Revisiting SAT Techniques for Abstract Argumentation

Jonas KLEIN and Matthias THIMM
University of Koblenz-Landau, Germany

Abstract. We present MINIAF, a general SAT-based abstract argumentation solver that can be used with any SAT solver. We use this general solver to evaluate 12 different SAT solvers wrt. their capability of handling abstract argumentation problems. While our results show that the runtime performance of different SAT solvers are generally comparable, we also observe some statistically significant differences.

Keywords. abstract argumentation, algorithms, satisfiability

1. Introduction

Approaches to formal argumentation [2] encompass non-monotonic reasoning techniques that focus on the interplay between arguments. One of the most influential models in this area is that of abstract argumentation [18] which represents argumentation scenarios as directed graphs, where arguments are identified with vertices and an "attack" between one argument and another is modelled via a directed edge. In order to reason with abstract argumentation frameworks one considers *extensions*, i. e., sets of arguments that are mutually acceptable, given some formal account to "acceptability" [6]. Many of the reasoning problems have been shown to be intractable in general [19] and there has been an increased effort in recent years to develop algorithms and systems to solve problems of practically relevant sizes [32,23]. One of the predominant paradigms for algorithms in this context, is to reduce the reasoning problem to one or more calls to a *satisfiability* (SAT) solver [9]. Systems following this paradigm are, e. g., ArgSemSAT [14,15], pyglaf [1], μ-toksia [29], argmat-sat [30], and many more. The actual systems differ in some subtleties how the reasoning problem is encoded in a SAT problem, strategies for iterative calls to SAT solvers, and, in particular, the employed SAT solver. For example, ArgSemSAT uses MiniSAT[1] [20] while pyglaf and μ-toksia use Glucose [3], and argmat-sat uses CryptoMiniSat5[2].

In this paper, we revisit SAT-based techniques for reasoning with abstract argumentation and, in particular, ask the question *if* and *how* the choice of a concrete SAT solver may influence the performance of the overall argumentation system. In order to address this question independently of any existing SAT-based argumentation solver (that may be tailored towards the use of a concrete SAT-solver), as a first contribution we present MINIAF, a minimal implementation of a reasoning engine making use of SAT-solving

[1] Although [15] also reports on an experimental comparison with using Glucose.
[2] `https://github.com/msoos/cryptominisat`

techniques. This solver can be parametrised by any SAT solver following the command line interface of the SAT competition[3]. As a second contribution, we perform an extensive experimental analysis of running MINIAF with 12 different SAT solvers in order to compare the SAT solvers performance on the ICCMA17 [23] benchmark set. Our findings are that most SAT solvers exhibit a similar performance, although certain deviations can be observed. In summary, the contributions of this paper are as follows.

1. We present MINIAF, a minimal and flexible SAT-based argumentation solver (Section 3).
2. We perform an extensive experimental evaluation parametrising MINIAF with 12 different SAT solvers (Section 4).

We discuss relevant preliminaries in Section 2 and conclude in Section 5.

2. Preliminaries

An *abstract argumentation framework* AF is a tuple $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ where A is a set of arguments and R is a relation $\mathsf{R} \subseteq \mathsf{A} \times \mathsf{A}$. For two arguments $a, b \in \mathsf{A}$ the relation $a\mathsf{R}b$ means that argument a attacks argument b. For $a \in \mathsf{A}$ define $a^- = \{b \mid b\mathsf{R}a\}$ and $a^+ = \{b \mid a\mathsf{R}b\}$. We say that a set $S \subseteq \mathsf{A}$ *defends* an argument $b \in \mathsf{A}$ if for all a with $a\mathsf{R}b$ then there is $c \in S$ with $c\mathsf{R}a$.

Semantics are given to abstract argumentation frameworks by means of extensions [18]. An extension E is a set of arguments $E \subseteq \mathsf{A}$ that is intended to represent a coherent point of view on the argumentation modelled by AF. Arguably, the most important property of a semantics is its admissibility. An extension E is called *admissible* if and only if

1. E is *conflict-free*, i. e., there are no arguments $a, b \in E$ with $a\mathsf{R}b$ and
2. E *defends* every $a \in E$,

and it is called *complete* (CO) if, additionally, it satisfies

3. if E defends a then $a \in E$.

Different types of classical semantics can be phrased by imposing further constraints. In particular, a complete extension E

- is *grounded* (GR) if and only if E is minimal,
- is *preferred* (PR) if and only if E is maximal, and
- is *stable* (ST) if and only if $\mathsf{A} = E \cup \{b \mid \exists a \in E : a\mathsf{R}b\}$.

All statements on minimality/maximality are meant to be with respect to set inclusion. Note that the grounded extension is uniquely determined and that stable extensions may not exist [18].

Example 1. Consider the abstract argumentation framework AF_1 depicted as a directed graph in Figure 1. In AF_1 there are three complete extensions E_1, E_2, E_3 defined via

[3] `http://www.satcompetition.org`

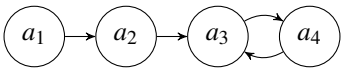

Figure 1. Abstract argumentation framework AF_1 from Example 1.

$$E_1 = \{a_1\}$$
$$E_2 = \{a_1, a_3\}$$
$$E_3 = \{a_1, a_4\}$$

E_1 is also grounded and E_2 and E_3 are both stable and preferred.

