Argumentation Extensions Enumeration as a Constraint Satisfaction Problem: a Performance Overview

Mauro Vallati[1], Federico Cerutti[2], and Massimiliano Giacomin[3]

[1] School of Computing and Engineering, University of Huddersfield, UK
m.vallati@hud.ac.uk
[2] Department of Computing Science, University of Aberdeen, UK
f.cerutti@abdn.ac.uk
[3] Department of Information Engineering, University of Brescia, Italy
massimiliano.giacomin@unibs.it

Abstract. Enumerating semantics extensions in abstract argumentation is generally an intractable problem. For preferred semantics four implementations have been recently proposed, **CONArg2**, **AspartixM**, **PrefSAT**and **NAD-Alg**, with significant runtime variations. This work is a first empirical evaluation of the performance of these implementations with the hypothesis that **NAD-Alg**, as representative of a family of ad-hoc approaches, will overcome in sequence **PrefSAT** — a SAT-based approach —, **CONArg2**— a CSP-based approach —, and the ASP-based approach **AspartixM**. The results shows that this is not always the case, as **PrefSAT** has been often the best approach both in terms of numbers of enumeration problems solved, and CPU-time. Moreover, we identify situations where **AspartixM** has been proved to be significantly faster than **CONArg2**.

Keywords: argumentation, preferred semantics, empirical evaluation

1 Introduction

Dung's theory of abstract argumentation frameworks [11] provides a general model, which is widely recognized as a fundamental reference in computational argumentation in virtue of its simplicity, generality, and ability to capture a variety of more specific approaches as special cases [11]. An abstract argumentation framework (AF) consists of a set of arguments and of an *attack* relation between them. The concept of *extension* plays a key role in this simple setting, where an *extension* is intuitively a set of arguments which can "survive the conflict together". Different notions of extensions and of the requirements they should satisfy correspond to alternative *argumentation semantics*. In [11] four "traditional" semantics were introduced, namely *complete*, *grounded*, *stable*, and *preferred* semantics (see [2, 1] for an introduction).

The introduction of preferred semantics is one of the main contribution of Dung's paper. The semantics' name, in fact, reflects a sort of preference w.r.t. other traditional semantics, 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 (differently from complete semantics).

The main computational problems in abstract argumentation are naturally related to extensions and can be partitioned into two classes: *decision* problems and *construction* problems. Decision problems pose yes/no questions like "Does this argument belong to one (all) extensions?" or "Is this set an extension?", while construction problems require to explicitly produce some of the extensions prescribed by a semantics. In particular, *extension enumeration* is the problem of constructing all the extensions prescribed by a given semantics for a given AF.

Theoretical analysis of worst-case computational issues in abstract argumentation is in a state of maturity with the available complexity results covering all Dung's traditional semantics and several subsequent prominent approaches in the literature (for a summary see [12]). In the case of preferred semantics, standard decision problems result to be *in-between* tractability and full second-level complexity [9, 8] and this extends directly to construction/enumeration problems. On the practical side, however, the investigation on efficient algorithms for abstract argumentation and on their empirical assessment is less developed, with few results available in the literature.

This paper contributes to fill this gap by comparing the two families of approaches considered in the literature. On one hand, one may develop a dedicated algorithm to obtain the problem solution, on the other hand, one may reduce the problem instance into an equivalent instance of a different class of problems for which solvers are already available. The results produced by the solver have then to be translated back to the original problem.

In our experiment, we compare the state-of-the-art approaches which reduce the preferred extension enumeration problem into a CSP (**CONArg2**), ASP (**AspartixM**) and SAT (**PrefSAT**) with the best ad-hoc approach [18] in its latest version — **NAD-Alg** [19]. Our experimental hypothesis is that there will be a strict ordering — under any configuration — regarding the performance of the software measured in (1) CPU-time needed to enumerate all the preferred extensions given an AF and in (2) percentage of successful enumeration. Such an ordering should see the ad-hoc approach **NAD-Alg** as the best one, followed by **PrefSAT**, **CONArg2**, and finally **AspartixM**. As discussed later on the paper, our hypothesis has been proved true only partially: we identified several cases where (1) **PrefSAT** has been the best implementation — and it is also the only one implementation that solved all the AFs considered in the experiment — and (2) **AspartixM** performed significantly — according to the Friedman statistic test confirmed by a post-hoc analysis with the Wilcoxon signed rank with a Bonferroni correction applied [22] — better than **CONArg2**.

