The Knowledge Engineering Review, Vol. 34, e6, 1–23. © Cambridge University Press, 2019
doi:10.1017/S0269888918000036

Predictive models and abstract argumentation: the case of high-complexity semantics

MAURO VALLATI[1], FEDERICO CERUTTI[2] and MASSIMILIANO GIACOMIN[3]

[1]*School of Computing & Engineering, University of Huddersfield, Huddersfield HD1 3DH, UK;*
e-mail: m.vallati@hud.ac.uk;
[2]*School of Computer Science & Informatics, Cardiff University, Cardiff CF24 3AA, UK;*
e-mail: CeruttiF@cardiff.ac.uk;
[3]*Dipartimento di Ingegneria dell'Infomazione, Università degli Studi di Brescia, via Branze 38, Brescia, Italy;*
e-mail: massimiliano.giacomin@unibs.it

Abstract

In this paper, we describe how predictive models can be positively exploited in abstract argumentation. In particular, we present two main sets of results. On one side, we show that predictive models are effective for performing algorithm selection in order to determine which approach is better to enumerate the preferred extensions of a given argumentation framework. On the other side, we show that predictive models predict significant aspects of the solution to the preferred extensions enumeration problem. By exploiting an extensive set of argumentation framework features—that is, values that summarize a potentially important property of a framework—the proposed approach is able to provide an accurate prediction about which algorithm would be faster on a given problem instance, as well as of the structure of the solution, where the complete knowledge of such structure would require a computationally hard problem to be solved. Improving the ability of existing argumentation-based systems to support human sense-making and decision processes is just one of the possible exploitations of such knowledge obtained in an inexpensive way.

1 Introduction

Dung's theory of abstract argumentation (Dung, 1995) is a unifying framework able to encompass a large variety of specific formalisms in the areas of non-monotonic 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*, that is sets of arguments that can 'survive the conflict together'. In Dung (1995) 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 admissibility-based semantics. The investigation on alternative argumentation semantics is an active research area since two decades (Baroni *et al*., 2011).

Several problems are associated to each semantics, notably *credulous* and *skeptical* acceptance of an argument with respect to a given AF—that is, determining whether an argument belongs to at least one (respectively, every) extension—and *enumeration* of *all* or *some* extensions given an AF.

In this work we focus on preferred semantics. It represents one of the main contributions in Dung's theory and is widely adopted—among other areas—in decision support systems (Tamani *et al*., 2015) and in critical thinking support systems (Toniolo *et al*., 2015), as it allows multiple extensions (differently from grounded semantics), the existence of extensions is always guaranteed (differently from stable semantics),

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

and no extension is a proper subset of another extension. On the other hand, grounded semantics prescribes a unique extension which can be computed in polynomial time, thus all the problems related to grounded semantics are easy to solve. As to stable semantics, enumerating stable extensions is known to be equivalent to solve a Satisfiability (SAT) problem, and thus techniques known in the literature for SAT problems (e.g. Gomes *et al*., 2009) can be re-applied to this case.

Several problems associated to preferred semantics are computationally difficult. In particular, (1) determining if there is at least one non-empty preferred extension is NP-complete; (2) counting the number of preferred extensions is #P-complete; and (3) determining whether an argument belongs to each preferred extension, that is, skeptical acceptance, is Π_2^p-COMPLETE (Dunne & Wooldridge, 2009; Baroni *et al*., 2010; Kröll *et al*., 2017).

Despite the importance of preferred semantics, and the computational complexity of several problems associated to it, few work has been done in argumentation for investigating the exploitability of machine learning techniques—and particularly empirical predictive models (EPMs)—for supporting computationally complex reasoning problems.

In order to fill this lack of studies in the context of computational models of argumentation, in this work we provide an extensive investigation of the exploitability of automatically generated predictions in dealing with high-complexity problems. Specifically, we analyze the ability of predictive models in (i) given a set of solvers for the preferred extensions enumeration problem, predicting the fastest one, as well as the amount of central processing unit (CPU)-time each solver will need to solve an instance of the problem; and (ii) predicting characteristics of the solution to the problem—without the need to solve it. These characteristics include (1) whether the framework has a single empty preferred extension; and (2) whether the framework has a number of extensions exceeding a given threshold representing the limit for an acceptable burden on the user; or (3) the (estimated) number of preferred extensions.

Our contribution encompasses previous works exploiting features in abstract argumentation (notably, Cerutti *et al*., 2014a; Vallati *et al*., 2014). Specifically, in this paper:

1. we empirically demonstrate the usefulness of models for predicting solvers performance in solving the enumeration problem associated to the preferred semantics;
2. we introduce a hierarchical predictive model for determining characteristics of the solution of the enumeration problem associated to the preferred semantics;
3. we demonstrate that the proposed hierarchical model is able to provide very accurate predictions, without solving the enumeration problem;
4. we expand the existing set of features used from 50 to 147. Specifically, we added 97 features derived from a matrix representation of AFs;
5. we provide a new, sound, methodology for deriving structured AFs.

The importance of this contribution is twofold. From a theoretical point of view, it explains the runtime variations between different approaches, and links AFs characteristics to the solutions of abstract argumentation problems. From a practical perspective, the proposed analysis provides tools for choosing among a set of available algorithms, in order to achieve the best performance from them, and techniques to predict information that can have a significant impact on argumentation applications. For instance, (1) and (2) can be exploited in cases where the knowledge base changes frequently, the set of arguments needs to be frequently updated and the extensions need to be frequently computed to evaluate the acceptability of each proposition in the knowledge base. In such cases, the acceptability of propositions might have been already changed by the time the answer is computed, thus the predictions can guide the decision on when the computation procedure should be triggered. Moreover, estimating the number of extensions (3) may be useful to get a rough assessment of the uncertainty inherent to an AF, to assess the proportion of acceptance of an argument with respect to the total number of extensions, or to identify a bound for an enumeration algorithm (Baroni *et al*., 2010).

In order to provide predictions, models exploit features extracted from target AFs. Features are real numbers that summarize potentially important properties of the AF, therefore a predictive model can be seen as a mapping from instance features to predictions (solvers performance or solutions structure). This work is based on prior well-received work, such as Cerutti *et al*. (2014a); here we establish the most extensive set of features for performing predictions in abstract argumentation.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

The paper is organized as follows. Section 2 discusses related work. In Section 3 we provide the required background on abstract argumentation. Then, in Section 4, we provide the list of the features of AFs that are exploited by predictive models. In Section 5 we present the machine learning models to evaluate and select algorithms for enumerating preferred extension, and we analyze them by providing some experimental results. Then, in Section 6, we turn to the models that predict properties of extensions given the chosen set of features, and again we empirically evaluate them. Finally, we provide conclusions and describe future developments in Section 7.