Let $\sigma \in \{\mathsf{CO}, \mathsf{GR}, \mathsf{ST}, \mathsf{PR}\}$ be some semantics and $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ be an abstract argumentation framework. Then, an argument $a \in \mathsf{A}$ is *skeptically accepted* in AF if a is contained in *every* σ-extension. An argument $a \in \mathsf{A}$ is *credulously accepted* in AF if a is contained in *some* σ-extension.

An equivalent way of defining different types of semantics is by means of *labellings*, rather than extensions [5,12]. Given a set of arguments S, a *labelling* is a total function $\mathsf{L} : S \to \{\mathtt{in}, \mathtt{out}, \mathtt{undec}\}$. An argument $a \in S$ is either labelled `in`-meaning a is accepted, labelled `out`-meaning a is rejected-or labelled `undec`-meaning the status of a is undecided. Given an $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, the set of all *labellings* is denoted as $\mathfrak{L}(\mathsf{AF})$. A *labelling* $\mathsf{L} \in \mathfrak{L}(\mathsf{AF})$ is called a *complete labelling* if and only if for any $a \in \mathsf{A}$ holds:

1. $\mathsf{L}(a) = \mathtt{in} \Leftrightarrow \forall b \in a^-, \mathsf{L}(b) = \mathtt{out}$;
2. $\mathsf{L}(a) = \mathtt{out} \Leftrightarrow \exists b \in a^-, \mathsf{L}(b) = \mathtt{in}$;

Comparable to the extension-based definition of semantics, other semantics can be phrased by imposing further constraints to a *complete labelling*. Let $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ be an argumentation framework. A *complete labelling* $\mathsf{L} \in \mathfrak{L}(\mathsf{AF})$

- is *grounded* if and only if L is maximising the set of arguments labelled `undec`,
- is *preferred* if and only if L is maximising the set of arguments labelled `in`, and
- is *stable* if and only if there is no argument labelled `undec`.

The following definition further emphasises the inherent connection between extensions and labellings: Let $\mathtt{in}(\mathsf{L}) = \{a \in \mathsf{A} | \mathsf{L}(a) = \mathtt{in}\}$ and $\mathtt{out}(\mathsf{L})$ resp. $\mathtt{undec}(\mathsf{L})$ be defined analogously. A labelling L is a complete (grounded, preferred, stable) labelling if and only if $\mathtt{in}(\mathsf{L})$ is a complete (grounded, preferred, stable) extension.

3. A minimal SAT-based solver: miniAF

MINIAF[4] is a lightweight SAT-based solver for reasoning tasks in abstract argumentation. It is implemented in the `C` programming language and based on the {j}ArgSemSAT [14,16] approach. To solve any reasoning task for a given AF, MINIAF traverses the search space of a complete extension via a SAT solver. In general, this task can be broken down in three sub-tasks: (1) Encoding the constraints analogous to a *complete la-*

[4] Source code available at `https://github.com/jklein94/miniAF`.

belling of the AF as a propositional formula; (2) iteratively modify or generate new formulæ based on previous found models and the reasoning task; and (3) using an external SAT solver to search for models of these formulæ.

The system is capable of solving the following tasks:

- EE-σ: Given $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ enumerate all sets $E \subseteq \mathsf{A}$ that are σ-extensions.
- SE-σ: Given $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ return some set $E \subseteq \mathsf{A}$ that is a σ-extension.
- DC-σ: Given $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, $a \in \mathsf{A}$ decide if a is credulously accepted under σ.
- DS-σ: Given $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, $a \in \mathsf{A}$ decide if a is skeptically accepted under σ.

for $\sigma \in \{\mathsf{CO}, \mathsf{GR}, \mathsf{ST}, \mathsf{PR}\}$. The MINIAF solver is parameterisable with any SAT solver, specified by a absolute path, following the commandline interface of the SAT competition. As input MINIAF supports abstract argumentation frameworks in the `ASPARTIX` format [22] and the `Trivial Graph Format`.[5]

In the following section, a more detailed explanation of the used algorithms is given. Since all algorithms are based on traversing the search space of a complete extension, the encoding of a corresponding complete labeling is first defined. Based on this encoding, the procedures for solving the above-stated problems are described for each semantics σ.

3.1. Complete semantics

For a given $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, MINIAF constructs a propositional formula Π_{AF}, so that each satisfying assignment of Π_{AF} corresponds to a *complete labelling* of AF. In particular, the following SAT encoding is used [13]. Given $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, with $|\mathsf{A}| = k$ and the bijection $\phi : \{1, \ldots, k\} \to \mathsf{A}$ an indexing of A. Let $V(\mathsf{AF}) \triangleq \cup_{1 \leq i \leq |\mathsf{A}|} \{I_i, O_i, U_i\}$ be the variables of AF. The conjunction of clauses (1)–(6), defined of the variables $V(\mathsf{AF})$, is an encoding of a *complete labelling*:

$$\bigwedge_{i \in \{1,\ldots,k\}} ((I_i \vee O_i \vee U_i) \wedge (\neg I_i \vee \neg O_i) \wedge (\neg I_i \vee \neg U_i) \wedge (\neg O_i \vee \neg U_i) \tag{1}$$

$$\bigwedge_{\{i \mid \phi(i)^- = \emptyset\}} (I_i \wedge \neg O_i \wedge \neg U_i) \tag{2}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \emptyset\}} \left(\bigwedge_{\{j \mid \phi(j) \to \phi(i)\}} \neg I_i \vee O_j \right) \tag{3}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \emptyset\}} \left(I_i \vee \left(\bigvee_{\{j \mid \phi(j) \to \phi(i)\}} (\neg O_j) \right) \right) \tag{4}$$

