University of Huddersfield Repository

Cerutti, Federico, Vallati, Mauro and Giacomin, Massimiliano

An Efficient Java-Based Solver for Abstract Argumentation Frameworks: jArgSemSAT

Original Citation


This version is available at http://eprints.hud.ac.uk/id/eprint/29334/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/
An Efficient Java-Based Solver for Abstract Argumentation Frameworks: jArgSemSAT

Federico Cerutti, Mauro Vallati, Massimiliano Giacomin

September 8, 2016

Abstract

Dung’s argumentation frameworks are adopted in a variety of applications, from argument-mining, to intelligence analysis and legal reasoning. Despite this broad spectrum of already existing applications, the mostly adopted solver—in virtue of its simplicity—is far from being comparable to the current state-of-the-art solvers. On the other hand, most of the current state-of-the-art solvers are far too complicated to be deployed in real-world settings. In this paper we provide an extensive description of jArgSemSAT, a Java re-implementation of ArgSemSAT. ArgSemSAT represents the best single solver for argumentation semantics with the highest level of computational complexity. We show that jArgSemSAT can be easily integrated in existing argumentation systems (1) as an off-the-shelf, standalone, library; (2) as a Tweety compatible library; and (3) as a fast and robust web service freely available on the Web. Our large experimental analysis shows that—despite being written in Java—jArgSemSAT would have scored in most of the cases among the three best solvers for the two semantics with highest computational complexity—Stable and Preferred—in the last competition on computational models of argumentation.

1 Introduction

Dung’s theory of abstract argumentation[18] is a unifying framework able to encompass a large variety of specific formalisms in the areas of nonmonotonic reasoning, logic programming and computational argumentation. It is based on the notion of argumentation framework (AF), that consists of a set of arguments and an attack relation between them. Different argumentation semantics introduce in a declarative way the criteria to determine which arguments emerge as “justified” from the conflict, by identifying a number of extensions, i.e. sets of arguments that can “survive the conflict together”. In Dung’s paper[18] three “traditional” semantics are introduced, namely grounded, stable, and preferred semantics, as well as the auxiliary notion of complete extension, to highlight the linkage between grounded and preferred semantics. Other literature proposals include semi-stable[10], ideal[17], and CF2[5] semantics.
The preferred semantics represents one of the main contributions in Dung’s theory[18] and is widely adopted—among other areas—in decision support systems[31] and in critical thinking support systems[35], as it allows multiple extensions (differently from grounded semantics), the existence of extensions is always guaranteed (differently from stable semantics), and no extension is a proper subset of another extension. The investigation on alternative argumentation semantics is an active research area since two decades[2].

Many problems associated to preferred, but also to stable, semantics turn to be at the high levels of the polynomial hierarchy[19]. In this paper we will focus on four problems, namely credulous and skeptical acceptance of an argument with respect to a given argumentation framework and a given semantics and enumeration of all or some semantics extensions given an argumentation framework. Those are the problems considered in the first International Competition on Computational Models of Argumentation (ICCMA2015) that determined the state-of-the-art of the current implementations for addressing the above problems with respect to the three aforementioned semantics (plus the complete extensions).

Surprisingly, the winner of ICCMA2015—CoQuiAAS1[24]—never scored in the first two positions with respect to the most computationally expensive semantics, namely stable and preferred semantics. Indeed, CoQuiAAS performed very well on grounded semantics—where each problem is polynomial—thanks to a very efficient unit propagation mechanism, as well as on the tracks associated to complete extensions problems. The interested reader is referred to the competition summary[34] and to the competition website2 for an overview of the results.

Instead, ArgSemSAT is the best single solver when facing semantics with an high level of computational complexity. It is constantly either first or second placed in each track associated to stable and preferred semantics—except one due to an implementation bug discovered after the competition3. Despite this bug, ArgSemSAT scored second—at one single Borda count point from CoQuiAAS—and the overhead in solving problems associated to grounded semantics is—on average—of 3.88 seconds from CoQuiASS. Therefore, this difference is neglectable in most real-world situation, when ultimately a human user will consume the result of argumentation-based reasoning procedures.

Building on top of the success of ArgSemSAT, we introduced jArgSemSAT that is specifically designed for being easily integrated within existing argumentation systems. Indeed, ArgSemSAT—as well as many other solvers that participated in ICCMA2015—is written in C++ and requires an external SAT solver as an NP oracle. This can hardly be considered an off-the-shelf system, as most of the current tools using argumentation technology are based on existing Java approaches (such as Dung-O-Matic[30], adopted e.g. in CISpaces[35]), or on the Tweety libraries for knowledge representation and reasoning[32], or use

---

1http://www.cril.univ-artois.fr/coquiaas/
2http://argumentationcompetition.org/2015/
3Details can be found in http://downloads.sourceforge.net/project/argsemsat/ArgSemSAT-1.0rc3/ArgSemSAT_1.0rc3.zip
a web-service interface as ArgTech[8]. We developed jArgSemSAT in Java, with a specific focus on being compatible with Dung-O-Matic, Tweety, and with a web-service interface in turn compatible with ArgTech. A large experimental analysis confirms that jArgSemSAT—despite being written in Java—would have been one of the best solvers for most of the ICCMA2015 tracks associated to the two semantics with highest computational complexity. Therefore, not only ArgSemSAT is compatible with existing technology, but it is also among the best solvers for stable and preferred semantics.

This paper is an extension of the short report from the field work presented at the 15th Conference on Principles of Knowledge Representation and Reasoning[16] with substantial additional material, including: (i) a complete description of jArgSemSAT which includes a thorough description of the implemented algorithms and processes; (ii) a significantly extended experimental analysis, that considers a comparison with the state of the art—including solvers which took part in ICCMA2015—on several problems associated to stable and preferred semantics; and (iii) an extensive discussion of the benefits of employing jArgSemSAT within CISpaces[35].

The rest of the paper is organised as follows. Section 2 provides the required background on abstract argumentation; Section 3 gives an overview of the jArgSemSAT system, while Section 4 focuses on system design; Section 5 reports our experimental results; finally, conclusions and discussions of the benefits of jArgSemSAT are given in Section 6.

2 Background

2.1 Argumentation frameworks and semantics

An argumentation framework[18] consists of a set of arguments and a binary attack relation between them.

**Definition 1** An argumentation framework (AF) is a pair \( \Gamma = (\mathcal{A}, \mathcal{R}) \) where \( \mathcal{A} \) is a set of arguments and \( \mathcal{R} \subseteq \mathcal{A} \times \mathcal{A} \). We say that \( b \) attacks \( a \) iff \( (b, a) \in \mathcal{R} \), also denoted as \( b \rightarrow a \). The set of attackers of an argument \( a \) will be denoted as \( a^- \equiv \{ b : b \rightarrow a \} \), the set of arguments attacked by \( a \) will be denoted as \( a^+ \equiv \{ b : a \rightarrow b \} \). We also extend these notations to sets of arguments, i.e. given \( E \subseteq \mathcal{A} \), \( E^- \equiv \{ b : \exists a \in E, b \rightarrow a \} \) and \( E^+ \equiv \{ b : \exists a \in E, a \rightarrow b \} \).

An argument \( a \) without attackers, i.e. such that \( a^- = \emptyset \), is said initial. Moreover, each argumentation framework has an associated directed graph where the vertices represent the arguments, and the edges represent the attacks.

The basic properties of conflict–freeness, acceptability, and admissibility of a set of arguments are fundamental for the definition of argumentation semantics.