The paper is organized as follows. Section 2 recalls the necessary basic theoretical concepts of Dung's argumentation frameworks, Constraint Satisfaction Programming, Answer Set Programming, and Propositional Satisfiability Problems. Section 3 summarises the current state-of-the-art of the approaches for enumerating preferred extensions, namely **NAD-Alg**, **CONArg2**, **AspartixM**, and **PrefSAT**. Section 4 describes the test setting and comments the experimental results and then Section 5 concludes the paper.

2 Background

2.1 Dung's Argumentation Framework

An argumentation framework [11] consists of a set of arguments and a binary attack relation between them.[4]

Definition 1. *An* argumentation framework *(AF) is a pair $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of arguments and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$. We say that $\boldsymbol{a}_2$* attacks *$\boldsymbol{a}_1$ iff $\langle \boldsymbol{a}_2, \boldsymbol{a}_1 \rangle \in \mathcal{R}$, also denoted as $\boldsymbol{a}_2 \rightarrow \boldsymbol{a}_1$. The set of attackers of an argument $\boldsymbol{a}_1$ will be denoted as $\boldsymbol{a}_1^- \triangleq \{\boldsymbol{a}_2 : \boldsymbol{a}_2 \rightarrow \boldsymbol{a}_1\}$, the set of arguments attacked by $\boldsymbol{a}_1$ will be denoted as $\boldsymbol{a}_1^+ \triangleq \{\boldsymbol{a}_2 : \boldsymbol{a}_1 \rightarrow \boldsymbol{a}_2\}$.*

An argument $\mathbf{a}_1$ without attackers, i.e. such that $\mathbf{a}_1^- = \emptyset$, is said *initial*. The *neighbour* of an argument $\mathbf{a}_1$ is $\mathbf{a}_1^- \cup \mathbf{a}_1^+$. Moreover, each argumentation framework can be represented as a directed graph where the vertices are the arguments, and the edges are 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 $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:*

- *a set $S \subseteq \mathcal{A}$ is a* conflict–free *set of Γ if $\nexists\, \boldsymbol{a}_1, \boldsymbol{a}_2 \in S$ s.t. $\boldsymbol{a}_1 \rightarrow \boldsymbol{a}_2$;*
- *an argument $\boldsymbol{a}_1 \in \mathcal{A}$ is* acceptable *with respect to a set $S \subseteq \mathcal{A}$ of Γ if $\forall \boldsymbol{a}_2 \in \mathcal{A}$ s.t. $\boldsymbol{a}_2 \rightarrow \boldsymbol{a}_1$, $\exists\, \boldsymbol{a}_3 \in S$ s.t. $\boldsymbol{a}_3 \rightarrow \boldsymbol{a}_2$;*
- *a set $S \subseteq \mathcal{A}$ is an* admissible set *of Γ if S is a conflict–free set of Γ and every element of S is acceptable with respect to S of Γ.*

An argumentation semantics σ prescribes for any AF Γ a set of *extensions*, denoted as $\mathcal{E}_\sigma(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by σ. Here we need to recall the definitions of complete (denoted as $\mathcal{CO}$), and preferred (denoted as $\mathcal{PR}$) semantics only.

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

- *a set $S \subseteq \mathcal{A}$ is a* complete extension *of Γ, i.e. $S \in \mathcal{E}_{\mathcal{CO}}(\Gamma)$, iff S is admissible and $\forall \boldsymbol{a}_1 \in \mathcal{A}$ s.t. $\boldsymbol{a}_1$ is acceptable w.r.t. S, $\boldsymbol{a}_1 \in S$;*
- *a set $S \subseteq \mathcal{A}$ is a* preferred extension *of Γ, i.e. $S \in \mathcal{E}_{\mathcal{PR}}(\Gamma)$, iff S is a maximal (w.r.t. set inclusion) complete extension of Γ.*

It can be noted that each extension S implicitly defines a three-valued *labelling* of arguments, as follows: an argument $\mathbf{a}_1$ is labelled `in` iff $\mathbf{a}_1 \in S$, is labelled `out` iff $\exists\, \mathbf{a}_2 \in S$ s.t. $\mathbf{a}_2 \rightarrow \mathbf{a}_1$, is labelled `undec` if neither of the above conditions holds. In the light of this correspondence, argumentation semantics can equivalently be defined in terms of labellings rather than of extensions (see [5, 1]). In particular, the notion of *complete labelling* [6, 1] provides an equivalent characterization of complete semantics, in the sense that each complete labelling corresponds to a complete extension and vice versa. Complete labellings can be (redundantly) defined as follows.