2 Related work

In the field of automated reasoning, the idea of generating models that, according to the values of some features, allow to identify the best algorithm(s) for an instance of a certain problem has been investigated by several researchers. For example, EPMs have been extensively and successfully applied in numerous areas of Artificial Intelligence, such as SAT problem, Mixed Integer Programming (MIP), Answer Set Programming (ASP), Quantified Boolean Formulae (QBF), travelling salesman problem (TSP), and Automated Planning (Brewer, 1994; Nudelman *et al*., 2004; Xu *et al*., 2008; Leyton-Brown *et al*., 2009; Smith-Miles *et al*., 2010; Gebser *et al*., 2011; Fawcett *et al*., 2014).

The work described by Fawcett *et al*. (2014) is focussed on generating models for accurately predicting planners runtime. Such models exploit a large set of instance features, derived from different representations of a given planning problem, and (short) runs of planners. Their experimental results show that the generated performance models are able to produce very accurate runtime predictions.

SATZilla is a prominent example of an algorithm portfolio designed for SAT (Xu *et al*., 2008). It exploits regression techniques to build a predictor of the runtime of a number of provided SAT solvers. For solving a new SAT problem, SATZilla computes the values of a large set of features, predicts the performance of the considered SAT solvers. The solvers are then ordered according to the predicted runtime, and executed for the predicted time, following the established ordering. It should be noted that, for the sake of overall performance, SATZilla exploits also pre- and backup solvers: the former is used before extracting features, in order to quickly solve trivial instances. The latter is run when all the selected solvers failed.

Similarly, Claspfolio (Gebser *et al*., 2011) exploits regression-based EPMs for selecting, among a range of predefined configurations of the well-known ASP solver Clasp (Gebser *et al*., 2007), the best configuration to minimize the runtime on a given ASP instances. Predictions are made according to a set of features that are extracted from the considered ASP problem. Claspfolio 2 (Hoos *et al*., 2014) is an improved version of Claspfolio, that provides a modular architecture that allows for integrating several different approaches and techniques for extracting features, predicting solvers' performance, and combine solvers into a portfolio.

Matos *et al*. (2008) propose an algorithm portfolio solving the MaxSAT problem. According to the values of several features, it estimates the runtime of each incorporated solver, and then solves the instance with the estimated fastest solver. The estimation is done using a (linear) model configured by performing ridge regression (Marquardt & Snee, 1975).

Similarly, Pulina and Tacchella (2007) study an algorithm portfolio solving the QBF problem. They identify some features of the QBF problem, and investigate the usage of four inductive models to select the best solver to use according to the values of the identified features. Maratea *et al*. (2014) applied a similar approach for solving ASP problems: given a set of easy-to-compute features, extracted from the given instance, the authors propose a classification-based approach for selecting the most promising solver to run in order to minimize the required runtime.

3 Background on Dung's Argumentation framework

An AF (Dung, 1995) consists of a set of arguments and a binary attack relation between them[1].

[1] In this paper, we consider only *finite* sets of arguments: see Baroni *et al*. (2013) for a discussion on infinite sets of arguments.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

DEFINITION 1 *An 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* $\boldsymbol{b}$ *attacks* $\boldsymbol{a}$ *iff* $\langle b, a \rangle \in \mathcal{R}$, *also denoted as* $\boldsymbol{b} \rightarrow \boldsymbol{a}$.

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* $\nexists\ \boldsymbol{a}, \boldsymbol{b} \in S$ *s.t.* $\boldsymbol{a} \rightarrow \boldsymbol{b}$;
- *an argument* $a \in \mathcal{A}$ *is acceptable with respect to a set* $S \subseteq \mathcal{A}$ of Γ *if* $\forall b \in \mathcal{A}$ *s.t.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$, $\exists\, c \in S$ *s.t.* $\boldsymbol{c} \rightarrow \boldsymbol{b}$;
- *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 definition of preferred (denoted as PR) semantics only.

DEFINITION 3 *Given an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$: *a set* $S \subseteq \mathcal{A}$ *is a preferred extension of* Γ, *that is* $S \in \mathcal{E}_{\text{PR}}(\Gamma)$, *iff* S *is a maximal (w.r.t. set inclusion) admissible set of* Γ.

Problems considered: For our purpose, given a semantics σ and an AF Γ, let us consider the following problems:

1. Ex_σ: determining if $\mathcal{E}_\sigma(\Gamma) \neq \emptyset$;
2. Ne_σ : determining if $\mathcal{E}_\sigma(\Gamma) \notin \{\emptyset, \{\emptyset\}\}$[2];
3. Count_σ: determining $|\mathcal{E}_\sigma(\Gamma)|$;
4. Enum_σ: enumerating $\mathcal{E}_\sigma(\Gamma)$.

In the case of the preferred semantics, it is known (Dunne & Wooldridge, 2009; Baroni *et al*., 2010; Kröll *et al*., 2017) that Ex_{PR} is trivial, that is the existence of at least a preferred extension is always guaranteed; Ne_{PR} is NP-COMPLETE; Count_{PR} is #P-COMPLETE[3]. We are not aware of any study on the computational complexity of determining if $|\mathcal{E}_\sigma(\Gamma)| \leq \chi$.

4 Features

In order to perform predictions, EPMs need to extract information from the considered AFs. Such information are provided under the form of features. A feature is either a real number or a Boolean value, that summarizes a potentially important property of an AF.

Our feature set includes 147 values, which exploit the representation of AFs as a graph or as a matrix.

4.1 Graph-related features

We are able to extract 50 features by representing an AF both as directed graph (DG)and as undirected graph (UG) (lossy) —following the notation introduced in Cerutti *et al*. (2014a).

In total, 26 features are extracted from the representation of AFs as DG. Each feature belongs to one of the following four classes: *graph size* (5 features), *degree* (4), Strongly Connected Components (*SCC*) (5), *graph structure* (5), *CPU-times* (7).

- *Graph size features*: number of vertices, number of edges, ratios vertices-edges and inverse, and graph density.
- *Degree features*: average, standard deviation, maximum, minimum degree values across the nodes in the graph.
- *SCC features*: number of SCCs, average, standard deviation, maximum, and minimum size of SCCs.

[2] In the case of $\sigma = \text{PR}$, this is equivalent to write $\mathcal{E}_{\text{PR}}(\Gamma) \notin \{\{\emptyset\}\}$, because the existence of a preferred extensions is always guaranteed, and thus it cannot be the case that $\mathcal{E}_{\text{PR}}(\Gamma) = \emptyset$.

[3] This class of problems was introduced by Valiant (1979).