**Definition 2** Given an AF \( \Gamma = (\mathcal{A}, \mathcal{R}) \):

\(^4\)In this paper we consider only finite sets of arguments: see [4] for a discussion on infinite sets of arguments.
• a set $S \subseteq A$ is a conflict–free set of $\Gamma$ if $\nexists a, b \in S$ s.t. $a \rightarrow b$;

• an argument $a \in A$ is acceptable with respect to a set $S \subseteq A$ of $\Gamma$ if $\forall b \in A$ s.t. $b \rightarrow a$, $\exists c \in S$ s.t. $c \rightarrow b$;

• the function $F_\Gamma : 2^A \rightarrow 2^A$ such that $F_\Gamma(S) = \{ a \mid a$ is acceptable w.r.t. $S \}$ is called the characteristic function of $\Gamma$;

• a set $S \subseteq A$ is an admissible set of $\Gamma$ if $S$ is a conflict–free set of $\Gamma$ and every element of $S$ is acceptable with respect to $S$, i.e. $S \subseteq F_\Gamma(S)$;

• a set $S \subseteq A$ is a complete extension of $\Gamma$ if $S$ is an admissible set of $\Gamma$ such that it contains each argument acceptable with respect to $S$, i.e. $S = F_\Gamma(S)$.

An argumentation semantics $\sigma$ prescribes for any AF $\Gamma$ a set of extensions, denoted as $E_{\sigma}(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by $\sigma$. Here we need to recall the definitions of grounded (denoted as GR), stable (denoted as ST), and preferred (denoted as PR) semantics only.

**Definition 3** Given an AF $\Gamma = \langle A, \mathcal{R} \rangle$:

• a set $S \subseteq A$ is the grounded extension of $\Gamma$, i.e. $S \in E_{\text{GR}}(\Gamma)$, iff $S$ is the least fixed point of $F_\Gamma$;

• a set $S \subseteq A$ is a stable extension of $\Gamma$, i.e. $S \in E_{\text{ST}}(\Gamma)$, iff $S$ is a conflict-free set of $\Gamma$ and $S \cup S^+ = A$;

• a set $S \subseteq A$ is a preferred extension of $\Gamma$, i.e. $S \in E_{\text{PR}}(\Gamma)$, iff $S$ is a maximal (w.r.t. set inclusion) admissible set of $\Gamma$.

The notion of complete extension has been introduced[18] as a linkage between preferred and grounded semantics. Given an AF $\Gamma = \langle A, \mathcal{R} \rangle$, a set $S \subseteq A$ is a complete extension of $\Gamma$ iff $S$ is a conflict-free set of $\Gamma$ and $S = F_\Gamma(S)$. The auxiliary notion of complete extension provides a mean for re-defining the grounded extension as the minimal (with respect to set inclusion) complete extension, and a preferred extension as a maximal (w.r.t. set inclusion) complete extension.

Each extension $S$ implicitly defines a three-valued labelling of arguments: an argument $a$ is labelled $\text{in}$ iff $a \in S$; is labelled $\text{out}$ iff $\exists b \in S$ s.t. $b \rightarrow a$; is labelled $\text{undec}$ if neither of the above conditions holds. In the light of this correspondence, argumentation semantics can be equivalently defined in terms of labellings rather than of extensions[9, 2].

**Definition 4** Given a set of arguments $S$, a labelling of $S$ is a total function $\text{Lab} : S \rightarrow \{ \text{in, out, undec} \}$. The set of all labellings of $S$ is denoted as $L_S$. Given an AF $\Gamma = \langle A, \mathcal{R} \rangle$, a labelling of $\Gamma$ is a labelling of $A$. The set of all labellings of $\Gamma$ is denoted as $L(\Gamma)$.

In particular, complete labellings can be defined as follows.
Definition 5 Let $\Gamma = \langle A, R \rangle$ be an argumentation framework. A labelling $\text{Lab} \in \Sigma(\Gamma)$ is a complete labelling of $\Gamma$ iff it satisfies the following conditions for any $a \in A$:

- $\text{Lab}(a) = \text{in} \iff \forall b \in a^-> \text{Lab}(b) = \text{out}$;
- $\text{Lab}(a) = \text{out} \iff \exists b \in a^- : \text{Lab}(b) = \text{in}$;

The grounded, stable, and preferred labelling can then be defined on the basis of complete labellings.

Definition 6 Let $\Gamma = \langle A, R \rangle$ be an argumentation framework. A labelling $\text{Lab} \in \Sigma(\Gamma)$ is

- the grounded labelling of $\Gamma$ if it is the complete labelling of $\Gamma$ maximising the set of arguments labelled $\text{undec}$;
- a stable labelling of $\Gamma$ if it is a complete labelling of $\Gamma$ and there is no argument labelled $\text{undec}$;
- a preferred labelling of $\Gamma$ if it is a complete labelling of $\Gamma$ maximizing the set of arguments labelled $\text{in}$.

In order to show the connection between extensions and labellings, let us recall[2] the definition of the function $\text{Ext2Lab}$, returning the labelling corresponding to a conflict–free set of arguments $S$.

Definition 7 Given an AF $\Gamma = \langle A, R \rangle$ and a conflict–free set $S$ of $\Gamma$, the corresponding labelling $\text{Ext2Lab}(S)$ is the labelling of $\Gamma$ $\text{Lab}$ where

- $\text{Lab}(a) = \text{in} \iff a \in S$
- $\text{Lab}(a) = \text{out} \iff \exists b \in S \text{ s.t. } b \rightarrow a$
- $\text{Lab}(a) = \text{undec} \iff a \notin S \land \exists b \in S \text{ s.t. } b \rightarrow a$

There is a bijective correspondence between the complete, grounded, stable, preferred extensions and the complete, grounded, stable, preferred labellings, respectively.[9]

Proposition 1 Given an an AF $\Gamma = \langle A, R \rangle$, $\text{Lab}$ is a complete (grounded, stable, preferred) labelling of $\Gamma$ if and only if there is a complete (grounded, stable, preferred) extension $S$ of $\Gamma$ such that $\text{Lab} = \text{Ext2Lab}(S)$.

2.2 Computational Problems in Abstract Argumentation

Credulous and skeptical acceptance of an argument are the two most studied decision problems in argumentation theory (see [19]).

An argument $a$ is credulously accepted with respect to a given semantics $\sigma$ and a given AF $\Gamma$ iff $a$ belongs to at least one extension of $\Gamma$ under $\sigma$: $\exists E \in \mathcal{E}_\sigma(\Gamma)$
Table 1: Computational complexity of credulous and skeptical acceptance in finite AFs w.r.t. the three semantics introduced by Dung.

<table>
<thead>
<tr>
<th>Semantics σ</th>
<th>GR</th>
<th>ST</th>
<th>PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC-σ</td>
<td>in P</td>
<td>NP-compl.</td>
<td>NP-compl.</td>
</tr>
<tr>
<td>DS-σ</td>
<td>in P</td>
<td>coNP-compl.</td>
<td>Π²_p-compl.</td>
</tr>
</tbody>
</table>

s.t. \(a \in E\). We denote such a problem as DC-σ. An argument \(a\) is skeptically accepted with respect to a given semantics \(σ\) and a given AF \(Γ\) iff \(a\) belongs to each extension of \(Γ\) under \(σ\): \(∀E \in \mathcal{E}_σ(Γ) \ a \in E\). We denote this problem as DS-σ.

The complexity of DC-σ and DS-σ when \(σ\) is the stable or preferred semantics lies at the first or second level of the polynomial hierarchy, as shown in Table 1 (see [19]).

In addition to credulous and skeptical acceptance, the following two problems are worth considering and have been included in ICCMA2015:

- given an AF, determine some extension (SE) of a given semantics;
- given an AF, determine all extensions (EE) of a given semantics.

As shown in Table 2, ArgSemSAT is always in the first two positions both in the case of stable and preferred semantics, except for DS-ST due to an implementation bug discovered after the competition.