[4] In this paper we consider only *finite* sets of arguments: see [3] for a discussion on infinite sets of arguments.

Definition 4. *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A total function $\mathcal{L}ab : \mathcal{A} \mapsto \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ is a* complete labelling *iff it satisfies the following conditions for any $\boldsymbol{a}_1 \in \mathcal{A}$:*

- $\mathcal{L}ab(\boldsymbol{a}_1) = \texttt{in} \Leftrightarrow \forall \boldsymbol{a}_2 \in \boldsymbol{a}_1^- \mathcal{L}ab(\boldsymbol{a}_2) = \texttt{out}$;
- $\mathcal{L}ab(\boldsymbol{a}_1) = \texttt{out} \Leftrightarrow \exists \boldsymbol{a}_2 \in \boldsymbol{a}_1^- : \mathcal{L}ab(\boldsymbol{a}_2) = \texttt{in}$;
- $\mathcal{L}ab(\boldsymbol{a}_1) = \texttt{undec} \Leftrightarrow \forall \boldsymbol{a}_2 \in \boldsymbol{a}_1^- \mathcal{L}ab(\boldsymbol{a}_2) \neq \texttt{in} \wedge \exists \boldsymbol{a}_3 \in \boldsymbol{a}_1^- : \mathcal{L}ab(\boldsymbol{a}_3) = \texttt{undec}$;

It is proved in [5] that preferred extensions are in one-to-one correspondence with those complete labellings maximizing the set of arguments labelled `in`.

2.2 Constraint Satisfaction Programming

Constraint programming is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence (AI), operations research, algorithms, and graph theory [21].

Following [4], a *Constraint Satisfaction Problem* (CSP) P [21] is a triple $P = \langle X, D, C \rangle$ such that:

- $X = \langle x_1, \ldots, x_n \rangle$ is a tuple of variables;
- $D = \langle D_1, \ldots, D_n \rangle$ a tuple of domains such that $\forall i, x_i \in D_i$;
- $C = \langle C_1, \ldots, C_t \rangle$ is a tuple of constraints, where $\forall j, C_j = \langle R_{S_j}, S_j \rangle$, $S_j \subseteq \{x_i | x_i \text{ is a variable}\}$, $R_{S_j} \subseteq S_j^D \times S_j^D$ where $S_j^D = \{D_i | D_i \text{ is a domain, and } x_i \in S_j\}$.

A solution to the CSP P is $A = \langle a_1, \ldots, a_n \rangle$ where $\forall i, a_i \in D_i$ and $\forall j, R_{S_j}$ holds on the projection of A onto the scope S_j. If the set of solutions is empty, the CSP is unsatisfiable.

2.3 Answer Set Programming

Over the the last years, Answer Set Programming (ASP) [15] has emerged as a declarative problem solving paradigm. It evolved from various fields such as Logic Programming, Deductive Databases, Knowledge Representation, and Nonmonotonic Reasoning, and serves as a flexible language for declarative problem solving. There are two main tasks in problem solving, representation and reasoning, which are clearly separated in the declarative paradigm. In ASP, representation is done using a rule-based language, while reasoning is performed using implementations of general-purpose algorithms, referred to as ASP solvers.

Rules in ASP are interpreted according to common sense principles, including a variant of the closed-world-assumption (CWA) and the unique-name-assumption (UNA). Collections of ASP rules are referred to as ASP programs, which represent the modelled knowledge. To each ASP program a collection of answer sets, or intended models, is associated, which stand for the solutions to the modelled problem; this collection can also be empty, meaning that the modelled problem does not admit a solution. Several reasoning tasks exist: the classical ASP task is enumerating all answer sets or determining whether an answer set exists, but ASP also allows for query answering in brave or cautious modes [16].

2.4 Propositional Satisfiability Problems

In the propositional satisfiability problem (SAT) the goal is to determine whether a given Boolean formula is satisfiable. A variable assignment that satisfies a formula is a solution. In SAT, formulae are commonly expressed in Conjunctive Normal Form (CNF). A formula in CNF is a conjunction of clauses, where clauses are disjunctions of literals, and a literal is either positive (a variable) or negative (the negation of a variable). If at least one of the literals in a clause is true, then the clause is satisfied, and if all clauses in the formula are satisfied then the formula is satisfied and a solution has been found.