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

- *Graph structure*: presence of auto-loops, number of isolated vertices, flow hierarchy, and results of test on Eulerian and aperiodic structure of the graph.
- *CPU-times*: the needed CPU-time for extracting each class of features.In our implementation, *graph transitivity* is computed as the fraction of all the possible triangles which is present in the considered graph. Possible (potential) triangles are identified by the number of 'triads', that is two edges with a shared vertex. A graph is *Eulerian* if there exists a path (starting and ending at the same vertex) that allows to visit every edge exactly once. A given graph is *Aperiodic* if there is no integer $k > 1$ that divides the length of every cycle. *Flow hierarchy* is computed as the fraction of edges not participating in cycles in a DG (Luo & Magee, 2011).The features extracted by the UG representation of AFs—that is replacing each directed attack with an undirected edge—and distinct from the DG features are 24, belonging to six classes: *graph size* (4), *degree* (4), *components* (5), *graph structure* (1), *triangles* (5), *CPU-times* (5).

- *Graph size features*: number of edges, ratios vertices-edges and inverse, and graph density.
- *Degree features*: average, standard deviation, maximum, minimum degree values across the nodes in the graph.
- *Components features*: number of connected components, average, standard deviation, maximum, and minimum size.
- *Graph structure*: transitivity of the graph.
- *Triangles features*: total number of triangles in the graph and average, standard deviation, maximum, minimum number of triangles per vertex.
- *CPU-times*: the needed CPU-time for extracting each class of features.

4.2 Matrix-related features

Any AF with N arguments can be encoded as an $N \times N$ matrix as follows. The value of a cell (a,b) is 1 if there exists an attack from argument a to argument b, 0 otherwise[4]. Given this encoding we are able to extract 97 features, divided in:

- *basic matrix* (9): includes information about rank, number of non-zeros, norm, etc.;
- *sum* (14): includes statistics, like standard deviation, mean, etc., on vectors resulting from sum of either columns or rows;
- *cumulative sum* (21): includes statistics, like standard deviation, mean, etc., on vectors resulting from cumulative sum of columns, rows, and diagonals;
- *differences* (21): includes statistics, like standard deviation, mean, etc., on vectors resulting from differences of cells on either columns, rows, and diagonals;
- *diagonals* (7): includes general information extracted by considering the first and second diagonal of the matrix;
- *Gradient* (21): includes statistics extracted by considering the gradient of the vector resulting by (i) the sum of the rows, (ii) the sum of the columns, and (iii) the sum of the diagonals of the matrix;
- *Correlation* (4): includes the correlation values between the sum of columns, sum of rows, sum of diagonals, and gradients.

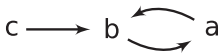
4.3 Example of extracted features

To illustrate the process of features extraction, let us consider an intuitive example from Besnard and Hunter (2014). Let **a** be the argument 'Patient has hypertension so prescribe diuretics'; **b:** 'Patient has hypertension so prescribe betablockers'; and **c:** 'Patient has emphysema which is a contraindication for betablockers'. Intuitively, assuming that only one treatment is possible at the very same time, **a** attacks **b** and vice versa, while **c** suggests that **b** should not be the case (**c** attacks **b**). Therefore, let $\Gamma_M = \langle \mathcal{A}_M, \mathcal{R}_M \rangle$ such that, $\mathcal{A}_M = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\mathcal{R}_M = \{\langle \mathbf{c}, \mathbf{b} \rangle, \langle \mathbf{b}, \mathbf{a} \rangle, \langle \mathbf{a}, \mathbf{b} \rangle\}$. Γ_M is depicted in Figure 1.

Table 1 presents a subset of the extracted features, and their extracted value on the considered example. For the sake of readability, we focus on less intuitive features. Features such as the number of arguments or the number of attacks are trivial to identify and extract.

[4] For the sake of readability, the complete list of features is provided in Appendix A.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press



	a	b	c
a	0	1	0
b	1	0	0
c	0	1	0

Figure 1 The argumentation framework (AF) Γ_M for the hypertension problem on the left, and its matrix encoding on the right

Table 1 A subset of the features extracted by considering the directed graph, undirected graph, and matrix representation of the argumentation framework Γ_M for the hypertension problem

Directed		Undirected		Matrix	
Feature	Value	Feature	Value	Feature	Value
Degree max., min.	3, 1	Degree max., min.	2, 1	Column sum max	2
Isolated vertices	0	Density	0.6	Rank	2
Eulerian	*False*	Transitivity	0.0	Sparse	*False*
Flow hierarchy	0.3	Triangles	0	Sum diagonals	0

5 Exploiting predictive models for performing algorithm selection

This section is devoted to investigate the use of predictive models for performing algorithm selection, in order to combine different systems into a portfolio and improve the overall performance in terms of both coverage (i.e. percentage of problem instances actually solved) and runtime. For this analysis we considered only the problem of enumerating preferred extensions.

5.1 Algorithms for solving the preferred extensions enumeration problem

In this paper we consider four approaches for solving the problem of preferred extensions enumeration: **Aspartix, ArgTools, ArgSemSAT**, and **SCC-P**. They have been chosen as representative of the variety of approaches used to enumerate preferred extensions.

First, **Aspartix** (Dvořák *et al*., 2011) expresses argumentation semantics in ASP: a single program is used to encode a particular argumentation semantics, and the instance of an AF is given as an input database. Tests for subset-maximality exploit the `metasp` optimization frontend for the ASP-package `gringo/claspD`[5].

Instead, **ArgTools** (Nofal *et al*., 2014) 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*, that is 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 neighbours is performed. Then the procedure recursively calls itself on the assumption that the above arguments are respectively *in* or *out* the extension. Any inconsistency leads to a backtrack[6].

On the other hand, **ArgSemSAT** (Cerutti *et al*., 2013, 2017) performs a search in the space of complete extensions to enumerate the maximal ones. In particular, ArgSemSATencodes the constraints corresponding to complete extensions into a propositional formula. A SAT-solver is then used to determine an assignment of propositional variables satisfying the formula, thus returning a complete extension. A depth-first technique is then applied in order to determine a maximal complete extension containing the one already found—that is, a preferred extension. Previously explored search states are excluded from further

[5] **Aspartix** has been executed with `gringo` version 3.0.3 and `claspD` version 1.1.4.

[6] **ArgTools's** implementation is available at `https://sourceforge.net/projects/argtools/files/?source=navbar`

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

exploration by adding specific constraints to the encoding of complete extensions. **ArgSemSAT** is, together with **Cegartix** (Dvorák *et al*., 2015)—a similar system exploiting a SAT-solver as a NP-oracle—the best system for addressing problems associated to preferred semantics as certified by the First International Competition on Computational Models of Argumentation (ICCMA) 2015 (Thimm *et al*., 2016).