ArgSemSAT scored second considering the Borda count across all tracks of ICCMA2015, at one point of distance from CoQuiAAS, which scored at most third in the tracks associated to stable and preferred semantics, but was constantly the best for grounded semantics. For this semantics, CoQuiAAS uses an efficient unit propagation mechanism, while ArgSemSAT searches for it via a maximisation process in the space of complete labellings. Despite this massive difference in the approaches, the difference of execution times between CoQuiAAS and ArgSemSAT over the competition benchmark and with respect to the four tracks related to grounded semantics is of 3.88 seconds on average (standard deviation 5.89).

---

5Computational complexity for credulous and skeptical acceptance w.r.t. admissible sets, as well as w.r.t. complete extensions can easily been identified, cf. [19]. In particular, credulous acceptance w.r.t. admissible sets is trivial, and skeptical acceptance w.r.t. complete extensions is equivalent to acceptance w.r.t. grounded semantics.

6http://argumentationcompetition.org/2015/results.html
Table 2: First three places in ICCMA2015 with respect to stable and preferred semantics.

<table>
<thead>
<tr>
<th>Semantics σ</th>
<th>ST</th>
<th>PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC-σ</td>
<td>1.ASPARTIX-D</td>
<td>1.ArgSemSAT</td>
</tr>
<tr>
<td></td>
<td>2.ArgSemSAT</td>
<td>2.LabSATSolver</td>
</tr>
<tr>
<td></td>
<td>3. LabSATSolver</td>
<td>3. CoQuiAAS</td>
</tr>
<tr>
<td>DS-σ</td>
<td>1. ASPARTIX-D</td>
<td>1. ArgSemSAT</td>
</tr>
<tr>
<td></td>
<td>2. LabSATSolver</td>
<td>2. Cegartix</td>
</tr>
<tr>
<td></td>
<td>3. CoQuiAAS</td>
<td>3. LabSATSolver</td>
</tr>
<tr>
<td>SE-σ</td>
<td>1. ASPARTIX-D</td>
<td>1. Cegartix</td>
</tr>
<tr>
<td></td>
<td>2. ArgSemSAT</td>
<td>2. ArgSemSAT</td>
</tr>
<tr>
<td></td>
<td>3. LabSATSolver</td>
<td>3. LabSATSolver</td>
</tr>
<tr>
<td>EE-σ</td>
<td>1. ASPARTIX-D</td>
<td>1. Cegartix</td>
</tr>
<tr>
<td></td>
<td>2. ArgSemSAT</td>
<td>2. ArgSemSAT</td>
</tr>
<tr>
<td></td>
<td>3. CoQuiAAS</td>
<td>3. CoQuiAAS</td>
</tr>
</tbody>
</table>
3 Overview of jArgSemSAT

jArgSemSAT and ArgSemSAT enumerate preferred extensions by multiple calls to a SAT solver[13]. A propositional formula over a set of Boolean variables is satisfiable iff there exists a truth assignment of the variables such that the formula evaluates to True. Checking whether such an assignment exists is the satisfiability (SAT) problem. jArgSemSAT and ArgSemSAT exploit an encoding of complete extensions as a propositional formula in Conjunctive Normal Form (CNF) and apply a filtering procedure over the space of complete extensions to select the maximal ones, i.e. the preferred extensions.

There are therefore three main components within jArgSemSAT and ArgSemSAT: (1) a propositional formula for complete labelling—cf. Definition 5; (2) an NP-oracle in the form of a SAT solver; and (3) a filtering process.

3.1 Propositional formulae for complete labellings

Given an AF $\Gamma = (A, R)$ we identify a propositional formula $\Pi_\Gamma$ such that each satisfying assignment of the formula corresponds to a complete labelling. Such formula is based on Definition 5, which however admits several logically equivalent propositional encodings that lead to severely different performance[13].

As a first step to explore alternative encodings, the conditions in Definition 5 can be redundantly expressed as a conjunction of 6 terms, i.e. $C_{in}^\rightarrow \land C_{in}^\leftarrow \land C_{out}^\rightarrow \land C_{out}^\leftarrow \land C_{undec}^\rightarrow \land C_{undec}^\leftarrow$, where

- $C_{in}^\rightarrow \equiv \{\text{Lab}(a) = \text{in} \Rightarrow \forall b \in a^- \text{Lab}(b) = \text{out}\};$
- $C_{in}^\leftarrow \equiv \{\text{Lab}(a) = \text{in} \Leftrightarrow \forall b \in a^- \text{Lab}(b) = \text{out}\};$
- $C_{out}^\rightarrow \equiv \{\text{Lab}(a) = \text{out} \Rightarrow \exists b \in a^- : \text{Lab}(b) = \text{in}\};$
- $C_{out}^\leftarrow \equiv \{\text{Lab}(a) = \text{out} \Leftarrow \exists b \in a^- : \text{Lab}(b) = \text{in}\};$
- $C_{undec}^\rightarrow \equiv \{\text{Lab}(a) = \text{undec} \Rightarrow \forall b \in a^- \text{Lab}(b) \neq \text{in} \land \exists c \in a^- : \text{Lab}(c) = \text{undec}\};$
- $C_{undec}^\leftarrow \equiv \{\text{Lab}(a) = \text{undec} \Leftarrow \forall b \in a^- \text{Lab}(b) \neq \text{in} \land \exists c \in a^- : \text{Lab}(c) = \text{undec}\}.$

Moreover we define $C_{in}^{\leftrightarrow} \equiv C_{in}^\rightarrow \land C_{in}^\leftarrow$, $C_{out}^{\leftrightarrow} \equiv C_{out}^\rightarrow \land C_{out}^\leftarrow$, $C_{undec}^{\leftrightarrow} \equiv C_{undec}^\rightarrow \land C_{undec}^\leftarrow$.

We identify[13, Proposition 2] 5 non redundant strict subsets of the above six terms that equivalently characterize complete extensions\footnote{The terms $C_{in}^{\leftrightarrow} \land C_{out}^{\leftrightarrow}$ and $C_{in}^\rightarrow \land C_{out}^\leftarrow \land C_{undec}^\rightarrow$ correspond to the alternative definitions of complete labellings in [11], where a proof of their equivalence is provided.}, namely: (i) $C_{in}^\rightarrow \land C_{out}^\leftarrow$; (ii) $C_{out}^\rightarrow \land C_{undec}^\rightarrow$; (iii) $C_{in}^\rightarrow \land C_{undec}^\leftarrow$; (iv) $C_{in}^\rightarrow \land C_{out}^\rightarrow \land C_{undec}^\rightarrow$; (v) $C_{in}^\rightarrow \land C_{out}^\leftarrow \land C_{undec}^\rightarrow$.

SAT solvers require such constraints in conjunctive normal form (CNF). Letting $k = |A|$ we define a bijection $\phi : \{1, \ldots, k\} \mapsto A$ (the inverse map is denoted as $\phi^{-1}$). $\phi$ is an indexing of $A$: for sake of brevity we might refer to the argument

\[ \phi^{-1}(\phi(a)) \]
φ(i) as “argument i.” For each argument i we define three Boolean variables, I_i, O_i, and U_i, with the intended meaning that I_i (resp. O_i, U_i) is True when argument i is labelled in (resp. out, undec), False otherwise. Given Γ = ⟨A, R⟩ we define the corresponding set of variables as V(Γ) ≜ ∪_{1≤i≤|A|} {I_i, O_i, U_i}.

