. 215–225.

[130] M. Thimm, T. Rienstra, Approximate reasoning with ASPIC+ by argument sampling, in: S. A. Gaggl, M. Thimm, M. Vallati (Eds.), Proc. SAFA, Vol. 2672 of CEUR Workshop Proceedings, CEUR-WS.org, 2020, pp. 22–33.

[131] M. Snaith, C. Reed, TOAST: Online ASPIC$^+$ implementation, in: B. Verheij, S. Szeider, S. Woltran (Eds.), Proc. COMMA, Vol. 245 of FAIA, IOS Press, 2012, pp. 509–510.

[132] R. Calegari, A. Omicini, G. Pisano, G. Sartor, Arg2P: an argumentation framework for explainable intelligent systems, J. Log. Comput. 32 (2) (2022) 369–401.

[133] C. Cayrol, F. D. de Saint-Cyr, M. Lagasquie-Schiex, Change in abstract argumentation frameworks: Adding an argument, J. Artif. Intell. Res. 38 (2010) 49–84.

[134] R. Baumann, G. Brewka, Expanding argumentation frameworks: Enforcing and monotonicity results, in: P. Baroni, F. Cerutti, M. Giacomin, G. R. Simari (Eds.), Proc. COMMA, Vol. 216 of FAIA, IOS Press, 2010, pp. 75–86.

[135] M. A. Falappa, A. J. García, G. Kern-Isberner, G. R. Simari, On the evolving relation between belief revision and argumentation, Knowl. Eng. Rev. 26 (1) (2011) 35–43.

[136] J. P. Wallner, A. Niskanen, M. Järvisalo, Complexity results and algorithms for extension enforcement in abstract argumentation, J. Artif. Intell. Res. 60 (2017) 1–40.

[137] D. Baumeister, D. Neugebauer, J. Rothe, H. Schadrack, Verification in incomplete argumentation frameworks, Artif. Intell. 264 (2018) 1–26.

[138] R. Baumann, S. Doutre, J.-G. Mailly, J. Wallner, Enforcement in formal argumentation, in: D. M. Gabbay, M. Giacomin, G. R. Simari, M. Thimm (Eds.), Handbook of Formal Argumentation, Vol. 2, College Publications, 2021, pp. 445–510.

[139] D. Baumeister, M. Järvisalo, D. Neugebauer, A. Niskanen, J. Rothe, Acceptance in incomplete argumentation frameworks, Artif. Intell. 295 (2021) 103470.

[140] G. Alfano, S. Greco, F. Parisi, G. I. Simari, G. R. Simari, An incremental approach to structured argumentation over dynamic knowledge bases, in: M. Thielscher, F. Toni, F. Wolter (Eds.), Proc. KR, AAAI Press, 2018, pp. 78–87.

[141] G. Alfano, S. Greco, F. Parisi, G. I. Simari, G. R. Simari, Incremental computation for structured argumentation over dynamic delp knowledge bases, Artif. Intell. 300 (2021) 103553.

[142] B. Testerink, D. Odekerken, F. Bex, A method for efficient argument-based inquiry, in: A. Cuzzocrea, S. Greco, H. L. Larsen, D. Saccà, T. Andreasen, H. Christiansen (Eds.), Proc. FQAS, Vol. 11529 of LNCS, Springer, 2019, pp. 114–125.

[143] A. Borg, F. Bex, Enforcing sets of formulas in structured argumentation, in: M. Bienvenu, G. Lakemeyer, E. Erdem (Eds.), Proc. KR, 2021, pp. 130–140.

[144] A. Rapberger, M. Ulbricht, On dynamics in structured argumentation formalisms, J. Artif. Intell. Res. 77 (2023) 563–643.

[145] H. Prakken, Relating abstract and structured accounts of argumentation dynamics: the case of expansions, in: P. Marquis, T. C. Son, G. Kern-Isberner (Eds.), Pro. KR, 2023, pp. 562–571.

[146] M. Berthold, A. Rapberger, M. Ulbricht, Forgetting aspects in assumption-based argumentation, in: P. Marquis, T. C. Son, G. Kern-Isberner (Eds.), Proc. KR, 2023, pp. 86–96.

[147] F. Cerutti, I. Tachmazidis, M. Vallati, S. Batsakis, M. Giacomin, G. Antoniou, Exploiting parallelism for hard problems in abstract argumentation, in: B. Bonet, S. Koenig (Eds.), Proc. AAAI, AAAI Press, 2015, pp. 1475–1481.

[148] M. Vallati, F. Cerutti, M. Giacomin, On the combination of argumentation solvers into parallel portfolios, in: W. Peng, D. Alahakoon, X. Li (Eds.), Proc. AI, Vol. 10400 of LNCS, Springer, 2017, pp. 315–327.

[149] F. Cerutti, M. Vallati, M. Giacomin, On the impact of configuration on abstract argumentation automated reasoning, Int. J. Approx. Reason. 92 (2018) 120–138.

[150] M. Vallati, F. Cerutti, M. Giacomin, Predictive models and abstract argumentation: the case of high-complexity semantics, Knowl. Eng. Rev. 34 (2019) e6.

[151] S. Doutre, M. Lafages, M. Lagasquie-Schiex, A distributed and clustering-based algorithm for the enumeration problem in abstract argumentation, in: M. Baldoni, M. Dastani, B. Liao, Y. Sakurai, R. Zalila-Wenkstern (Eds.), Proc. PRIMA, Vol. 11873 of LNCS, Springer, 2019, pp. 87–105.

[152] J. Klein, I. Kuhlmann, M. Thimm, Graph neural networks for algorithm selection in abstract argumentation, in: I. Kuhlmann, J. Mumford, S. Sarkadi (Eds.), Proceedings of the 1st Workshop on Argumentation & Machine Learning, Vol. 3208 of CEUR Workshop Proceedings, CEUR-WS.org, 2022, pp. 81–95.

[153] J. Klein, M. Thimm, Revisiting SAT techniques for abstract argumentation, in: H. Prakken, S. Bistarelli, F. Santini, C. Taticchi (Eds.), Proc. COMMA, Vol. 326 of FAIA, IOS Press, 2020, pp. 251–262.

[154] S. Gning, J. Mailly, On the impact of SAT solvers on argumentation solvers, in: S. A. Gaggl, M. Thimm, M. Vallati (Eds.), Proc.SAFA, Vol. 2672 of CEUR Workshop Proceedings, CEUR-WS.org, 2020, pp. 68–73.