Finally, **SCC-P** is an instantiation of the meta-algorithm proposed in Cerutti *et al*. (2014c) which exploits the SCC-recursiveness schema (Baroni *et al*., 2005): it recursively decomposes a framework so as to compute extensions on restricted sub-frameworks, in order to reduce the computational effort. At the beginning, the extensions of the framework restricted to the initial SCCs, that is, those not receiving attacks from others, are computed and combined together. Then each SCC which is attacked only from initial SCCs is considered, and for each extension already obtained, the extensions of such a SCC are locally computed and merged with it. The process is then applied to all SCCs following their partial order, until no remaining SCCs are left to process. Moreover, as the name suggests, the algorithm exploits a recursive procedure for the local computation in every SCC, according to the following steps. First, all arguments attacked by the extension selected in the previous SCCs are suppressed. Then, the procedure is recursively applied to the remaining part of the SCC. When the base of the recursion is reached, a specific 'base algorithm' is called. Such base algorithm can be obtained by generalizing existing algorithms that enumerate extensions so as to work on restricted sub-frameworks, for example, a variation of **ArgSemSAT** as proposed in Cerutti *et al*. (2014c).

It should be noted that, while a larger number of solvers can be considered in the general EPMs framework, the focus of our work is on identifying the exploitability of predictive models for effectively estimating the runtime needed for solving a given AF and, as a side effect, how such information can be used for selecting the most appropriate solver. For this reason, and for allowing a more in-depth analysis of features, we selected a restricted number of solvers.

5.2 Experimental protocol

We randomly generated a set of 10 000 AFs using the generator described in Cerutti *et al*. (2014b), varying the number of SCCs between 0 and 100, the number of arguments between 10 and 5000; and then considering four classes of AFs with uniform probability of attacks—that is, with the probability of having an attack between two arguments respectively of 0.25; 0.5; 0.75; and completely random between 0 and 1—leading to AFs with a number of attacks between 35 and (approximately) 270 000. This was done in order to have, for each selected solver, unsolved AFs as well as AFs solved quickly (few seconds) or in hundreds of seconds. Such a wide distribution is extremely informative for effectively training EPMs. Table 2 lists some of the relevant characteristics of the generated 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. A cutoff of 900 seconds was imposed to compute the preferred extensions for each AF. For each solver we recorded the overall result: success (if it finds each preferred extension), crashed, timed-out, or ran out of memory. Unsuccessful runs—crashed, timed-out, or out of memory—were assigned a runtime equal to the cutoff.

We observed that graph-related features are usually quick to compute: on average, the extraction process takes around 3 CPU-time seconds on AFs considered in this experimental analysis. However, extracting information about the aperiodicity of the graph can require up to 20 seconds—even though that happened only once. On the other hand, matrix-related features require a higher CPU-time to be properly extracted: on average, 7 CPU-time seconds on the considered AFs. Given the fact that enumerating the preferred extensions of an AF can require between tens and hundreds of seconds, we decided to focus on the graph-related sets, as they are much quicker to extract.

We considered EPMs for both *classification* and *regression* approaches. Classification approaches classify the AF into a single category, corresponding to the algorithm which is predicted to be the fastest. These approaches do not estimate the performance difference between solvers, rather they only look for the best one. Regression techniques model the behaviour of each algorithm in order to predict its runtime on a new AF.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Table 2 Characteristics of the considered argumentation frameworks.

	Average	Minimum	Maximum
#Attacks	58 487	35	270 118
#Arguments	1941	16	5000
Density	0.39	0.01	10.18
Degree	60	4	171
#SCC	50	0	100
SCC size	40	1	175

SCC = Strongly Connected Components.

We first assessed the performance of various classification and regression models, using the WEKA tool (Hall *et al*., 2009). We considered well-known machine learning techniques: linear regression, neural networks, Gaussian processes, decision trees, and rule-based techniques. The EPMs were evaluated using a 10-fold cross-validation approach on a uniform random permutation of our instances—a standard method where nine slices are used for training and the tenth for testing. According to this analysis, we observed that random forests perform best in classification, and M5-rules in regression. For both classification and regression, the second best approach is based on decision tables (Kohavi, 1995). In classification, decision tables achieve accuracy performance that is 4% worse than random forest; in regression, the root mean square error (RMSE) is increased, on average, by 0.09.

We then assessed the importance of features by means of a greedy forward search based on the Correlation-Based Feature Subset Selection (CFS) attribute evaluator (Hall, 1998), which determines the most relevant subset of features for the task at hand. In short, the search starts from an empty subset of features. At each iteration, the feature that increases the most the evaluation of the current subset is added. The search stops when the addition of any attribute results in a decrease of the subset evaluation. The evaluation of a subset of features considers the predictive ability of each feature along with the degree of redundancy between them (Hall, 1998). Given the results shown in Cerutti *et al*. (2014c), SCC-related information are believed to be extremely informative, thus expected to play a significant role in the EPMs performance. However, it has to be pointed out that the relation between the importance of the features exploited by an EPM and its performance is not deterministic.

Finally, the EPMs have been generated considering seven different sets of features:

- the set including the single best feature (B1), the sets including the best two and three features (B2, B3);
- the set including the features extracted from the DG;
- the set including those extracted from UG;
- the set with all the extracted features (*All*).

It turns out that B1, B2, and B3 include only features from the DG.

5.3 Runtime prediction

As to the M5-rules model to predict solvers runtime, the three best features selected by the greedy approach are summarized in Table 3. This analysis provides several insights on features that are expected to affect solvers performance. First, the density of DG—that is how close it is to a complete graph—seems strongly related to the performance of all the solvers since it is always among the best three features. Second, we observe that also SCC-related information, mainly the number of SCCs and the size of the maximum one, are often selected.

Table 4 shows the results, in terms of RMSE, for regression with 10-fold cross-validation on a uniform random permutation of our full set of 10 000 AFs. Since solvers runtimes vary from 0.01 to 900 CPU seconds, we trained our regression models to predict log-runtime rather than absolute runtime: this have been effective in similar circumstances (Hutter *et al*., 2014).

Results shown in Table 4 indicate that using all the extracted features leads to the best possible results. Some solvers seem to have easier to predict behaviours, while others are somehow more complex. It is

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Table 3 Best features selected for each solver, by the greedy approach, for predicting runtime

Solver	First feature (B1)	Second feature (B2)	Third feature (B3)
Aspartix	Number of nodes	Density of directed graph	Size of max. SCC
ArgSemSAT	Density of directed graph	Number of SCCs	Aperiodicity
ArgTools	Density of directed graph	CPU-time for density	CPU-time for Eulerian
SCC-P	Density of directed graph	Number of SCCs	Size of the max. SCC

SCC = Strongly Connected Components.