The conjunction of the following formulae in CNF format is equivalent to C_{in}^{++} ∧ C_{out}^{++} ∧ C_{undec}^{++}:

\[
\bigwedge_{i\in\{1,\ldots,k\}} (I_i \lor O_i \lor U_i) \land (\neg I_i \lor \neg O_i)
\land (\neg I_i \lor \neg U_i) \land \neg O_i \land \neg U_i
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} I_i \land \neg O_i \land \neg U_i
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} \neg I_i \lor O_j
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} I_i \lor \bigvee_{\{j|\phi(j)\rightarrow\phi(i)\}} (\neg O_j)
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} I_j
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} I_j \lor O_i
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} \neg U_i \lor \neg I_j
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} \neg U_i \lor \neg I_j
\]

\[
\bigwedge_{\{i|\phi(i)\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} U_i \lor \neg U_k \lor \bigvee_{\{j|\phi(j)\rightarrow\phi(i)\}} I_j
\]

where (1) ∧ (3) ≡ C_{in}^{++}; (1) ∧ (2) ∧ (4) ≡ C_{out}^{++}; (1) ∧ (5) ≡ C_{undec}^{++}; (1) ∧ (6) ≡ C_{out}^{++}; (1) ∧ (7) ≡ C_{undec}^{++}; (1) ∧ (8) ≡ C_{undec}^{++}.

Users can choose the desired encoding of complete labellings, i.e. the formula Π_Γ, by specifying a sequence of 6 Boolean values—0 for ⊥ and 1 for ⊤, corresponding to the sequence (C_{in}, C_{in}, C_{out}, C_{out}, C_{undec}, C_{undec}). For instance, the sequence 101010 identifies C_{in}^{++} ∧ C_{out}^{++} ∧ C_{undec}^{++}. Incorrect configurations that do not correspond to encodings of complete labellings—e.g. 000000—are discarded and the user receives an error message.
3.2 SAT solvers

The second component of jArgSemSAT and ArgSemSAT is an NP-oracle in the form of a SAT solver, whose efficiency can be very sensible to the chosen encoding[13]. Following the experience of ArgSemSAT, jArgSemSAT allows the user to choose any desired SAT solver—whose full path must be provided—that supports the DIMACS format, accepts a CNF from the STDIN, and returns a model to the STDOUT. For instance, jArgSemSAT can—as ArgSemSAT does—use GLUCOSE[1] as an external SAT solver.

In order to provide an off-the-shelf solver, jArgSemSAT also integrates as a library Sat4j[25]. Sat4j is an open source library which allows Java programmers to access cross-platform SAT-based solvers. The Sat4j library project started in 2004 as an implementation in Java of the MiniSAT specification[21]. It has been developed with the spirit to keep the technology easily accessible to a newcomer. For instance, it allows the Java programmer to express constraints on objects and hides all the mapping to the various research community input formats from the user.

By default, jArgSemSAT utilises Sat4j with the encoding 111100—equivalent to $C_{in}^{+} \land C_{out}^{+}$—that on average performs best on MiniSAT-based approaches[13].

3.3 Filtering process

Computing grounded, stable and preferred labellings is then a question of implementing efficient filters of complete labellings (see Definition 6) that can be computed as a SAT assignment of a propositional formula $\Pi_1$ variables.

Stable labellings—i.e. complete labellings with no undec arguments—are the solutions to the formula $\Pi'_1 := \Pi_1 \land \bigwedge_{a \in A} \neg U_{\phi^{-1}(a)}$. Each time the SAT solver finds a solution sol, the formula $\Pi'_1$ is updated to $\Pi'_1 \land \neg sol$ and the SAT solver is called on $\Pi'_1$ in order to find an additional stable labelling. The process is iterated until the SAT solver returns no solution, thus enumerating all stable labellings.

Preferred extensions are computed as per Algorithm 1, that is an—unpublished—evolution of the algorithm presented in previous work[13].

Algorithm 1 consists of two nested loops. The external one—lines 5–22—iterates over a (sub)set of complete labellings to identify preferred labellings, while the internal one—lines 8–17—performs an optimisation procedure on a complete labelling to maximise the set of in-labelled arguments. Algorithm 1 uses four auxiliary functions. SatS refers to a SAT solver able to prove unsatisfiability too: it accepts as input a CNF formula and returns a variable assignment satisfying the formula if it exists, $\varepsilon$ otherwise. I-ARGS (resp. O-ARGS, U-ARGS) accepts as input a variable assignment concerning $\mathcal{V}(\Gamma)$ and returns the corresponding set of arguments labelled as in (resp. out, undec).

There are two variables that play a pivotal role in Algorithm 1: cnf and cnfdf. The former, cnf, keeps track of the complete labellings already visited,
Algorithm 1 Enumeration of Preferred Extensions

1: Input: $\Gamma = \langle A, R \rangle$
2: Output: $E_p \subseteq 2^A$
3: $E_p := \emptyset$
4: $cnf := \Pi_\Gamma \land \bigvee_{a \in A} I_{\phi^{-1}(a)}$
5: repeat
6: $cnfdf := cnf$
7: $prefcand := \emptyset$
8: repeat
9: $aCompl := \text{SatS}(cnfdf)$
10: if $aCompl \neq \varepsilon$ then
11: $prefcand := aCompl$
12: if $U-\text{ARGS}(aCompl) \neq \emptyset$ then
13: $cnfdf := cnfdf \land \bigwedge_{a \in I-\text{ARGS}(aCompl)} I_{\phi^{-1}(a)} \land \bigwedge_{a \in O-\text{ARGS}(aCompl)} O_{\phi^{-1}(a)} \land \bigvee_{a \in U-\text{ARGS}(aCompl)} I_{\phi^{-1}(a)}$
14: end if
15: $cnf := cnf \land \bigvee_{a \in A \setminus I-\text{ARGS}(aCompl)} I_{\phi^{-1}(a)}$
16: end if
17: until ($aCompl \neq \varepsilon \land U-\text{ARGS}(aCompl) \neq \emptyset$)
18: if $prefcand \neq \emptyset$ then
19: $E_p := E_p \cup \{ I-\text{ARGS}(prefcand) \}$
20: $cnf := cnf \land \neg \left( \bigwedge_{a \in I-\text{ARGS}(prefcand)} I_{\phi^{-1}(a)} \land \bigwedge_{a \in O-\text{ARGS}(prefcand)} O_{\phi^{-1}(a)} \land \bigwedge_{a \in U-\text{ARGS}(prefcand)} U_{\phi^{-1}(a)} \right)$
21: end if
22: until ($prefcand \neq \emptyset$)
23: if $E_p = \emptyset$ then
24: $E_p = \{\emptyset\}$
25: end if
26: return $E_p$
and thus affects both loops. The latter, \( cnfdf \), keeps track of the search within the optimisation process, thus affecting the inner loop only.

At first, \( cnf \) is initialised (l. 4) to \( \Pi_{\Gamma} \land \bigvee_{a \in A} I_{\phi^{-1}(a)} \): \( \emptyset \) is excluded since it is always admissible. At l. 6 \( cnfdf \) is initialised to the value of \( cnf \) and, after entering the inner loop, the SAT solver is called over \( cnfdf \) returning a complete labelling \( aCompl \) (l. 9). This is a candidate to become a preferred labelling until either (i) a “bigger” complete labelling containing \( aCompl \) is found; or (ii) it is proven that there are no further complete labellings containing \( aCompl \). To search for (i), at line 13 each \( \text{in} \)-labelled \( (\text{out} \)-labelled) argument in \( aCompl \) is forced to be labelled \( \text{in} \) \( (\text{out} \) until a preferred labelling is found, and to guide future searches towards a strictly bigger labelling at least one more argument is enforced to be labelled \( \text{in} \).

The inner loop is exited when either (i) \( \text{SatS} \) returns no solution \( (\varepsilon) \); or (ii) \( I-ARGS(aCompl) \cup O-ARGS(aCompl) = A \). In the first case, the \( \text{prefcand} \) preferred labelling candidate found at the previous iteration cannot be extended to a complete labelling including a greater set of \( \text{in} \)-labelled arguments, thus it is a preferred labelling\(^9\). In the second case, this is due to the fact that there are no undecided arguments.