A wide range of decision and optimisation problems can be reduced as SAT instances. A few examples are logical circuit design [14], AI Planning [20], and Sudoku [23]. Finally, a growing number of high performance SAT solver is available, mainly fostered by the annual SAT competition.[5] The competition allows to have a good overview of the performance of the current state-of-the-art on structurally different SAT instances.

3 The State of the Art of Enumerating Preferred Extensions

In this section we summarise the main idea behind the most competitive ad-hoc approach for solving the enumeration problem for preferred semantics, viz. **NAD-Alg**, and the three champions for transforming such a problem into a (resp.) CSP (**CONArg2**), ASP (**AspartixM**), and SAT (**PrefSAT**) problem.

3.1 NAD-Alg

The algorithm proposed in [18, 19] has been shown to outperform the other ad-hoc approaches [10, 17] and will be therefore taken as the only term of comparison for this family of approaches.

NAD-Alg [19] is a depth-first backtracking procedure that traverses a binary search tree. The root considers the case where all the arguments in the AF are *blank*, i.e. not yet visited. At each step of the search process, a blank node is selected and assumed to belong to the preferred extension, and a local evaluation on its neighbour is performed. Then the procedure recursively call itself on the assumption that the above arguments are respectively *in* or *out* the extension. Any inconsistency leads to a backtrack.[6]

3.2 CONArg2

In [4], the authors propose a mapping from AFs to CSPs. Given an AF $\langle \mathcal{A}, \mathcal{R} \rangle$, they first create a variable for each argument whose domain is always $\{0, 1\}$ — $\forall \mathbf{a}_i \in \mathcal{A}, \exists x_i \in X$ such that $D_i = \{0, 1\}$.

[5] `http://satcompetition.org/`

[6] **NAD-Alg**'s implementation is available at `https://sourceforge.net/projects/argtools/files/?source=navbar` (retrieved on 11th March 2014).

Subsequently, they describe constraints associated to different definitions of Dung's argumentation framework: for instance $\{\mathbf{a}_1, \mathbf{a}_2\} \subseteq \mathcal{A}$ is conflict–free iff $\neg(x_1 = 1 \wedge x_2 = 1)$. The other constraints are imposed following the same reasoning line.

In this paper, we consider the most recent advancement of the Conarg project, viz. **CONArg2**, which is the command line version that is implemented by Gecode 4.0 (C++). This version is faster and it is able to compute credulous/skeptical acceptance of an argument for stable, complete and admissible AF semantics too.

3.3 AspartixM

AspartixM [13] expresses argumentation semantics in Answer Set Programming (ASP): a single program is used to encode a particular argumentation semantics, and the instance of an argumentation framework is given as an input database. Tests for subset-maximality exploit the `metasp` optimisation frontend for the ASP-package `gringo/claspD`.[7]

Given an $AF\ \langle \mathcal{A}, \mathcal{R} \rangle$, Aspartix encodes the requirements for a "semantics" (e.g. the conflict–free requirements) in an ASP program whose database considers:

$$\{\mathrm{arg}(\mathbf{a}) \mid \mathbf{a} \in \mathcal{A}\} \cup \{\mathrm{defeat}(\mathbf{a}_1, \mathbf{a}_2) \mid \langle \mathbf{a}_1, \mathbf{a}_2 \rangle \in \mathcal{R}\}$$

The following program fragment is thus used to check the conflict–freeness [13]:

$$\begin{aligned} \pi_{cf} = \{\ & \mathrm{in}(X) \leftarrow not\mathrm{out}(X), \mathrm{arg}(X); \\ & \mathrm{out}(X) \leftarrow not\mathrm{in}(X), \mathrm{arg}(X); \\ & \leftarrow \mathrm{in}(X), \mathrm{in}(Y), \mathrm{defeat}(X, Y)\}. \end{aligned}$$

3.4 PrefSAT

PrefSAT [7] performs a search in the space of complete extensions to enumerate the maximal ones. In particular, **PrefSAT** encodes the constraints corresponding to complete extensions into a SAT-problem. A SAT-solver is then called to solve it, thus returning a complete extension. A depth-first technique is then applied in order to determine the maximal complete extension containing the one already found — i.e. a preferred extension. Previously explored search's states are excluded from further exploration by adding specific constraints to the encoding of complete extensions.