Table 4 Cross-validated root mean square error of $\log_{10}$(runtime) for M5-rules models using different features subsets

	Regression (lower is better)						
	B1	B2	B3	DG	UG	SCC	*All*
Aspartix	0.66	0.49	0.49	0.48	0.49	0.52	**0.48**
ArgSemSAT	1.39	0.93	0.93	0.89	0.92	0.94	**0.89**
ArgTools	1.48	1.47	1.47	0.77	0.57	1.61	**0.55**
SCC-P	1.36	0.80	0.78	0.75	0.75	0.79	**0.74**

B1, B2, B3 indicates the best 1, 2, and 3 features according to a greedy forward search; DG (UG) indicates features extracted only from directed (undirected) representation; SCC (Strongly Connected Components) only SCC-related ones; *All* indicates all the extracted features.
Values in bold indicate the best results, also considering hidden decimals.

interesting to note that DG and UG features lead to similar predictive performances, which are usually close to those achieved by exploiting the B3 set. Moreover, DG features allow to achieve an RMSE which is very close to the one achievable by exploiting the *All* set for **Aspartix, SCC-P**, and **ArgSemSAT**. Interestingly, this is not true for **ArgTools**. In this case the UG set of features provides more useful information for EPMS than DG. Moreover, SCC-related features are not informative for predicting the performance of **ArgTools**, and this is confirmed also by the selected three best features (Table 3). Finally, we noted that the behaviours of **ArgSemSAT** and **SCC-P** appear to be hard to predict, when compared to the other considered solvers. A possible explanation is due to the fact that **ArgTools** ran out of time on most of the AFs, and **Aspartix** solved a significant number of them in 800–899 CPU-time seconds. **ArgSemSAT** and **SCC-P** show different, and more variegated, CPU-time distributions.

5.4 Classification

Differently from the regression EPMS, which generate a predictive model for each considered solver, the classification approach trains a single model that selects the most performant solver.

In this case, the three best features selected for the classification EPM are, in the order, (1) the number of vertices, (2) the density of the DG, and (3) the minimum degree value of the DG. Table 5 shows the results, in terms of overall accuracy and per-algorithm precision, of the classification EPM, evaluated with 10-fold cross-validation. From the set of 10 000 AFs, those which were not solved by any solver were removed. In this approach, an AF is a member of the class corresponding to the solver which has been the fastest in enumerating preferred extensions.

Accuracy indicates the proportion of correctly predicted results in the whole data set, while precision considers only a single class, and is the proportion of the elements correctly classified as members of the class against all the actual members of the class.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Table 5 Evaluation of classification empirical predictive models for different features subsets

	Classification (Higher is better)						
	B1 (%)	B2 (%)	B3 (%)	DG (%)	UG (%)	SCC (%)	*All* (%)
Accuracy	48.5	70.1	69.9	78.9	79.0	55.3	**79.5**
Precision **Aspartix**	35.0	64.6	63.7	74.5	74.9	42.2	**76.1**
Precision **ArgSemSAT**	53.7	67.8	68.1	79.6	**80.5**	60.4	80.1
Precision **ArgTools**	26.5	69.2	69.0	81.7	85.1	35.3	**86.0**
Precision **SCC-P**	54.3	73.0	72.7	76.6	76.8	57.8	**77.2**

B1, B2, B3 indicates the best 1, 2, and 3 features according to a greedy forward search, namely: (1) the number of vertices, (2) the density of the directed graph and (3) the minimum degree value of the directed graph; DG (UG) indicates features extracted only from directed (undirected) representation; SCC (Strongly Connected Components) only SCC-related ones; *All* indicates all the extracted features. A precision value smaller than 50% indicates that most of the argumentation frameworks have been mistakenly classified as members of the class.
Values in bold indicate the best results, also considering hidden decimals.

Several considerations can be drawn from Table 5:

- the whole set of features usually allows the EPM to achieve the best predicting performance;
- using the best two features instead of the single best one guarantees better performance;
- SCC-related features are not as informative as we believed;
- features derived from UG are usually better than those generated by considering DG. This counter-intuitive result (UG are lossy) is further addressed in Section 5.6. However, no features from the UG set are included in the B3 set: this seems to suggest that UG features are useful when exploited all together, while DG ones seem to be better when considered singularly.

As mentioned above, the greedy approach used for determining the most informative features indicated the number of vertices as the best one. Figure 2 shows how the number of vertices of AFs affect the performance of the considered solvers. Given the precision performance reported in Table 5, and the shape of the graph in Figure 2, we can argue that this feature is quite informative for selecting between **SCC-P** and **ArgSemSAT**, which are the fastest systems on most of the considered AFs, but does not provide very reliable information about the other solvers.

5.5 Algorithm selection

After evaluating the predicting performance of classification and regression EPMs, we compared them in order to understand which is the most promising approach to be used for online algorithm selection. Algorithm selection done by exploiting the classification EPM is straightforward: the best predicted solver is used for solving the testing AF. Instead, regression EPMs require to predict the runtime of each considered solver, and to select the one predicted to be the fastest. Let us notice that the CPU-time required for generating the predictions, in both classification and regression EPMs, is extremely low. Such predictions are generated by solving easy case-based formulas that test a feature value at a time. The most expensive step is the generation of the predictive models, which requires also to solve the training problems and extract the features from all of them; this is done offline, thus it does not affect the algorithm selection performance.

For this comparison we used the set of *All* the considered features, and divided the 10 000 AFs in separate training and testing instances. To achieve a more objective comparison between the techniques, the testing instances have been randomly selected across all the different sizes, resulting in 1040 AFs. All the solvers failed to enumerate the preferred extensions before the cutoff time on 33 AFs over the 1040 test instances: these instances were thus excluded from the following evaluation.

For each AF we considered (1) the fastest solver and (2) whether or not either the regression or the classification EPMs successfully select such a solver, counting the times that this happened. The result is

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

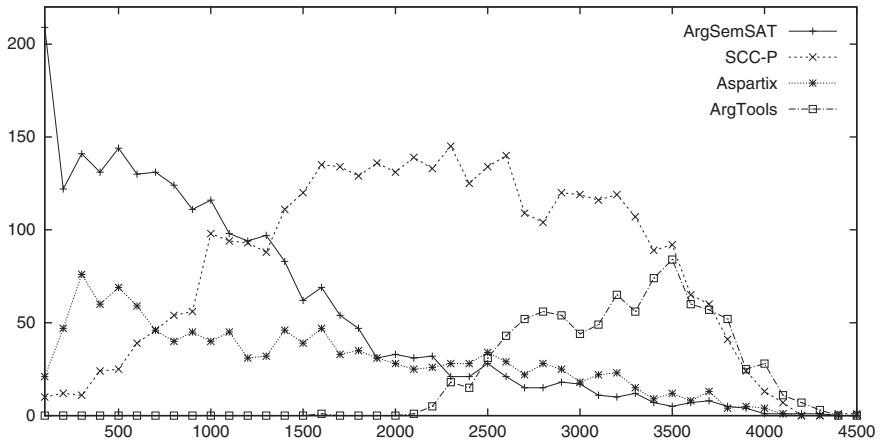


Figure 2 The number of time each solver has been the fastest one w.r.t. the cardinality of the set of arguments. SCC = Strongly Connected Components

Table 6 Algorithm selection performance—w.r.t. single solvers (upper part)—using two metrics: the number of times each solver and the selected solver has been the fastest one on the testing set of argumentation frameworks (AFs) (Fastest); and the International Planning Competition (IPC) value achieved by each solver and selected solver