[5] See `http://en.wikipedia.org/wiki/Trivial_Graph_Format`

$$\bigwedge_{\{i|\phi(i)^- \neq \emptyset\}} \left(\neg O_i \vee \left(\bigvee_{\{j|\phi(j) \to \phi(i)\}} I_j \right) \right) \tag{5}$$

$$\bigwedge_{\{i|\phi(i)^- \neq \emptyset\}} \left(\bigwedge_{\{j|\phi(j) \to \phi(i)\}} \neg I_j \vee O_i \right) \tag{6}$$

The resulting formula is in conjunctive normal form (CNF), as from SAT solvers demanded. To enumerate all extensions, each time a solution s is found, the formula Π_{AF} is updated to the conjunction $\Pi_{\mathsf{AF}} \wedge \neg s$, thus excluding the previous result. This formula is then passed back to the SAT solver to find a solution, i. e. an additional *complete labelling*. The procedure is repeated until no satisfying assignment is found, therefore enumerating all extensions.

To decide the credulous acceptance of an argument a, Π_{AF} is updated to $\Pi_{\mathsf{AF}} \wedge I_{\phi^{-1}(a)}$. If some extension with a labelled as `in` exists, i. e. there is a solution to $\Pi_{\mathsf{AF}} \wedge I_{\phi^{-1}(a)}$, a is credulously accepted. To determine whether a is contained in every complete extension and thus skeptically accepted, MINIAF uses the grounded extension.

3.2. Stable semantics

The *stable labellings* are complete labellings with no argument labelled `undec`. Consequently, they are the solutions to the formula $\Pi'_{\mathsf{AF}} := \Pi_{\mathsf{AF}} \wedge \bigwedge_{a \in A} \neg U_{\phi^{-1}(a)}$, which excludes the label `undec` for every argument. The enumeration of all stable extensions is computed the same way as for the complete semantics.

An argument a is credulously accepted, if there is a solution to the formula $\Pi'_{\mathsf{AF}} \wedge I_{\phi^{-1}(a)}$. The question of whether a is labelled as `in` in every stable labelling can be rephrased as: Is there a stable labelling where a is labelled as `out`? The equivalent formula to this question is $\Pi'_{\mathsf{AF}} \wedge O_{\phi^{-1}(a)}$ [16]. If there is a solution to the formula, a is not accepted. In the case that there is no solution and a *stable labelling* exist, a is skeptically accepted.

3.3. Preferred semantics

The *preferred labellings* are computed by using an evolution of the `PrefSAT` algorithm [16]. In general the algorithm consists of two routines: (1) Iterating over a set of complete labellings to identify the preferred ones and (2) an optimization procedure to maximise complete labellings wrt. set inclusion. The credulous acceptance of an argument a is decided by finding—analogous to the complete semantics—a solution to the formula $\Pi_{\mathsf{AF}} \wedge I_{\phi^{-1}(a)}$. To check whether a is contained in every *preferred labelling*, MINIAF subsequently enumerates all *preferred labellings* until it finds a labelling, where a is not in the set of arguments labelled `in`. If no counterexample is found, the argument is skeptically accepted.

3.4. Grounded semantics

Grounded labellings, i. e. complete labellings maximising the set of `undec` arguments, are computed with basically the same optimization procedure as the preferred labellings.[6] However, the arguments labelled `undec` are maximized, rather than the arguments labelled `in`. Since the grounded extension is unique, the problem of credulous and skeptical acceptance of an argument a are equivalent. If a is contained in the *grounded labelling*, it is credulously and skeptically accepted.

4. Experiments

In this section, we present the results of an experimental analysis, in which we investigated the impact of various state-of-the-art SAT solvers on the performance of MINIAF. This analysis aims to give an overview *if* and *how* the overall performance of a SAT-based system is affected by the choice of the exploited SAT-solver. Below, we give a brief description of the investigated SAT solvers and the experimental setup and subsequently discuss our findings.

4.1. Experimental setup

In our experiments, we compared a total of 12 SAT solvers:

CADICAL [8]: is based on conflict-driven clause learning (CDCL) [27] with inprocessing [24].

GLUCOSE (Version 4.1) [4]: is a CDCL solver heavily based on MINISAT [21], with a special focus on removing useless clauses as soon as possible, and an original restart scheme.

The familiy of MAPLELCMDISTCHRONOBT-DL (Version 3, 2.2 and 2.1) [25]: solvers are based on the SAT Competition 2018 winner MAPLELCMDISTCHRONOBT [28] augmented with duplicate learnts heuristic.

MAPLELCMDISTCBTCOREFIRST [17]: is a hack version of MAPLELCMDISTCHRONOBT. This solver adds only Core First Unit Propagation. The remainder keeps unchanged.