Differently from Aspartix and Conarg, **PrefSAT** encodes the Caminada labelling requirements in a CNF. To do so, for each argument $\mathbf{a}_i \in \mathcal{A}$, three propositional variables are considered: I_i (which is true iff $\mathcal{L}ab(\mathbf{a}_i) = \texttt{in}$), O_i (which is true iff $\mathcal{L}ab(\mathbf{a}_i) = \texttt{out}$), U_i (which is true iff $\mathcal{L}ab(\mathbf{a}_i) = \texttt{undec}$).

Given $|\mathcal{A}| = k$ and $\phi : \{1, \ldots, k\} \mapsto \mathcal{A}$, the conjunction of the formulae listed below is a CNF representing the requirements for a complete labelling (the last formula filters out the trivial case of the empty set):

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

[7] **AspartixM** has been executed with `gringo` version 3.0.3 and `claspD` version 1.1.4.

$$\bigwedge_{\{i|\phi(i)^-=\emptyset\}} I_i \tag{2}$$

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

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

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

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

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \left(\bigwedge_{\{k|\phi(k)\rightarrow\phi(i)\}} \left(U_i \vee \neg U_k \vee \left(\bigvee_{\{j|\phi(j)\rightarrow\phi(i)\}} I_j \right) \right) \right) \tag{7}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \left(\left(\bigwedge_{\{j|\phi(j)\rightarrow\phi(i)\}} (\neg U_i \vee \neg I_j) \right) \wedge \left(\neg U_i \vee \left(\bigvee_{\{j|\phi(j)\rightarrow\phi(i)\}} U_j \right) \right) \right) \tag{8}$$

$$\bigvee_{i\in\{1,\ldots k\}} I_i \tag{9}$$

As noticed in [7], the conjunction of the above formulae is redundant. However, the non-redundant CNFs are not equivalent from an empirical evaluation [7]: the overall performance is significantly affected by the chosen configuration pair CNF encoding–SAT solver. In the following, we considered the most efficient configuration of **PrefSAT** as described in [7].

4 Empirical Evaluation

In this section we present the results of an experimental study examining **CONArg2**, **AspartixM**, **PrefSAT** and **NAD-Alg**.

The experimental analysis has been conducted on 720 AFs that were divided in different classes, according to two dimensions: the number of arguments, $|\mathcal{A}|$ and the criterion of random generation of the attack relation, $|\mathcal{R}|$. As to $|\mathcal{A}|$ we considered 8 different values, ranging from 25 to 225 with a step of 25. As to the generation of the attack relation we used two alternative methods. The first method consists in fixing the probability p_{att} that there is an attack for each ordered pair of arguments (self-attacks are included): for each pair a pseudo-random number uniformly distributed between 0 and 1 is generated and if it is lesser or equal to p_{att} the pair is added to the attack relation. We considered three values for p_{att}, namely 0.25, 0.5 and 0.75. Combining the 9 values of $|\mathcal{A}|$ with the 3 values of p_{att} gives rise to 27 test classes, each of which has been populated with 20 AFs. 10 of them included auto-attacks, the other do not.

The second method consists in generating randomly, for each AF, the number n_{att} of attacks it contains (extracted with uniform probability between 0 and $|\mathcal{A}|^2$). Then the n_{att} distinct pairs of arguments constituting the attack relation are selected randomly. Applying the second method with the 9 values of $|\mathcal{A}|$ gives rise to 9 further test classes, each of which has been populated with 20 AFs.

The solvers and the feature extraction algorithms have been run on a cluster with computing nodes equipped with 2.5 Ghz Intel Core 2 Quad Processors™, 8 GB of RAM and Linux operating system. As in the International Planning Competition (IPC), a cutoff of 900 seconds was imposed to compute the preferred extensions for each AF.[8] The systems under evaluation have been compared with respect to the ability to produce solutions within the time limit and to the execution time. As to the latter comparison, we adopted the IPC speed score, also borrowed from the planning community, which is defined as follows:

- For each test case (in our case, each test AF) let T^* be the best execution time among the compared systems (if no system produces the solution within the time limit, the test case is not considered valid and ignored).
- For each valid case, each system gets a score of $1/(1 + \log_{10}(T/T^*))$, where T is its execution time, or a score of 0 if it fails in that case. Runtimes below 1 sec get by default the maximal score of 1.
- The (non normalised) IPC score for a system is the sum of its scores over all the valid test cases. The normalised IPC score ranges from 0 to 100 and is defined as $(IPC/\#\text{ of valid cases}) * 100$.

Therefore, the higher the IPC score, the better the performance.