Metric fastest (max. 1007)		Metric IPC (max. 1007, log scale)	
Aspartix	106	**ArgTools**	210.1
ArgTools	170	**Aspartix**	288.3
ArgSemSAT	278	**ArgSemSAT**	546.7
SCC-P	453	**SCC-P**	662.4
EPMs regression	755	EPMs regression	887.7
EPMs classification	788	EPMs classification	928.1

SCC = Strongly Connected Components; EPM = empirical predictive models.
The approaches are ordered from the worse to the best (according to each metric).

shown in the left part of Table 6. There is not a great difference between classification and regression-based algorithm selection; the former is able to select the best algorithm on 788 testing AFs (78%, clearly consistent with Table 5), while the latter on 755 (75%). In most of the cases (834 times) both the EPMs select the same algorithm, and in 687 cases such selection was right. It is worth noting that there is a remarkable difference between the performance of algorithm selection approaches and the single solvers, that is each single solver achieves the best performance in a greatly lower number of AFs w.r.t. the solver predicted by any of the two algorithm selection approaches.

It can be argued that the metric used, that is the number of time a system has been the fastest, albeit good for evaluating the algorithm selection performance, might not be very informative from the performance gain perspective. Thus, we analyzed the International Planning Competition (IPU) score, borrowed from the 2008 International Planning Competition[7]. For each AF, each system gets a score of T^*/T, where T is its execution time and T^* the best execution time among the compared systems, or a score of 0 if it fails in that case. Runtimes below 0.01 seconds get by default the maximal score of 1. The IPC score is interesting because it considers, at the same time, the runtimes and the number of solved instances. The maximum IPC score achievable on the testing AFs is 1007, which corresponds to a system that always selects the best solver on the 1007 instances where at least a solver succeeded. Clearly, on AFs where both the EPMs selected the same algorithm, they obtain the same IPC score. According to IPC score, the classification-based algorithm selection approach scores 928.1, followed by the regression EPMs with a score of 887.7.

[7] `http://ipc.informatik.uni-freiburg.de/`

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

SCC-P and **ArgSemSAT** score, respectively, 662.4 and 546.7; finally **Aspartix** scores 288.3 and **ArgTools** 210.1 (right part of Table 6). Also by comparing the systems using the IPC score, a significant difference can be found between algorithm selection methods and single solvers.

5.6 Discussion

Remarkably, the EPMs which consider features derived from UG—that is without considering the directionality of attacks—are usually better than those generated by considering DG ones (cf. Table 5). Since preferred semantics satisfies the *directionality* principle (Baroni & Giacomin, 2007), one could see this as a counter-intuitive result. However, it is worth mentioning that both **Aspartix** and **ArgTools** have some recurrent behaviours—the former often ends the enumeration around 800–900 seconds, the latter often do not end the enumeration in the given time—that turn out to be easily predictable. It is possible that these behaviours are affected just by the features of the UG. Nevertheless, this fails to explain the behaviour of **ArgSemSAT** and **SCC-P**. As to **ArgSemSAT**, it has to be remarked that its performance depends on the performance of the SAT-solver used, in this case GLUCOSE (Audemard & Simon, 2014). Therefore, further investigations on the translation into SAT problems and its dependency from the AF structure should be considered. Moreover, in Cerutti *et al*. (2013) we show the dramatic performance difference due to the different *encoding* of complete semantics in SAT problems: an additional investigation on this topic is already envisaged. As to SCC-P, the (little) advantage exhibited by the features derived from the UG seems to be consistent with the meta-algorithm based on the SCC-recursiveness schema (Baroni *et al*., 2005), which allows to apply the SAT-solver just to SCCs.

6 Exploiting predictive models for estimating the number of preferred extensions

The aim of this investigation is to exploit predictive models for providing in 'almost real-time' (i.e. without solving the enumeration problem) an estimation of the number of preferred extensions of a given AF. Specifically, we are interested in determining (1) the solution to the non-emptiness problem (a NP-complete problem), and (2) whether there is more than one preferred extension, with an estimation of the relevant number. Moreover, we introduce an additional theoretical problem, that is 'does the preferred semantics identify less than $\overline{X}$ extensions?', where $\overline{X}$ is a configurable, arbitrary, but fixed, parameter[8].

6.1 Experimental protocol

The first step of our experimental protocol consists in the generation of an initial data set. To this purpose, we first generated a large number of AFs by using an improved version of **AFBenchGen** (Cerutti *et al*., 2014b). In this analysis, differently from experiments in Section 5, our focus is on generating AFs with very different number of preferred extensions, in order to check if EPMs are able to accurately predict the number of extensions of a given framework.

For the purpose of training the models, and maximizing the diversity of AFs in terms of the number of extensions, we randomly generated three sets of 4000 AFs, each one corresponding to a different graph model: Erdös and Rényi (1959), Watts and Strogatz (1998), and Barabasi and Albert (1999). Notably, these graph topologies have been already exploited for testing abstract argumentation techniques (Bistarelli *et al*., 2015a, 2015b). In short, in Erdös–Rényi graphs the attacks between arguments are randomly selected according to a uniform distribution. On the other hand, Watts–Strogatz graphs are aimed at modelling many biological, technological, and social networks that are neither completely regular nor completely random. These graphs are obtained by rewiring regular networks so as to introduce increasing amounts of disorder. As a result, they can be highly clustered, like regular lattices, yet have small characteristic path lengths, like random graphs. Finally, in Barabasi–Albert graphs the node connectivities follow a scale-free power-law distribution, with the aim of modelling many large networks that expand continuously by the addition of new nodes that attach preferentially to sites that are already well connected.

[8] Inspired by well-known results from psychological investigations (Miller, 1956), experimental results in the following consider $\overline{\chi} = 7$.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Table 7 Characteristics of the considered argumentation frameworks

	Average	Minimum	Maximum
#Attacks	9766	11	231 882
#Arguments	432	11	1200
Density	0.76	0.01	4.24
Degree	43	2	609
#SCC	138	1	990
SCC size	212	1	1200

SCC = Strongly Connected Components

In particular, as Barabasi and Albert (1999) and Watts and Strogatz (1998) produce acyclic graphs—and thus a single extension (Dung, 1995)—we introduced a parameter, varying between 0 and 1, for selecting the probability of having mutual attacks instead of directed ones: if the parameter is 0, then the produced graph is acyclic; if it is 1, each attack is mutual. The number of arguments vary between 11 and 1200, and the number of attacks range between 10 and (approximately) 230 000: further details are shown in Table 7. To give an intuition of 'how difficult' those AFs are to solve, the current state-of-the-art approaches, w.r.t. the last competition—not restricted to the approaches we considered in previous sections—managed to solve 72% of them before the cutoff time of 900 seconds.