MERGESAT [26]: is a CDCL solver based on the competition winner of 2018, MAPLELCMDISTCHRONOBT, and adds several known techniques as well as some novel ideas.

PADC_MAPLE_LCM_DIST [31]: is based on the SAT Competition 2017 winner MAPLE_LCM_DIST and integrates the periodic aggressive learned clause database cleaning (PADC) strategy [31].

PSIDS_MAPLELCMDISTCHRONOBT [31]: is based on MAPLELCMDISTCHRONOBT and integrates the polarity state independent decaying sum (PSIDS) heuristic.

PICOSAT [7] (Version 965): is an attempt to optimise low-level performance of BooleForce,[7] which shares many of its key features with MiniSAT(version 1.14).

[6] Note that we use a reduction to SAT here as well, despite the fact that the grounded labelling can be computed in polynomial time. We do that because we wish to have a general system that makes use of SAT solvers as often as possible without relying on proprietary algorithms.

[7] `http://fmv.jku.at/booleforce`.

RELAXED_LCMDISTCHRONOBT [11]: is a CDCL-based solver. The method used for this solver aims to improve CDCL solvers by relaxing the backtracking and integrating local search techniques. As a local search solver CCANR [10] is used.

OPTSAT [17]: is a CDCL solver using the core first unit propagation technique.

For the evaluation we used the ICCMA'17 benchmark.[8] This benchmark is made up of three groups: A, B and C. Each group, in turn, consists of 350 instances classified into 5 hardness categories: (1) very easy, (2) easy, (3) medium, (4) hard and (5) too hard. Since the *grounded* labelling is uniquely defined, only the SE and DC problems were employed. According to the ICCMA'17 rules [23], each task was assigned to a group as follows:

- A: DS-PR, EE-PR, EE-CO
- B: DS-ST, DC-ST, SE-ST, EE-ST, DC-PR, SE-PR, DC-CO
- C: DS-CO, SE-CO, DC-GR, SE-GR

For all 14 tasks, MINIAF was run 12 times—every time parameterised with a different SAT solver—on the instances of the corresponding group. A cutoff value of 600 seconds (10 minutes) per instance was imposed. All SAT solvers were executed with their default (and non-parallel) configuration. For each SAT solver and task we recorded: (1) the number of solved instances, (2) the number of unsolved instances and (3) the execution time per solved instance.

We ran the experiments on a virtual machine running Ubuntu 18.04 with a 2.9 GHz CPU core and 8GB of RAM.

4.2. Results

The performance achieved by MINIAF is measured in terms of instance coverage (**Cov.**)—percentage of successfully analysed instances—and Penalised Average Runtime (**PAR10**). The PAR10 score is a hybrid measure, defined as the average of runtimes which counts (1) the runtimes of unsolved instances as ten times the cutoff value and (2) the runtimes of solved instances as the actual runtimes. Thus, it allows runtime to be considered and still setting a strong focus on instance coverage. The results of this analysis, with regards to the different semantics, are shown in Table 1 (CO track), Table 2 (ST track), Table 3 (PR track) and Table 4 (GR track). The first column (**SAT**) contains the names of the used SAT solvers. Hereinafter, we will refer to MINIAF just with the name of the used SAT solver to express MINIAF was parameterised with this solver.

Considering the performance achieved on all instances of a track (**ALL**), most SAT solvers are generally comparable. The CADICAL solver performs best (PAR10 score and coverage) for the CO, ST and GR track. As for the PR track, the MAPLELCMDISTCBTCOREFIRST system accomplished the best results. However, the fact that a concrete SAT solver excels all other systems on the whole set of instances, does not necessarily mean this solver exhibits the best performance for all computational tasks of the considered semantics. Rather, we note that for three (CO, ST, PR) of the four semantics, there is at least one task where the overall best solver for this track is outperformed

[8] A more detailed description of the ICCMA'17 benchmark and the selection process can be found here `http://argumentationcompetition.org/2017/benchmark_selection_iccma2017.pdf`.

CO										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	**1027.65**	**83.04**	3009.22	50.29	**35.83**	**99.43**	**41.75**	**99.33**	882.95	85.43
Glucose	1100.00	82.15	3084.61	49.14	131.47	98.29	197.61	97.33	857.38	86.00
MapleLCMDistChronoBT-DL-v2.1	1057.65	82.89	2974.07	51.14	229.29	96.57	230.37	96.67	678.69	89.14
MapleLCMDistChronoBT-DL-v2.2	1065.61	82.74	2988.36	50.86	244.73	96.29	230.38	96.67	679.64	89.14
MapleLCMDistChronoBT-DL-v3	1115.78	81.85	3022.13	50.29	245.54	96.29	249.69	96.33	822.01	86.57
MapleLCMdistCBTcoreFirst	1092.59	82.30	2968.79	51.14	269.04	96.00	330.52	95.00	693.15	88.86
MergeSAT	1054.95	82.89	2965.29	51.14	197.32	97.14	230.39	96.67	709.02	88.57
PADC_Maple_LCM_Dist	1050.67	82.96	**2947.17**	**51.43**	246.77	96.29	230.77	96.67	**660.84**	**89.43**
PSIDS_MapleLCMDistChronoBT	1055.16	82.89	2950.45	**51.43**	228.95	96.57	248.92	96.33	677.13	89.14
PicoSAT	1093.74	81.93	3212.29	46.86	35.90	**99.43**	41.90	**99.33**	934.61	84.57
Relaxed_LCMDistChronoBT	1128.34	81.78	3121.88	48.86	213.51	96.86	230.26	96.67	819.41	86.86
optsat	1156.70	81.70	3183.14	47.43	258.05	97.14	306.84	96.67	757.36	87.71