Once a preferred labelling is found, the set of preferred extensions is enriched (l. 19) with \( I-ARGS(\text{prefcand}) \) and \( \neg \text{prefcand} \) is added as a further constraint (line 20) within \( cnf \) before executing the external loop once again.

The implemented procedure for computing the grounded labelling is analogous, with the difference that the set of \( \text{undec} \)-labelled arguments—instead of the set of \( \text{in} \)-labelled arguments—is maximised in the inner loop.

As per checking the credulous acceptance of an argument \( x \) w.r.t. the:

- grounded semantics: \( \text{jArgSemSAT} \) checks whether \( x \) is in the set of \( \text{in} \)-labelled arguments of the grounded labelling;

- stable semantics: \( \text{jArgSemSAT} \) checks whether there is a solution to the formula \( \Pi_{\Gamma} \land I_{\phi^{-1}(x)} \);

- preferred semantics: \( \text{jArgSemSAT} \) checks whether there is a solution to \( \Pi_{\Gamma} \land I_{\phi^{-1}(x)} \), which implies that there is a maximum labelling—i.e. a preferred labelling—containing that argument.

As per checking the skeptical acceptance of an argument \( x \) w.r.t. the:

- grounded semantics: since the grounded extension is unique, it is equivalent to check the credulous acceptance of \( x \);

- stable semantics:

  1. if there is a solution to the formula \( \Pi_{\Gamma} \land O_{\phi^{-1}(x)} \) (equivalent to the question: does a stable labelling where \( x \) is \( \text{out} \) exist?) then \( \text{jArgSemSAT} \) returns False;

\(^9\)In the case this happens at the first execution, it means that \( \emptyset \) is the only preferred extension, cf. lines 23–24.
2. at this point, if there is a solution to the formula $\Pi'_\Gamma$ (i.e. there exists at least a stable extension), then $x$ belongs to the $\text{in}$-labelled sets for each stable labellings, and $j\text{ArgSemSAT}$ returns True (otherwise it returns False);

- preferred semantics: $j\text{ArgSemSAT}$ checks within Algorithm 1 whether $x$ is not in the set of $\text{in}$-labelled arguments of a found preferred labelling and returns False in this case. $j\text{ArgSemSAT}$ returns True otherwise.

4 System design

$j\text{ArgSemSAT}$ is a mature application that now exists in four different versions:

1. Stand-alone application: this guarantees compatibility with the Probo interface for the International Competition on Computational Models of Argumentation (ICCMA)[15];

2. Dung-O-Matic (DoM)[30] compatible library: this ensures compatibility for works already using DoM such as CISpaces[35];

3. Tweety compatible library[32]: we proudly support the Tweety project whose aim is to provide a general framework for implementing and testing knowledge representation formalisms;

4. ArgTech [8] compatible web-service: we created a Tomcat web-service exporting $j\text{ArgSemSAT}$ with ArgTech-compatible RESTful interfaces.

$j\text{ArgSemSAT}$ is freely (MIT licence) available on SourceForge\(^\text{10}\) and as Maven projects directly accessible from the central repository\(^\text{11}\). It is composed by two jar files and a war file.

$j\text{ArgSemSAT-VERSION.jar}$ provides both the stand-alone application compatible with the Probo interface and the DoM compatible library: we chose not to distribute the library without the Probo interface to facilitate future experiments also from different research groups and to improve the awareness in the community of the ICCMA competition.

Figure 1 depicts the UML graph of the main classes included in the net.sf.jargsemsat.jarsemsat.alg, namely those implementing the algorithms for computing complete, grounded, preferred, stable, and semi-stable\(^\text{12}\) extensions. In particular, two methods are particularly important in CompleteSemantics class: basicComplete and satlab.

basicComplete computes—depending on the parameter Encoding passed to it—the propositional formula for complete labellings among all those introduced in Section 3.1. Instead, satlab is the method which deals with SAT solvers: it

\(^\text{10}\)https://sourceforge.net/projects/jargsemsat/
\(^\text{11}\)http://search.maven.org/
\(^\text{12}\)semi-stable semantics implementation is still experimental and, as such, not described in this document.
Figure 1: UML diagram of the core components of jArgSemSAT
requires as input an object of type SATFormulae, an empty Labelling to store the result of the computation, and a DungAF.\textsuperscript{13} It returns true if a satisfiable assignment is found, false otherwise.

jArgSemSATTweety-VERSION.jar is a self-contained, Tweety-compatible, library: it includes jArgSemSAT-VERSION.jar and provides a Tweety-compatible interface.

jArgSemSATWeb-VERSION.war is a self-contained Tomcat\textsuperscript{14} web-service archive compatible with ArgTech\textsuperscript{15} specifications. This web-service is also available free-of-charge—with best effort SLA—at http://cicero.cs.cf.ac.uk/jArgSemSATWeb/restapi/argtech/ . Its source code is also freely available.

\subsection{Stand-alone application}

jArgSemSAT exports the same command line interface of ArgSemSAT, which is a superset of the Probo interface. In addition to the options discussed in previous work\cite{15}, jArgSemSAT allows the user to choose (1) the SAT solver to be used—Sat4j is the default; and (2) the encoding to use—111100, equivalent to $C^\text{in} \land C^\text{out}$, is the default.

\subsection{Dung-O-Matic (DoM) compatible library}

jArgSemSAT exports methods whose signature are compatible with Dung-O-Matic\cite{30}: those methods encapsulate the code for calling jArgSemSAT with the default configurations, and on data-structures that reside on memory instead on a file.

Therefore, the following snippet code:

\begin{verbatim}
Vector<String> args = new Vector<String>();
    args.add("a");
    args.add("b");
Vector<String []> atts = new Vector<String []>();
    atts.add(new String []{"a", "b");
new DungAF(args, atts).getStableExts();
\end{verbatim}

is valid if either DoM or jArgSemSAT library is imported.

\subsection{Tweety compatible library}

Tweety libraries\cite{32} implement abstract argumentation reasoning procedures in the package net.sf.tweety.arg.dung. For instance, Figure 2 depicts a simple piece of code for creating a Dung’s argumentation framework with two

\textsuperscript{13}SATFormulae, Labelling, and DungAF belong to the package net.sf.jargsemsat.jargsemsat.datastructures.

\textsuperscript{14}http://tomcat.apache.org/

\textsuperscript{15}http://ws.arg.tech/
DungTheory at = new DungTheory();
at.add(new Argument("a"));
at.add(new Argument("b"));
at.add(new Attack(a,b));
PreferredReasoner r = new PreferredReasoner(at);
System.out.println(r.getExtensions());

Figure 2: Creating a simple Dung's Argumentation Framework using Tweety libraries and enumerating its preferred extensions.

arguments, a, and b, where a attacks b, and for enumerating its preferred extensions using the Tweety libraries.

In order to guarantee the full compatibility with the Tweety libraries[32], and to reduce the burden on programmers already using them, jArgSemSATTweety extends the net.sf.tweety.arg.dung.GroundReasoner, net.sf.tweety.arg.dung.PreferredReasoner, and net.sf.tweety.arg.dung.StableReasoner, overriding only the method computeExtensions in each of them. Therefore, by importing jArgSemSATTweety and using net.sf.jargsemsat.jArgSemSATTweety.PreferredReasoner instead of net.sf.tweety.arg.dung.PreferredReasoner, the software will automatically use Algorithm 1 for enumerating preferred extensions. As for the DoM compatible library, jArgSemSAT uses the default configurations only.

4.4 ArgTech compatible web-service

As presented in http://ws.arg.tech/, the ArgTech web-service solver[8] for abstract argumentation problems requires a POST message with the following fields:

- arguments, type String Array, e.g. ["A","B", "Arg_1"];
- attacks, type String Array, e.g. ["(A,B)", "(B,Arg_1)"];
- semantics, type String, one of grounded, preferred, stable, semistable\textsuperscript{16}.