In order to generate our initial data set, a pre-processing step has then been applied to the obtained AFs. For each AF we (1) extracted its features; (2) computed the number of preferred extensions; and (3) in the case of a single preferred extension, we checked whether it was empty or not. We used various approaches, from ICCMA 2015 (Thimm *et al*., 2016), to compute the preferred extensions to maximize the chances of obtaining a solution.

Similarly to the process described in Section 5, we computed various classification and regression models included in WEKA (Hall *et al*., 2009) and assessed their performance in predicting the solution to the non-emptiness problem, and the above mentioned properties of the solution to the enumeration problem for preferred semantics. We considered linear regression, neural networks, Gaussian processes, decision trees, and rule-based techniques. As in Section 5, models have been evaluated using a 10-fold cross-validation approach on a uniform random permutation of our instances.

On AFs considered in this experimental analysis, the overall features extraction process takes around 1.5 CPU-time seconds. As it is apparent, AFs considered in this empirical analysis tends to be smaller, in terms of number of arguments, than those considered in Section 5. Again, this is due to the different focus of the investigation, and to the different final use of the trained EPMs. It is worth reminding that features will be used for estimating the number of preferred extensions, and no solver will be run afterwards: it is therefore reasonable to invest CPU-time in extracting features, in order to maximize the potential accuracy of predictions.

6.2 Hierarchical model design

We observed that the direct use of regression models for predicting the actual number of preferred extensions achieve a too low accuracy to make them useful. As a consequence, we decided to design a hierarchical predictive model organized in two levels. First, the model predicts if the given AF has an empty (and thus unique) preferred extension, a unique non-empty preferred extension, or more than one preferred extension (and thus all extensions being non-empty). In case the presence of many preferred extensions is predicted, a different classification model is used for determining whether or not the number of preferred extensions is greater than $\overline{\chi}$. Moreover, in order to investigate the generality of this approach, at the second level of the hierarchy we consider also a regression model that will output the number of extensions. Figure 3 depicts the two-level system.

The exploited hierarchical approach is appropriate because the levels evaluate different aspects of frameworks: the first focusses on the non-emptiness problem, while the second one counts the number of extensions. Intuitively, when exploiting a hierarchical approach, models of higher levels should have very good

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

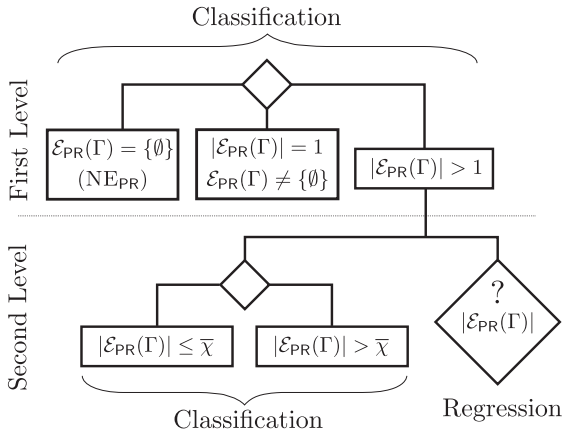


Figure 3 The two-level classifier of a argumentation framework (AF)Γ, according to the number/type of preferred extensions. The first level is said to be higher in the hierarchy than the second level

Table 8 Cross-validation of the classification model: accuracy and precision w.r.t. each considered class (percentages, higher is better)

	Groups of feature				
	DG	UG	*Graph*	Matrix	*All*
Level 1					
Accuracy	90.2	84.6	90.1	91.1	**91.4**
Precision ($\mathcal{E}_{\mathrm{PR}}(\Gamma) = \{\emptyset\}$)	**93.7**	91.0	**93.7**	**93.7**	93.5
Precision ($\lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert = 1, \mathcal{E}_{\mathrm{PR}}(\Gamma) \neq \{\emptyset\}$)	79.4	63.3	79.4	84.9	**85.6**
Precision ($\lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert > 1$)	90.7	87.3	90.7	90.5	**91.2**
Level 2					
Accuracy	85.4	64.9	85.3	**86.7**	86.3
Precision ($\lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert \leq \overline{\chi}$)	89.1	71.5	88.9	**90.0**	**90.0**
Precision ($\lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert > \overline{\chi}$)	78.3	47.5	**85.3**	80.0	79.2

Graph is the union of direct graph (DG) and undirect graph (UG) features.
Values in bold indicate the best results, also considering hidden decimals.

performance; otherwise subsequent levels will be negatively affected by the poorly classified elements. Also, it has been observed that it is possible to achieve better prediction accuracy and simpler models by using hierarchical approaches (Xu *et al*., 2008). Our models use the random forests algorithm for the classification tasks, and the M5-rules (Holmes *et al*., 1999) for regressions. These approaches showed best performance.

We trained both levels by considering the AFs in which at least a solver succeeded. In total, we considered 8640 AFs: about 4000 with empty preferred extensions, 1700 with a single non-empty preferred extension, while the others have more than one preferred extension. The training set for the second level consists of about 2800 AFs with more than one preferred extension. Since the number of preferred extensions is on average 832.1 with a variance of 2310.9, we trained our regression models to predict log-number of extensions (i.e. $\log_{10} |\mathcal{E}_{\mathrm{PR}}(\Gamma)|$) rather than absolute number (i.e. $|\mathcal{E}_{\mathrm{PR}}(\Gamma)|$): this has been effective in similar circumstances (Hutter *et al*., 2014).

6.3 Cross-validated results

Table 8 shows the results of the two-level EPMs in terms of accuracy—proportion of instances correctly classified—and precision—the proportion of true positives within a class.

As to the first level, the *All* set of features (including all the considered features) provide the best model. According to the results shown, AFs which have a single non-empty preferred extension are the

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Table 9 Root mean square error (RMSE) of $\log_{10} |\mathcal{E}_\sigma(\Gamma)|$ of the cross-validated regression model

	Groups of feature				
	DG	UG	*Graph*	Matrix	*All*
RMSE	**1.18**	2.45	**1.18**	1.24	1.81

Graph is the union of direct graph (DG) and undirect graph (UG) features.
Values in bold indicate the best results, also considering hidden decimals.

hardest to identify. It should be noted that the UG set of features, which correspond to information extracted from the UG representation of AFs, usually provide the worst predictive performance. Also, we observed that by removing the UG features from the *All* set, accuracy performance increases to 91.7%. UG features act like noise for the classification model. As expected from the principle-based analysis of argumentation semantics in Baroni and Giacomin (2007), where the directionality principle is shown to characterize preferred semantics, information regarding the direction of the attacks is important for predicting the structure of preferred extensions.