Table 1. Performance comparison for all instances (ALL) and the different tasks of the CO track: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

ST										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	**959.40**	**84.43**	**1717.27**	**72.00**	**589.89**	**90.57**	**834.88**	**86.57**	695.56	88.57
Glucose	1321.71	78.71	2353.80	61.71	861.00	86.57	1249.78	80.00	822.25	86.57
MapleLCMDistChronoBT-DL-v2.1	1161.59	81.50	2232.03	64.00	740.48	88.57	1044.60	83.43	629.24	**90.00**
MapleLCMDistChronoBT-DL-v2.2	1160.10	81.57	2196.68	64.57	730.31	88.86	1035.16	83.71	678.27	89.14
MapleLCMDistChronoBT-DL-v3	1219.18	80.50	2180.47	64.86	837.35	86.86	1117.99	82.29	740.91	88.00
MapleLCMdistCBTcoreFirst	1156.61	81.57	2210.21	64.29	707.02	89.14	1064.30	83.14	644.90	89.71
MergeSAT	1146.66	81.71	2206.40	64.29	722.25	88.86	1029.71	83.71	628.27	**90.00**
PADC_Maple_LCM_Dist	1136.12	81.86	2171.51	64.86	702.17	89.14	1043.43	83.43	**627.36**	**90.00**
PSIDS_MapleLCMDistChronoBT	1151.37	81.64	2159.23	65.14	737.93	88.57	1080.59	82.86	627.75	**90.00**
PicoSAT	1416.85	76.64	2251.46	62.86	1074.08	82.29	1474.78	75.71	867.06	85.71
Relaxed_LCMDistChronoBT	1326.29	78.93	2528.81	59.14	875.66	86.57	1145.96	82.00	754.72	88.00
optsat	1130.05	82.00	2060.24	66.86	736.38	88.57	1016.62	84.00	706.97	88.57

Table 2. Performance comparison for all instances (ALL) and the different tasks of the ST track: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

PR										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	1383.89	77.33	**2537.02**	**58.29**	1039.66	83.14	1024.59	83.33	882.95	85.43
Glucose	1414.81	76.96	2710.77	55.43	1027.31	83.71	1005.27	83.67	857.38	86.00
MapleLCMDistChronoBT-DL-v2.1	1294.54	79.04	2610.56	57.14	896.74	86.00	941.76	84.67	678.69	89.14
MapleLCMDistChronoBT-DL-v2.2	1286.44	79.19	2594.44	57.43	862.88	86.57	962.53	84.33	679.64	89.14
MapleLCMDistChronoBT-DL-v3	1360.39	77.93	2665.65	56.29	934.23	85.43	962.87	84.33	822.01	86.57
MapleLCMdistCBTcoreFirst	**1262.10**	**79.56**	2608.12	57.14	**776.62**	**88.00**	**921.93**	**85.00**	693.15	88.86
MergeSAT	1301.08	78.89	2576.44	57.71	908.22	85.71	962.22	84.33	709.02	88.57
PADC_Maple_LCM_Dist	1279.18	79.26	2605.32	57.14	843.38	86.86	961.86	84.33	**660.84**	**89.43**
PSIDS_MapleLCMDistChronoBT	1289.68	79.11	2591.29	57.43	879.38	86.29	964.45	84.33	677.13	89.14
PicoSAT	1575.31	74.07	2743.64	54.86	1303.45	78.57	1276.93	79.00	934.61	84.57
Relaxed_LCMDistChronoBT	1467.90	76.37	2868.76	53.14	1093.36	82.86	1027.09	83.67	819.41	86.86
optsat	1319.90	78.59	2629.04	56.86	907.54	85.71	929.98	**85.00**	757.36	87.71

Table 3. Performance comparison for all instances (ALL) and the different tasks of the PR track: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

by another system. The only exception is the GR semantics. For the GR track CADICAL performs best on all instances and each task (SE-GR, DC-GR).

Furthermore, we observe noticeably differences for individual tasks and semantics[9]:

[9]We also carried out a significant analysis of the execution times that show significant differences. As a statistical test, we used the Kruskal–Wallis test and the Dunn-Bonferroni test for post-hoc analysis.