For instance, the following JSON structure

\{
"arguments": ["a","b"],
"attacks": ["(a,b)"],
"semantics": "stable"
\}

is a valid POST request for jArgSemSATWeb. jArgSemSAT is then invoked with the default configurations only.

\textsuperscript{16}To ensure full compatibility, jArgSemSAT contains an experimental implementation of an algorithm for enumerating semi-stable extensions, see Section 6.
5 Evaluation

In this section, we present the result of a large experimental analysis comparing the performance of \textit{jArgSemSAT} with respect to \textit{ArgSemSAT}[14], Dung-o-Matic[30] and top ICCMA2015 solvers of tracks related to preferred and stable semantics: namely, ASPARTIX-D[22], Cegartix[20], CoQuiAAS[24], and LabSATSolver[7]. To complete the picture, we also include an analysis on complete extensions in A.

The aim of this section is to provide a good overview of the performance gap between the Java-based proposed system and the more efficient C++ implementations commonly exploited in competitions and academic studies. Moreover, the comparison with Dung-o-Matic (hereinafter DoM), helps to compare the performance of \textit{jArgSemSAT} with a Java-based tool that is currently exploited in research-grade prototypes such as those presented in [8, 35]. Among those, CISpaces[35] is now under analysis for transitioning into commercial products.

5.1 Experimental Setup

The experiments were performed on a cluster with computing nodes equipped with 2.4 Ghz Dual Core AMD Opteron\textsuperscript{TM} processors, 4 GB of RAM and Linux operating system. A cutoff of 600 seconds (10 minutes) —as in ICCMA2015— was imposed for solving each problem on a single \textit{AF}.

\textit{jArgSemSAT} can exploit any SAT solver that supports the DIMACS format. In the current version, it comes with the Java-based solver Sat4j\textsuperscript{25} integrated. This guarantees the maximum portability of the proposed system, and minimises the overhead due to external system calls. Therefore, Sat4j is the SAT solver used by \textit{jArgSemSAT} in this experimental analysis, unless differently specified. For the solvers selected according to their ICCMA2015 performance, the latest available version has been considered in this analysis\textsuperscript{17}. DoM has been provided with a Probo compatible command line interface by reusing part of the code wrote for \textit{jArgSemSAT}. For each solver we recorded the overall result: success, crashed, timed-out or ran out of memory.

Experiments have been conducted on the ICCMA2015 benchmark, which is a set of 192 randomly generated \textit{AF}s. They have been generated considering three different graph models, in order to provide different levels of complexity. More details can be found on the ICCMA website. Here we considered credulous acceptance DC-\(\sigma\), skeptical acceptance DS-\(\sigma\) and extensions enumeration EE-\(\sigma\) problems for \(\sigma \in \{\text{stable, preferred}\}\), as they are the most computationally difficult problems among those included in the competition.

Performance are measured in terms of IPC score and Penalised Average Runtime. The IPC score, borrowed from the planning community\textsuperscript{18}, is defined as follows. For each \textit{AF}, each system gets a score of \(1/(1+\log_{10}(T/T^*))\), where \(T\) is its execution time and \(T^*\) the best execution time among the compared

\textsuperscript{17}Solvers have been retrieved in September 2015 from the corresponding websites, provided in [33].

\textsuperscript{18}http://www.icaps-conference.org/index.php/Main/Competitions
systems, or a score of 0 if it fails in that case. Runtimes below 1.0 sec get by default the maximal score of 1.

The Penalised Average Runtime (PAR score) is a real number which counts (i) runs that fail to solve the considered problem as ten times the cutoff time (PAR10) and (ii) runs that succeed as the actual runtime. PAR scores are commonly used in automated algorithm configuration, algorithm selection, and portfolio construction[23] because using them allows runtime to be considered while still placing a strong emphasis on high instance set coverage.

### 5.2 Comparison with the State of the Art of Abstract Argumentation Solvers

Table 3: Performance achieved by jArgSemSAT, and the top three participants of the corresponding ICCMA2015 tracks on preferred semantics. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success, and ordered according to PAR10. ICCMA15 ranking is also reported.

<table>
<thead>
<tr>
<th>Solver</th>
<th>ICCMA15 Rank</th>
<th>IPC score</th>
<th>PAR10</th>
<th>% Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC-PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>1</td>
<td>164.7</td>
<td>3.2</td>
<td>100.0</td>
</tr>
<tr>
<td>LabSATSolver</td>
<td>2</td>
<td>167.9</td>
<td>3.4</td>
<td>100.0</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td>3</td>
<td>135.4</td>
<td>30.7</td>
<td>100.0</td>
</tr>
<tr>
<td>CoQuiAAS</td>
<td></td>
<td>186.4</td>
<td>63.6</td>
<td>98.9</td>
</tr>
<tr>
<td>DS-PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>1</td>
<td>171.2</td>
<td>5.9</td>
<td>100.0</td>
</tr>
<tr>
<td>Cegartix</td>
<td>2</td>
<td>161.2</td>
<td>7.9</td>
<td>100.0</td>
</tr>
<tr>
<td>LabSATSolver</td>
<td>3</td>
<td>171.0</td>
<td>12.6</td>
<td>100.0</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td></td>
<td>136.9</td>
<td>40.4</td>
<td>100.0</td>
</tr>
<tr>
<td>EE-PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cegartix</td>
<td>1</td>
<td>157.8</td>
<td>15.2</td>
<td>100.0</td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>2</td>
<td>147.7</td>
<td>66.1</td>
<td>99.5</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td></td>
<td>122.7</td>
<td>194.2</td>
<td>97.9</td>
</tr>
<tr>
<td>CoQuiAAS</td>
<td>3</td>
<td>172.9</td>
<td>218.2</td>
<td>96.9</td>
</tr>
</tbody>
</table>
This set of experiments focuses on assessing the performance gap between the proposed jArgSemSAT, ArgSemSAT and the top three solvers of the ICCMA2015 stable and preferred semantics tracks.

Table 3 shows the results, in terms of IPC score, PAR10 and percentage of successfully analysed frameworks, of the performed comparison on the preferred semantics tracks. Results of stable semantics tracks are reported in Table 4.

According to results shown in Table 3 and Table 4, we can safely state that jArgSemSAT is an off-the-shelf and ready-to-use efficient solver for computationally complex abstract argumentation problems. In terms of AFs successfully analysed, jArgSemSAT shows performance that are very similar to ArgSemSAT and to the winner of the considered tracks, though it is slower according to PAR10 and IPC score. Tables 3 and 4 also allow one to identify the performance gain given by the C++ implementation. Admittedly, ArgSemSAT is faster than jArgSemSAT; however, the performance gap is not critical.

According to the results shown in Tables 3 and 4, jArgSemSAT is comparable with the state of the art of solvers for abstract argumentation problems. Only in the DC-ST and DS-PR tracks jArgSemSAT is not among the best three considered solvers.

5.3 Comparison with the State of the Art of Off-the-Shelf Solvers

This analysis aims at comparing jArgSemSAT with the only available Java-based, off-the-shelf solver DoM.

Compared to the existing off-the-shelf implementation, DoM, jArgSemSAT statistically significantly outperforms DoM in both considered EE-σ problems in terms of runtime (WSRT, \( p < 0.05 \)), cf. Table 5. It is also noticeable the fact that DoM is able to successfully analyse a small number of benchmark AFs. Interestingly, we observed that DoM is the only system—among considered—that does not show a statistically significant difference (WSRT, \( p = 0.90 \)) in the CPU-time required for enumerating stable and preferred extensions of a given AF. Moreover, it is worth noting that DoM demonstrated to be very sensitive to the structure of the AFs to solve. Specifically, it did not solve—with respect to the considered enumeration problems—any of the graphs generated by using the “GroundedGenerator”. Such graphs are characterised by a very large grounded extension and a large number of nodes.