Figure 1 presents the values of normalised IPC score considering all test cases grouped w.r.t. $|\mathcal{A}|$, while Table 1 shows the average time needed by the four systems for computing the preferred extensions. Cutoff runtime is considered for unsolved AFs. Both **PrefSAT** and **NAD-Alg** performed significantly better (note that the IPC score is logarithmic) than **AspartixM** and **CONArg2** for all values of $|\mathcal{A}| > 75$, and the performance gap increases with increasing $|\mathcal{A}|$. Interestingly, it is possible to clusterise the approaches according to their IPC performance. In one cluster there are **AspartixM** and **CONArg2**, and the other includes **PrefSAT** and **NAD-Alg**. It is a surprising result, in the light of the very different approaches they exploit in order to enumerate preferred

[8] `http://helios.hud.ac.uk/scommv/IPC-14/`

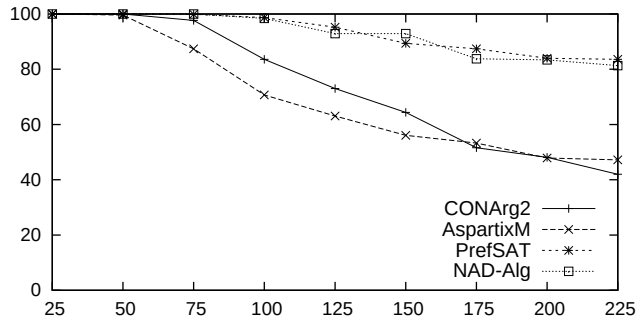

Fig. 1. Normalised IPC score (y axis) w.r.t. the number of arguments (x axis) of each considered system.

extensions. While the good performance of **NAD-Alg** are expected, given the fact that this system does not encode the AF into a CSP problem, we would have expected all the others to perform somehow similarly. The behaviour observed in Figure 1 is confirmed by the results shown in Table 1. While the average runtime of **AspartixM** and **CONArg2** rapidly increases, **NAD-Alg** and **PrefSAT** have similar performance, even though **NAD-Alg** run out of time in some of the largest considered AFs.

	Average CPU-Time								
	25	50	75	100	125	150	175	200	225
CONArg2	0.25	0.27	0.65	2.15	5.48	14.98	73.78	86.62	187.11
AspartixM	0.18	0.67	1.44	3.26	6.02	15.70	27.99	87.46	117.18
PrefSAT	0.04	0.11	0.23	**0.44**	**0.81**	**1.67**	**3.76**	**6.41**	**16.21**
NAD-Alg	**0.01**	**0.02**	**0.06**	0.99	10.23	12.74	60.35	42.78	75.07

Table 1. Average runtime for each of the considered solvers, according to the number of arguments of the AFs.

Figure 2 presents the values of normalised IPC considering all test cases grouped w.r.t. $|\mathcal{R}|$. Again, as in the previous analysis, it is possible to clearly distinguish between two clusters of solvers, and the composition of clusters is the same as before. On the other hand, it is worth noticing that relatives performance change significantly with the probability of attacks between arguments. Systems that perform well with a high probability (50 to 75 %), are slow with small (25%) probabilities. Performance of solvers within the same cluster tend to be the same on AFs with a random probability of attacks. Intuitively, in highly constrained AFs, i.e. with a large number of attacks, it is easier to enumerate preferred extensions; their number is low. If the number of attacks

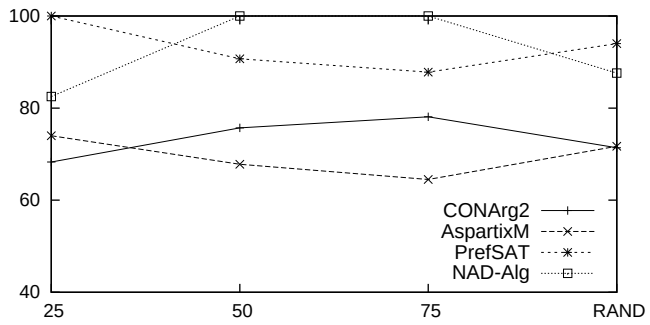

Fig. 2. Normalised IPC score (y axis) w.r.t. the probability of attacks (x axis) of each considered system.

decrease, the number of extensions increases, thus a bigger number of search steps is required to enumerate all the possible extensions.