SAT	GR					
	ALL		SE		DC	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	**38.64**	**99.38**	**35.95**	**99.43**	**41.78**	**99.33**
Glucose	235.79	96.92	219.52	97.14	254.78	96.67
MapleLCMDistChronoBT-DL-v2.1	231.39	96.62	231.12	96.57	231.70	96.67
MapleLCMDistChronoBT-DL-v2.2	239.68	96.46	246.43	96.29	231.81	96.67
MapleLCMDistChronoBT-DL-v3	239.85	96.46	246.74	96.29	231.82	96.67
MapleLCMdistCBTcoreFirst	381.63	94.15	356.64	94.57	410.79	93.67
MergeSAT	213.99	96.92	198.40	97.14	232.16	96.67
PADC_Maple_LCM_Dist	239.39	96.46	245.94	96.29	231.76	96.67
PSIDS_MapleLCMDistChronoBT	231.23	96.62	230.71	96.57	231.83	96.67
PicoSAT	38.78	**99.38**	36.06	**99.43**	41.96	**99.33**
Relaxed_LCMDistChronoBT	222.20	96.77	198.69	97.14	249.62	96.33
optsat	308.00	96.46	295.75	96.57	322.30	96.33

Table 4. Performance comparison for all instances (ALL) and the different tasks of the GR track: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface

Two of the examined SAT systems, namely CADICAL and PICOSAT, perform distinctly better on the instances of the GR track. A similar scenario shows the result for the CO semantics in Table 1. Here too, CADICAL and PICOSAT stand out from the other solvers for the CO-SE and CO-DS tasks.[10] It is interesting, however, that the PICOSAT solver achieves the worst results in terms of coverage and PAR10 score for the all other tasks on this track. In addition, we find that some solvers tend to do better for a certain reasoning problem, regardless of the semantics under consideration. For example, the SAT solver PADC_MAPLE_LCM_DIST achieves the best results for the DC problem of semantics CO, ST and PR. The SE problem for semantics CO, ST and GR is best solved by CADICAL. Surprisingly, none of the MAPLELCMDISTCHRONOBT-DL (Version 3, 2.2, 2.1) solvers achieves the best performance for any task, even though they ranked second place for the SAT track (Version 3, 2.2 and 2.1) and first place for the UNSAT (Version 3) and SAT+UNSAT (Version 3, 2.2, 2.1) track in the last years SAT competition.[11]

Apart from the inherent complexity of a particular reasoning problem, the performance of an argumentation system is also affected by the hardness of the instance to be solved. In order to identify deviations in the performance concerning the level of difficulty of an instance, we compared the SAT solvers based on the hardness categories of the benchmark set.

CADICAL is able to solve all instances of the hardness categories Very Easy,[12] Easy and Medium best. For the Hard and Too Hard instances, PADC_MAPLE_LCM_DIST attains the best results. Moreover we observe—covering the previously presented results—that most solvers are comparable, although there are some differences between the categories. For example, CADICAL and PICOSAT perform clearly better for the Easy instances.

Another interesting scenario is shown in Table 7. Albeit, the PICOSAT system is nearly indistinguishable from the best solver for all instances of the Very Easy, Easy and the DC instances of the Medium set, it performs significantly worse for all other Medium

[10] This observation is actually not so surprising, as CO-SE can be answered just as GR-SE and CO-DS and DC-GR are identical.

[11] `http://sat-race-2019.ciirc.cvut.cz/index.php`.

[12] Since the results for the Very Easy instances are practically identical for most solvers, we refrain from presenting them in a table.

Easy										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	**207.82**	**96.71**	**789.64**	**87.33**	**67.9**	**99.0**	**87.13**	**98.67**	1.89	**100.0**
Glucose	415.02	93.43	1475.29	76.0	145.83	98.0	258.13	96.0	6.66	**100.0**
MapleLCMDistChronoBT-DL-v2.1	501.59	92.0	1455.93	76.67	348.34	94.5	293.54	95.33	95.12	98.5
MapleLCMDistChronoBT-DL-v2.2	502.05	92.0	1457.71	76.67	348.58	94.5	293.68	95.33	95.05	98.5
MapleLCMDistChronoBT-DL-v3	509.46	91.86	1491.17	76.0	347.43	94.5	296.01	95.33	95.3	98.5
MapleLCMdistCBTcoreFirst	486.63	92.29	1491.94	76.0	265.85	96.0	294.77	95.33	97.34	98.5
MergeSAT	508.19	91.86	1490.15	76.0	344.68	94.5	295.05	95.33	95.07	98.5
PADC_Maple_LCM_Dist	507.85	91.86	1488.85	76.0	345.6	94.5	293.44	95.33	95.16	98.5
PSIDS_MapleLCMDistChronoBT	492.17	92.14	1415.43	77.33	344.74	94.5	294.72	95.33	95.25	98.5
PicoSAT	251.28	96.0	912.25	85.33	69.34	**99.0**	165.61	97.33	**1.75**	**100.0**
Relaxed_LCMDistChronoBT	629.48	90.0	1811.17	71.0	338.86	94.67	375.38	94.0	89.93	98.67
optsat	466.13	92.71	1406.24	77.33	247.87	96.5	302.67	95.33	101.89	98.5

Table 5. Performance comparison for all instances (ALL) and the different tasks of the Easy instances: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