5.4 Importance of the SAT Solver

This analysis investigates the impact of different SAT solvers on the performance of jArgSemSAT. Specifically, we considered the Java-based SAT solver Sat4j—which guarantees high portability and easy usage—and glucose3.0[1]

\[19\] In the following we rely on the Wilcoxon Signed-Rank Test (WSRT) as a paired difference test to establish statistically significant difference[36].
Table 4: Performance achieved by jArgSemSAT, ArgSemSAT, and the top three participants of the corresponding ICCMA2015 tracks on stable semantics. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success, and ordered according to PAR10. ICCMA15 ranking is also reported. The provided ordering and the ICCMA15 ranking differ for DC-ST, due to close performance and a slightly different hardware configuration; and for DS-ST due to a bugfix.

<table>
<thead>
<tr>
<th>Solver</th>
<th>DC-ST</th>
<th>DS-ST</th>
<th>EE-ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ICCMA15 Rank</td>
<td>IPC score</td>
<td>PAR10</td>
</tr>
<tr>
<td>ASPARTIX-D</td>
<td>1</td>
<td>183.2</td>
<td>1.7</td>
</tr>
<tr>
<td>LabSATSolver</td>
<td>3</td>
<td>186.7</td>
<td>1.7</td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>2</td>
<td>172.5</td>
<td>2.7</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td>138.1</td>
<td>30.3</td>
<td>100.0</td>
</tr>
<tr>
<td>CoQuiAAS</td>
<td>3</td>
<td>180.2</td>
<td>65.7</td>
</tr>
</tbody>
</table>
Table 5: Performance achieved by jArgSemSAT and DoM on the corresponding ICCMA2015 tracks. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success. "–" indicates that the solver does not support the considered problem for the given semantic.

<table>
<thead>
<tr>
<th>Track</th>
<th>IPC score (192.0)</th>
<th>PAR10</th>
<th>% Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>jArg DoM</td>
<td>jArg DoM</td>
<td>jArg DoM</td>
</tr>
<tr>
<td>DC-PR</td>
<td>192.0 –</td>
<td>30.7 –</td>
<td>100.0 –</td>
</tr>
<tr>
<td>DS-PR</td>
<td>192.0 –</td>
<td>40.4 –</td>
<td>100.0 –</td>
</tr>
<tr>
<td>EE-PR</td>
<td>187.7 31.5</td>
<td>194.2 4332.1</td>
<td>97.9 28.1</td>
</tr>
</tbody>
</table>

Table 6 shows the results of the comparison between jArgSemSAT exploiting the Java-based Sat4j solver and jArgSemSAT using the C++ glucose3.0 SAT solver. For the sake of comparison, also the performance of the ArgSemSAT system are shown. Interestingly, results shown in Table 6 seem to indicate that

that is written in C++. It should be noted that Sat4j can keep the learned constraints between two satisfiability checks, in order to exploit the gained knowledge in subsequent calls on very similar CNFs. For the sake of modularity, and for providing a more objective comparison, this feature is not exploited in the jArgSemSAT framework.

We are aware that the exploitation of a C++ software can pose some strong portability issues, mainly due to compilers and libraries, but C++ solvers are generally believed to be faster than corresponding Java-based systems. Therefore, here we are interested in measuring such performance gap, in order to make jArgSemSAT users aware of the importance of the solver. However, it should be noted that Sat4j has been included in the overall jArgSemSAT framework, while glucose3.0 has to be executed through PIPE communication system among processes.

For stressing the importance of SAT solvers, thus obtaining a better understanding of their impact on jArgSemSAT performance, we considered the empirically most computationally expensive tasks. According to the results shown in Table 3 and Table 4, the problem of enumerating the preferred extensions (EE-PR) of a given AF requires the largest amount of CPU-time. This can be easily derived by the PAR10 scores and the percentage of successfully analysed AFs by the considered solvers.

Table 6 shows the results of the comparison between jArgSemSAT exploiting the Java-based Sat4j solver and jArgSemSAT using the C++ glucose3.0 SAT solver. For the sake of comparison, also the performance of the ArgSemSAT system are shown. Interestingly, results shown in Table 6 seem to indicate that
Table 6: Performance achieved by jArgSemSAT exploiting either Sat4j or glucose3.0 on the ICCMA2015 benchmark. For the sake of comparison, also the performance of ArgSemSAT are provided. Results are shown in terms of IPC score (maximum achievable is 192.0), percentages of success, percentages of AFs in which the system has been the fastest and PAR10. Values in bold indicate the best results.

<table>
<thead>
<tr>
<th></th>
<th>jArgSemSAT</th>
<th></th>
<th>ArgSemSAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sat4j</td>
<td>glucose3.0</td>
<td></td>
</tr>
<tr>
<td>IPC score</td>
<td>151.2</td>
<td>146.0</td>
<td>184.4</td>
</tr>
<tr>
<td>% success</td>
<td>97.9</td>
<td>99.5</td>
<td>99.5</td>
</tr>
<tr>
<td>% best</td>
<td>13.5</td>
<td>9.9</td>
<td>65.6</td>
</tr>
<tr>
<td>PAR10</td>
<td>194.2</td>
<td>81.7</td>
<td>66.1</td>
</tr>
</tbody>
</table>

the use of glucose3.0 does not provide a remarkable performance improvement. In particular, the exploitation of the external C++ solver has a detrimental effect on the performance of jArgSemSAT in terms of IPC score and number of times the approach has been the fastest. However, when a closer look to the observed performance is taken, an interesting pattern emerges. Surprisingly, the performance of considered SAT solvers are not directly related to the number of preferred extensions, i.e. there is no direct relation between the number of times the solver is called by jArgSemSAT and the runtime. Furthermore, Sat4j improves the performance of jArgSemSAT on AFs that can be solved in less than—approximately—50 CPU-time seconds; on more complex AFs, the use of glucose3.0 is usually beneficial. Mainly because of that, the Wilcoxon test indicates that the performance of the compared systems are significantly different (WSRT, $p = 0.01$). The ability of glucose3.0 to handle empirically complex AFs, is confirmed by the fact that the use of glucose3.0 allows jArgSemSAT to solve, within the given time, a few more AFs from the considered ICCMA2015 benchmark.

6 Conclusion

In this paper we present jArgSemSAT, an efficient off-the-shelf solver for abstract argumentation problems. In the previous sections we give evidence of how jArgSemSAT not only is compatible with the current off-the-shelf solver, namely Dung-O-Matic[30], and with the Tweety libraries[32]; not only exists in a web-service version compatible with ArgTech technologies[8]—and we made it freely available at http://cicero.cs.cf.ac.uk/jArgSemSATWeb/restapi/argtech/--; but it is among the best solvers in particular for most of the tracks of the ICCMA2015 competition associated to the two semantics with highest computational complexity, namely stable and preferred semantics. As discussed
in Section 5.4, the choice of the oracle can significantly impact the performance of a solver. This is true not only for the proposed $j\text{ArgSemSAT}$ solver, but also for competitors. Indeed, as noticed by one of the reviewers, it would be interesting how other systems such as CoQuiAAS would perform with a more powerful MSS enumerator, such as the one proposed in [29].

Currently, $j\text{ArgSemSAT}$ is used within CISpaces[35] that has been our main use-case. CISpaces (Collaborative Intelligence Spaces) is a tool mostly written in Java—only the GUI is written in Python—to help analysts in acquiring, evaluating and interpreting information. Indeed, the aim of intelligence analysis is to make sense of information that is often conflicting or incomplete, and to weigh competing hypotheses that may explain a situation. This imposes a high cognitive load on analysts, and there are few automated tools to aid them in their task. CISpaces assists analysts in reasoning with different types of evidence: analysts are supported in structuring evidence using argumentation schemes, and in identifying plausible hypotheses via the computation of preferred extensions.