Table 2 shows the percentages of AFs in which each solver was able to provide a solution, and the average time w.r.t. to the probability of attacks. Concerning the ability to produce solutions, **CONArg2** is the solver that is mostly affected by the probability of attacks, and **PrefSAT** is the only one that solves all the considered instances. The average times trend confirms the observation made with regards to the IPC score: solvers are either quick on AFs with small or high probabilities of attacks between arguments, but not on both of them.

	% Solved				Average CPU-Time			
	25	50	75	RAND	25	50	75	RAND
CONArg2	97.8	100.0	100.0	97.2	87.4	11.0	7.1	59.6
AspartixM	98.3	100.0	100.0	98.9	56.5	14.7	10.0	34.0
PrefSAT	100.0	100.0	100.0	100.0	**5.1**	1.6	2.2	**4.2**
NAD-Alg	100.0	100.0	100.0	93.9	18.9	**0.2**	**0.2**	70.6

Table 2. Percentage of solved AFs and average runtime for each of the considered solvers, according to the percentages of attacks.

In order to better understand the performance of the considered solvers, we performed the Friedman test, and post-hoc analysis with Wilcoxon tests. The level of confidence we used has been modified to $p = 0.0125$ due to the usage of the Bonferroni's correction. The results of this analysis are shown in Table 3. We considered the performance of solvers w.r.t. the probabilities of attacks between arguments. Interestingly, the Friedman indicates that there is always a statistically significant difference between

the performance of the solvers, regardless to the probability of attacks. On the other hand, the post-hoc analysis performed with the Wilcoxon on solvers with similar runtime median, indicates that the performance difference is always significant but in the RAND set. This supports what we observed in Figure 2, i.e. in the RAND set similar-performing solvers show analogous behaviours. Thus, the Wilcoxon test also suggests that the "clusters" identified by the IPC score include solvers with significantly different performance.

	Median				Friedman		Wilcoxon A–C		Wilcoxon N–P	
	C	A	P	N	$\chi^2(3)$	p	Z	p	Z	p
ALL	2.0	3.9	0.4	0.2	1566.7	0.000	-3.8	0.000	-6.8	0.000
25	4.6	3.0	0.3	0.9	362.7	0.000	-3.8	0.000	-8.8	0.000
50	1.8	3.8	0.4	0.1	502.7	0.000	-4.3	0.000	-11.6	0.000
75	1.4	5.7	0.5	0.1	507.7	0.000	-7.1	0.000	-11.7	0.000
RAND	2.0	3.3	0.3	0.2	326.2	0.000	-0.6	0.585	-1.7	0.094

Table 3. Friedman and post-hoc analysis with Wilcoxon (Bonferroni correction — significance level set at $p < 0.0125$). In evidence the non-significant results. A, N, P and C stands respectively for **AspartixM**, **NAD-Alg**, **PrefSAT** and **CONArg2**.

5 Conclusions and Future Work

Recently, a number of high-performance algorithms have been proposed for computing preferred extensions. They exploit two approaches: (i) using a dedicated algorithm to obtain the problem solution, or (ii) translating the problem instance at hand into an equivalent instance of a different class of problems, for which solvers are already available, an then translate the solution back to the original problem.

This paper provided the first comparison of state-of-the-art approaches which transform the preferred enumeration problem into a CSP (**CONArg2**), ASP (**AspartixM**) and SAT (**PrefSAT**) with the best argumentation-dedicated approach **NAD-Alg**. Our experimental hypothesis was that there would be a strict ordering regarding the performance, in term of CPU-time required for enumerating the preferred extensions of given AFs, of the solvers. The experimental analysis, conducted on more than 700 AFs with increasing number of arguments and attacks, proved that the hypothesis was only partially true. While in most of the cases an order is respected, i.e. **NAD-Alg**, **PrefSAT**, **CONArg2** and **AspartixM**, we identified several cases in which: (1) **PrefSAT** has been the best implementation — and it is also the only one implementation that solved all the AFs considered in the experiment — and (2) **AspartixM** performed significantly — according to the Friedman statistic test confirmed by a post-hoc analysis with the Wilcoxon signed rank with a Bonferroni correction applied — better than **CONArg2**.

The take-home message of this work is that non-dedicated approaches can achieve similar (and sometimes better) performance than ad-hoc ones and, moreover, SAT-based algorithms experimentally demonstrated to be the fastest among the non-dedicated ones.

While **NAD-Alg** performs well in "constrained" AFs, w.r.t. the percentage of attacks, **PrefSAT** is faster and more reliable in instances with a low number of attacks. Clearly, there are many aspects that affect algorithms' performance, like data structures, procedures implementations, etc., thus it is hard to give definitive conclusions.