Medium										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	**414.46**	**93.43**	**1507.2**	**75.67**	**200.05**	**97.0**	**157.16**	**97.67**	**2.29**	**100.0**
Glucose	577.71	91.57	1857.93	70.33	280.67	**97.0**	349.88	95.33	85.45	99.25
MapleLCMDistChronoBT-DL-v2.1	580.02	91.43	1800.67	71.33	328.18	96.0	304.37	96.0	123.11	98.5
MapleLCMDistChronoBT-DL-v2.2	582.79	91.36	1796.89	71.33	339.31	95.75	306.31	96.0	123.07	98.5
MapleLCMDistChronoBT-DL-v3	594.49	91.14	1816.03	71.0	355.57	95.5	322.83	95.67	121.01	98.5
MapleLCMdistCBTcoreFirst	631.71	90.5	1796.96	71.33	399.79	94.75	360.5	95.0	193.08	97.25
MergeSAT	563.79	91.64	1812.36	71.0	265.99	**97.0**	302.58	96.0	121.06	98.5
PADC_Maple_LCM_Dist	580.02	91.36	1791.13	71.33	338.01	95.75	302.25	96.0	122.04	98.5
PSIDS_MapleLCMDistChronoBT	577.46	91.43	1794.49	71.33	311.87	96.25	322.7	95.67	121.35	98.5
PicoSAT	738.02	87.93	2161.91	64.67	577.44	90.5	509.28	91.67	2.24	**100.0**
Relaxed_LCMDistChronoBT	829.19	87.38	2684.03	56.5	466.09	93.67	394.88	94.5	204.1	97.33
optsat	601.44	91.36	1801.61	71.33	371.92	95.75	306.55	96.33	152.0	98.25

Table 6. Performance comparison for all instances (ALL) and the different tasks of the Medium instances: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

Hard										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	952.46	84.5	3390.94	44.33	**413.22**	**93.4**	**553.84**	**91.14**	235.99	96.22
Glucose	1125.6	81.81	3728.3	38.67	646.12	89.8	744.06	88.0	219.98	96.89
MapleLCMDistChronoBT-DL-v2.1	966.19	84.44	3407.01	44.33	473.8	92.6	591.66	90.57	177.36	97.33
MapleLCMDistChronoBT-DL-v2.2	964.68	84.5	3366.79	45.0	476.57	92.6	612.96	90.29	179.19	97.33
MapleLCMDistChronoBT-DL-v3	1003.63	83.88	3433.98	44.0	557.75	91.2	616.48	90.29	179.94	97.33
MapleLCMdistCBTcoreFirst	990.34	84.06	3358.3	45.0	514.61	92.0	614.44	90.29	232.66	96.44
MergeSAT	946.33	84.75	3298.24	**46.0**	462.17	92.8	609.01	90.29	178.72	97.33
PADC_Maple_LCM_Dist	**940.79**	**84.81**	**3292.2**	**46.0**	448.95	93.0	610.44	90.29	**176.59**	**97.33**
PSIDS_MapleLCMDistChronoBT	946.44	84.75	3298.69	46.0	461.65	92.8	611.18	90.29	177.69	97.33
PicoSAT	1155.87	81.06	3775.77	37.67	610.91	90.0	807.0	86.86	286.11	95.56
Relaxed_LCMDistChronoBT	1042.21	83.33	3578.9	41.5	490.12	92.25	623.38	90.0	219.18	96.86
optsat	1021.56	84.06	3496.95	42.67	539.23	92.4	635.53	90.57	207.46	97.33

Table 7. Performance comparison for all instances (ALL) and the different tasks of the Hard instances: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

instances. The performance of PICOSAT is also in the lower range for the Hard and Too Hard set. The opposite is the case for the RELAXED_LCMDISTCHRONOBT solver. It tends to achieve similar—for two problems even higher—coverage for the Hard and Too Hard instances, but lower coverage for the Very Easy, Easy and Medium set. We can derive that a good result on a particular hardness category (or problem), does not

Too Hard										
SAT	**ALL**		**EE**		**SE**		**DS**		**DC**	
	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
CaDiCal	3313.65	45.0	6000.0	0.0	2949.47	51.0	2387.0	60.67	2555.19	57.67
Glucose	3464.51	42.43	6000.0	0.0	3010.92	50.0	2627.89	56.67	2766.28	54.0
MapleLCMDistChronoBT-DL-v2.1	3107.57	48.71	6000.0	0.0	2837.28	53.0	2498.8	58.67	2055.83	66.67
MapleLCMDistChronoBT-DL-v2.2	3109.57	48.71	6000.0	0.0	2731.88	55.0	2464.64	59.33	2112.71	65.67
MapleLCMDistChronoBT-DL-v3	3347.25	44.43	6000.0	0.0	2891.96	52.0	2653.66	56.0	2519.44	58.33
MapleLCMdistCBTcoreFirst	3115.65	48.57	6000.0	0.0	2673.72	56.0	2538.8	58.0	2109.2	65.67
MergeSAT	3140.19	48.14	6000.0	0.0	2901.41	52.0	2466.64	59.33	2126.66	65.33
PADC_Maple_LCM_Dist	**3072.81**	**49.29**	6000.0	0.0	2719.46	55.0	2497.63	58.67	**2014.59**	**67.33**
PSIDS_MapleLCMDistChronoBT	3132.51	48.29	6000.0	0.0	2897.68	52.0	2581.88	57.33	2052.36	66.67
PicoSAT	3569.55	40.57	6000.0	0.0	3066.47	49.0	3011.66	50.0	2800.98	53.33
Relaxed_LCMDistChronoBT	3206.9	46.92	6000.0	0.0	**2538.58**	**58.0**	**2136.9**	**65.0**	2447.71	59.6
optsat	3231.18	46.57	6000.0	0.0	2950.19	51.0	2447.11	60.0	2332.47	61.67

Table 8. Performance comparison for all instances (ALL) and the different tasks of the Too Hard instances: Used SAT solver, instance coverage and PAR10 score. Best result highlighted in boldface.

necessarily transfer to other categories.