By adopting $j\text{ArgSemSAT}$, CISpaces now computes the preferred extensions of average analysis almost instantaneously: before, using Dung-O-Matic, it required 60 seconds or more. This was becoming a serious impediment to the adoption of CISpaces for training new analysts—its main goal—and it was listed as one of the improvements needed to be addressed before moving the project towards a commercial transition. After the excellent performance of $\text{ArgSemSAT}$ at ICCMA2015, we decided to re-code it in Java to ease the integration. This also satisfied the other requirement to make it available as a replacement for Dung-O-Matic and integrate it within the Tweety libraries—both written in Java. Finally, it also greatly simplified the task of producing a web-service interface.

Moreover, $j\text{ArgSemSAT}$ allows CISpaces to use its probabilistic argumentation engine in real analysis. Indeed, CISpaces includes a probabilistic argumentation engine[27, 26] that heavily resides on preferred extensions computed on probabilistic manipulation of AFs. Therefore, the preferred extension enumeration solver needs to be invoked an exponential number of times.

As discussed in Section 3.3 of Li’s work[26], when an argumentation framework is analysed from a probabilistic standpoint, it is necessary to know the semantics extensions of all the possible combinations of sub-graphs. Therefore, for a simple argumentation framework with two arguments $a$, and $b$, this requires to enumerate the semantics extensions of the frameworks: $\emptyset$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\emptyset$, $\{a, b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{b\}$, $\{a\}$, $\{b\}$, $\{a, b\}$.

In order to compute the results for those probabilistic approaches, semantics extensions must be computed exhaustively for all the possible sub-graphs. While there is some work in the dynamics in abstract argumentation (e.g. [12]) to produce efficient algorithms for reusing partially computed results, possibly exploiting the concept of Input/Output multipoles[3], efficient algorithms for
computing such results are surely needed.

In the case of CISpaces, while Dung-O-Matic limited the use of the proba-
bilistic argumentation engine to toy examples of less than ten arguments, and
still requiring between 30-90 seconds, jArgSemSAT makes it available for real
analysis involving up to 50/60 arguments with solutions within 10 seconds.

Therefore, jArgSemSAT has positively contributed to push the research grade
prototype CISpaces towards a plan for transitioning into a commercial product.
Indeed, jArgSemSAT helped CISpaces to receive positive qualitative feedback
from trained analysts chosen to evaluate it.

The future of jArgSemSAT, in our view, lays in supporting all the research
community to build and exploit argumentation-based tools. That is the reason
that motivated us in providing a free-of-charge, but clearly with best-effort only
SLA, web-service interface to jArgSemSAT. From a technical perspective, we
need to ultimate the technical documentation and we plan to include support
for the remaining semantics, notably semi-stable[10]—that is already supported
in an experimental, non-optimised version.

We will also create a web-based interface for goal-driven manipulation and
evaluation of AFs. Currently, most of the web-based interfaces to argumentation
tools, e.g. OVA\textsuperscript{20}, TOAST\textsuperscript{21}, Aspartix\textsuperscript{22}, Conarg\textsuperscript{23}, allow a user first to prepare
an argumentation knowledge base, and then to run a solver on it. However, they
do not provide “versioning” support: a user needs to manually keep track of the
correspondence between its action on the knowledge base and the effects on the
computed extensions. Moreover, they do not provide “strategical” support: a
user aiming at having a specific argument accepted has no guidance on how to
achieve such a goal. The tool we plan to build will prove itself very useful for
researchers on dynamics and argumentation\textsuperscript{[28, 6, 3]} and more broadly for the
entire argumentation community.

Acknowledgments

The authors thank (in alphabetical order) Prof. Chris Reed and Dr. Mark
Snaith—University of Dundee—for their support in ensuring compatibility be-
tween jArgSemSAT and existing technologies at ArgTech. Moreover, we thank
Dr. Matthias Thimm—Universität Koblenz-Landau—for his support in inte-
grating jArgSemSAT with Tweety, and Dr. Alice Toniolo—University of Aberdeen—
for integrating jArgSemSAT within CISpaces.

The authors would like to acknowledge the use of the University of Hudder-
sfield Queensgate Grid in carrying out this work.

The authors also thank the anonymous reviewers for their helpful comments.

\textsuperscript{20}http://ova.arg-tech.org/
\textsuperscript{21}http://toast.arg-tech.org/
\textsuperscript{22}http://rull.dbai.tuwien.ac.at:8080/ASPARTIX/index.faces
\textsuperscript{23}http://www.dmi.unipg.it/conarg/
References


A Experimental Evaluation for Complete Extensions
Table 7: Performance achieved by $j$ArgSemSAT, ArgSemSAT, and the top three participants of the corresponding ICCMA2015 tracks on complete extensions. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success, and ordered according to PAR10. ICCMA15 ranking is also reported.

<table>
<thead>
<tr>
<th>Solver</th>
<th>ICCMA15 Rank</th>
<th>IPC score</th>
<th>PAR10</th>
<th>% Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC-CO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>1</td>
<td>179.0</td>
<td>2.5</td>
<td>100.0</td>
</tr>
<tr>
<td>LabSATSolver</td>
<td>3</td>
<td>182.0</td>
<td>2.8</td>
<td>100.0</td>
</tr>
<tr>
<td>ASPARTIX-D</td>
<td>2</td>
<td>171.2</td>
<td>3.1</td>
<td>100.0</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td></td>
<td>145.6</td>
<td>15.5</td>
<td>100.0</td>
</tr>
<tr>
<td>DS-CO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LabSATSolver</td>
<td>2</td>
<td>179.8</td>
<td>1.1</td>
<td>100.0</td>
</tr>
<tr>
<td>ASGL</td>
<td>1</td>
<td>191.5</td>
<td>2.3</td>
<td>100.0</td>
</tr>
<tr>
<td>ConArg</td>
<td>3</td>
<td>169.1</td>
<td>3.9</td>
<td>100.0</td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>4</td>
<td>161.0</td>
<td>4.5</td>
<td>100.0</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td></td>
<td>148.6</td>
<td>14.5</td>
<td>100.0</td>
</tr>
<tr>
<td>EE-CO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASPARTIX-D</td>
<td>1</td>
<td>175.8</td>
<td>8.0</td>
<td>100.0</td>
</tr>
<tr>
<td>CoQuiAAS</td>
<td>3</td>
<td>181.3</td>
<td>46.1</td>
<td>99.4</td>
</tr>
<tr>
<td>jArgSemSAT</td>
<td></td>
<td>114.6</td>
<td>215.7</td>
<td>97.4</td>
</tr>
<tr>
<td>ArgSemSAT</td>
<td>2</td>
<td>131.2</td>
<td>254.0</td>
<td>96.4</td>
</tr>
</tbody>
</table>
Table 8: Performance achieved by jArgSemSat and DoM on the corresponding ICCMA2015 tracks. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success. “–” indicates that the solver does not support the considered problem for the given semantic.

<table>
<thead>
<tr>
<th>Track</th>
<th>IPC score (192.0)</th>
<th>PAR10</th>
<th>% Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>jArg</td>
<td>DoM</td>
<td>jArg</td>
</tr>
<tr>
<td>DC-CO</td>
<td>192.0</td>
<td>–</td>
<td>15.5</td>
</tr>
<tr>
<td>DS-CO</td>
<td>192.0</td>
<td>–</td>
<td>14.5</td>
</tr>
<tr>
<td>EE-CO</td>
<td>180.1</td>
<td>39.3</td>
<td>215.7</td>
</tr>
</tbody>
</table>