Future work include a larger experimental evaluation and, whether possible, the exploitation of a white-box approach. Looking at the design of the solvers can give further information on their performance, in particular can provide hints on procedures / design decisions that should be refined, and, potentially, drive to significant performance improvements for both CSP and not CSP-based systems.

Acknowledgements. The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

1. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowledge Engineering Review 26(4), 365–410 (2011)
2. Baroni, P., Giacomin, M.: Semantics of abstract argumentation systems. In: Argumentation in Artificial Intelligence, pp. 25–44. Springer (2009)
3. Baroni, P., Cerutti, F., Dunne, P.E., Giacomin, M.: Automata for Infinite Argumentation Structures. Artificial Intelligence 203(0), 104–150 (May 2013)
4. Bistarelli, S., Santini, F.: Modeling and solving afs with a constraint-based tool: Conarg. In: Modgil, S., Oren, N., Toni, F. (eds.) Theorie and Applications of Formal Argumentation, Lecture Notes in Computer Science, vol. 7132, pp. 99–116. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-29184-5_7
5. Caminada, M.: On the issue of reinstatement in argumentation. In: Proceedings of JELIA 2006. pp. 111–123 (2006)
6. Caminada, M., Gabbay, D.M.: A logical account of formal argumentation. Studia Logica (Special issue: new ideas in argumentation theory) 93(2–3), 109–145 (2009)
7. Cerutti, F., Dunne, P.E., Giacomin, M., Vallati, M.: Computing preferred extensions in abstract argumentation: A sat-based approach. In: Proceedings of Theory and Applications of Formal Argumentation (TAFA 2013). pp. 176–193 (2013)
8. Dimopoulos, Y., Nebel, B., Toni, F.: Preferred arguments are harder to compute than stable extensions. In: Proceedings of IJCAI 1999. pp. 36–43 (1999)
9. Dimopoulos, Y., Torres, A.: Graph theoretical structures in logic programs and default theories. Journal Theoretical Computer Science 170, 209–244 (1996)
10. Doutre, S., Mengin, J.: Preferred extensions of argumentation frameworks: Query answering and computation. In: Proceedings of IJCAR 2001. pp. 272–288 (2001)
11. Dung, P.M.: On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games. Artificial Intelligence 77(2), 321–357 (1995)
12. Dunne, P.E., Wooldridge, M.: Complexity of abstract argumentation. In: Argumentation in Artificial Intelligence, pp. 85–104. Springer (2009)
13. Dvořák, W., Gaggl, S.A., Wallner, J., Woltran, S.: Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. In: Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011) (2011)
14. Estrada, G.G.: A note on designing logical circuits using sat. In: Evolvable Systems: From Biology to Hardware, pp. 410–421 (2003)

15. Faber, W.: Answer set programming. In: Reasoning Web. Semantic Technologies for Intelligent Data Access, Lecture Notes in Computer Science, vol. 8067, pp. 162–193. Springer Berlin Heidelberg (2013)
16. Leone, N., Faber, W.: The dlv project: A tour from theory and research to applications and market. In: de la Banda, M.G., Pontelli, E. (eds.) Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings. Lecture Notes in Computer Science, vol. 5366, pp. 53–68. Springer (2008)
17. Modgil, S., Caminada, M.: Proof theories and algorithms for abstract argumentation frameworks. In: Argumentation in Artificial Intelligence, pp. 105–129. Springer (2009)
18. Nofal, S., Dunne, P.E., Atkinson, K.: On preferred extension enumeration in abstract argumentation. In: Proceedings of COMMA 2012. pp. 205–216 (2012)
19. Nofal, S., Atkinson, K., Dunne, P.E.: Algorithms for decision problems in argument systems under preferred semantics. Artificial Intelligence 207, 23–51 (2014)
20. Rintanen, J.: Engineering efficient planners with sat. In: the 20th European Conference on Artificial Intelligence (ECAI). pp. 684–689 (2012)
21. Rossi, F., van Beek, P., Walsh, T.: Chapter 4 constraint programming. In: Frank van Harmelen, V.L., Porter, B. (eds.) Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 181 – 211. Elsevier (2008), http://www.sciencedirect.com/science/article/pii/S1574652607030040
22. Shaffer, J., P.: Multiple hypothesis testing. Annual Review of Psych 46, 561–584 (1995)
23. Weber, T.: A sat-based sudoku solver. In: 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR). pp. 11–15 (2005)