5. Summary and Conclusion

In this paper, we compared the performance of MINIAF parameterised with 12 different state-of-the-art SAT solvers on the ICCMA17 benchmark. The results of our analysis shows that: (1) the performance of most SAT solvers is generally comparable for all considered problems, but (2) some systems tend to be more suitable for individual reasoning tasks than others. These insights indicate that the use of SAT-based portfolio systems—i. e., systems that select different SAT solvers depending on instance and task information—may be beneficial for addressing a wide variety of abstract argumentation problems. Moreover, since all SAT solvers have been evaluated with their standard configurations, future work could investigate the influence of various parameter configurations on performance.

Acknowledgements The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (project number 375588274).

References

[1] M. Alviano. The pyglaf argumentation reasoner. In *The Third International Competition on Computational Models of Argumentation (ICCMA'19)*, 2019.

[2] K. Atkinson, P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G. R. Simari, M. Thimm, and S. Villata. Toward artificial argumentation. *AI Magazine*, 38(3):25–36, October 2017.

[3] G. Audemard and L. Simon. Lazy clause exchange policy for parallel SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 197–205, 2014.

[4] G. Audemard and L. Simon. Glucose and Syrup in the SAT'17. *Proceedings of SAT Competition*, pages 16–17, 2017.

[5] P. Baroni, M. Caminada, and M. Giacomin. An introduction to argumentation semantics. *The knowledge engineering review*, 26(4):365–410, 2011.

[6] P. Baroni, M. Caminada, and M. Giacomin. Abstract argumentation frameworks and their semantics. In P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, editors, *Handbook of Formal Argumentation*, pages 159–236. College Publications, 2018.

[7] A. Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):75–97, 2008.

[8] A. Biere. Cadical at the sat race 2019. *SAT RACE 2019*, page 8, 2019.

[9] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185. IOS Press, 2009.

[10] S. Cai, C. Luo, and K. Su. Ccanr: A configuration checking based local search solver for non-random satisfiability. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–8, 2015.

[11] S. Cai and X. Zhang. Four relaxed CDCL Solvers. *SAT RACE 2019*, page 35, 2019.

[12] Martin W.A. Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2–3):109–145, 2009.

[13] F. Cerutti, P. E. Dunne, M. Giacomin, and M. Vallati. Computing preferred extensions in abstract argumentation: A sat-based approach. In *International Workshop on Theorie and Applications of Formal Argumentation*, pages 176–193. Springer, 2013.

[14] F. Cerutti, M. Giacomin, and M. Vallati. Argsemsat: Solving argumentation problems using sat. *COMMA*, 14:455–456, 2014.

[15] F. Cerutti, M. Giacomin, and M. Vallati. How we designed winning algorithms for abstract argumentation and which insight we attained. *Artificial Intelligence*, 276:1–40, 2019.

[16] F. Cerutti, M. Vallati, and M. Giacomin. jargsemsat: an efficient off-the-shelf solver for abstract argumentation frameworks. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 541–544, 2016.

[17] J. Chen. Smallsat, Optsat and MapleLCMdistCBTcoreFirst: Containing Core First Unit Propagation. *SAT RACE 2019*, page 31, 2019.

[18] P. M. Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2):321–358, 1995.

[19] W. Dvořák and P. E. Dunne. Computational problems in formal argumentation and their complexity. In P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14. 2018.

[20] N. Eén and N. Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.

[21] N. Eén and N. Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518, 2003.

[22] U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, 1(2):147–177, 2010.

[23] S. A. Gaggl, T. Linsbichler, M. Maratea, and S. Woltran. Design and results of the second international competition on computational models of argumentation. *Artificial Intelligence*, 279:103193, 2020.

[24] M. Järvisalo, M. J. H. Heule, and A. Biere. Inprocessing rules. In *International Joint Conference on Automated Reasoning*, pages 355–370. Springer, 2012.

[25] S. Kochemazov, O. Zaikin, V. Kondratiev, and A. Semenov. MapleLCMDistChronoBT-DL, duplicate learnts heuristic-aided solvers at the SAT Race 2019. *SAT RACE 2019*, page 24, 2019.

[26] N. Manthey. Mergesat. *Proceedings of SAT Competition*, 2019:29, 2019.

[27] J. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning sat solvers. In *Handbook of satisfiability*, pages 131–153. IOS Press, 2009.

[28] A. Nadel and V. Ryvchin. Chronological backtracking. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 111–121, 2018.

[29] A. Niskanen and M. Järvisalo. μ-toksia Participating in ICCMA 2019. In *The Third International Competition on Computational Models of Argumentation (ICCMA'19)*, 2019.

[30] F. Pu, H. Ya, and G. Luo. argmat-sat: Applying sat solvers for argumentation problems based on boolean matrix algebra. In *The Second International Competition on Computational Models of Argumentation (ICCMA'17)*, 2017.

[31] R. K. Tchinda and C. T. Djamegni. PADC MapleLCMDistChronoBT, PADC Maple LCM Dist and PSIDS MapleLCMDistChronoBT in the SR19. *SAT RACE 2019*, page 33, 2019.

[32] M. Thimm and S. Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, August 2017.