As to the second level classification, Matrix features reveal to be quite informative for accuracy and also precision in the class $|\mathcal{E}_{\mathrm{PR}}(\Gamma)| \leq \overline{\chi}$ (together with *All*). However, it is worth mentioning that the precision for the class $|\mathcal{E}_{\mathrm{PR}}(\Gamma)| > \overline{\chi}$ is slightly lower (85.3%), thus suggesting that the model is less accurate in classifying those AFs. Reasons for that are twofold. On one hand, while graph structures are quite informative at level 1 (i.e. to discriminate between only one empty extension, only one non-empty extension, or more than one extension), they are slightly less accurate in distinguishing between $\overline{\chi}$ or $\overline{\chi} - 1$ or $\overline{\chi} + 1$ extensions. In addition, the test set is moderately unbalanced (60% of AFs has less than $\overline{\chi}$ extensions), but given the good results (accuracy and precision between 85% and 90%), we chose not to apply techniques of oversampling to overcome this.

Table 9 shows the performance, in terms of RMSE, of regression models predicting the log-number of preferred extensions (i.e. $\log_{10} |\mathcal{E}_{\mathrm{PR}}(\Gamma)|$). In this case, UG features are significantly worse than others. On the other hand, either using only Matrix-related or DG features allow to achieve better performance than the *All* set. This is probably due to the noise added by considering the UG set.

6.4 Most important features

The importance of features has been assessed using the CFS attribute evaluator, as described in Section 5. The most important features selected are:

- Classification, Level 1: number of SCCs (DG); average degree (DG); aperiodicity (DG).
- Classification, Level 2: ratio vertice-edges (DG); standard deviation degree (DG); size of maximal SCC (DG); flow hierarchy (DG); average lines sum (Matrix); norm lines sum (Matrix).
- Regression: standard deviation degree (DG); aperiodicity (DG); density triangles (UG); average triangles (UG); averages difference columns (Matrix).

As to the first level, by using the three obtained features it is possible to build a model with a 91.0% of accuracy. This indicates that a small set of features is extremely informative, and allows to achieve very good predictive performance. The remaining DG and Matrix performance are probably helpful in complex cases only.

As to the classification task at the second level, with the six identified features it is possible to build a model with 85.7% of accuracy, and with a great precision in the case of $|\mathcal{E}_{\mathrm{PR}}(\Gamma)| \leq \overline{\chi}$, viz. 89.1%, although not exceptional for the other case (78.9%).

By using the five features the CFS techniques selects for the regression task, it is possible to generate regression models with a RMSE of 2.36. This is better than using UG features only (cf. Table 8), but worse than any other set. Interestingly, CFS included features from each of the three considered sets. Intuitively, few of the UG features are informative with regards to the regression task.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

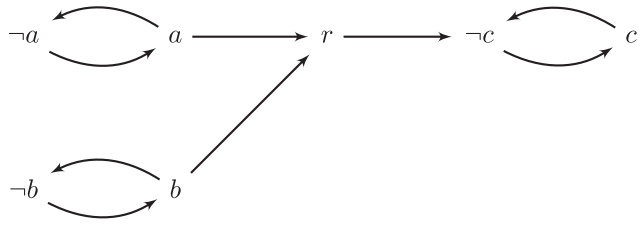


Figure 4 Argumentation framework (AF) derived from *r*: ¬a, ¬b → c

6.5 Validation of models against argumentation frameworks with identifiable clusters of information

Frameworks generated using the AFBenchGen tool do not carry any sort of information: they are random sets of arguments organized according to a predefined structure. However, as demonstrated in other areas of AI such as SAT, instances that do carry information lead to results that are very different from those observed on randomly generated instances. For instance, in the case of SAT, information carried by Conjunctive Normal Forms (CNFs) usually relates to some manufacturing processes, and the relevant impact needs to be verified. To prove the generality of our approach w.r.t. the *Argument Generation* process—thus guaranteeing re-usability of our results in contexts not linked to our investigation—here we validate the proposed framework on AFs with *identifiable clusters of information*. In the following, their generation process, starting from automated planning problems, is detailed.

6.5.1 Deriving argumentation frameworks with identifiable clusters of information

Given a planning domain model and a problem, we:

1. encode the planning problem in SAT formulae (Sideris & Dimopoulos, 2010) in CNF: six different encodings have been considered, for a total of 500 different formulae;
2. transform each disjunction in a material implication: for example, $a \vee b \vee c$ becomes $\neg a \wedge \neg b \supset c$;
3. evaluate each material implication as a named strict rule (e.g. *r*: $\neg a$, $\neg b \rightarrow c$) and apply the transformation described in Wyner *et al*. (2015) (see Figure 4)[9].

The advantage of using planning domain models relies on the small number of operators and objects, which limits the size of resulting SAT formulae. We did not consider SAT instances from SAT competitions due to their size and complexity: in total, the considered solvers were able to analyze less than 10 instances. At the same time, we did not include the AFs proposed by Cabrio and Villata (2014) because there are very few of them.

We chose to use the approach presented by Wyner *et al*. (2015) because there is a one-to-one syntactic correspondence between the SAT variables and the generated AF. Dealing with the entire model of argumentation, and thus connecting the two processes of argument generation, using for instance ASPIC+ from a propositional knowledge base, and of semantic evaluation of the proposition in the knowledge base, is already envisaged as the main future work.

6.5.2 Performance of empirical predictive models on argumentation frameworks with identifiable clusters of information

To evaluate the performance of the trained models we derived 500 AFs following the above procedure and using two well-known benchmark planning domains: Blocksworld and Ferry. The former deals with controlling a robot arm for re-assembling stackable blocks, the latter models the use of a ferry boat for moving cars between different islands. Out of the 500 considered AFs with identifiable clusters of information, the solvers successfully analyzed 200 of them, which have been included in our evaluation. The number of arguments range from 86 to ~2000, the number of attacks is between 136 and 4255.

[9] An interested reader can find in Wyner *et al*. (2015) a discussion on the relations between preferred extensions and models of the original logical formulae.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Table 10 Classification model evaluated against argumentation frameworks with identifiable clusters of information: accuracy and precision w.r.t. each considered class (percentages, higher is better)

	Groups of feature				
	DG	UG	*Graph*	Matrix	*All*
Level 1					
Accuracy	**100.0**	92.4	**100.0**	**100.0**	**100.0**
Level 2					
Accuracy	**94.4**	56.3	**94.4**	78.8	86.5
Precision ($\lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert \leq \overline{\chi}$)	**33.3**	0.0	**33.3**	0.0	0.0
Precision ($\lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert > \overline{\chi}$)	**96.8**	92.8	**96.8**	94.7	95.2

Graph is the union of direct graph (DG) and undirect graph (UG) features.
Values in bold indicate the best results, also considering hidden decimals.

Table 11 Root mean square error (RMSE) of $\log_{10}\lvert\mathcal{E}_{\sigma}(\Gamma)\rvert$ of the regression model evaluated against argumentation frameworks with identifiable clusters of information

	Groups of feature				
	DG	UG	*Graph*	Matrix	*All*
RMSE	12.9	**1.3**	5.8	15.5	38.0

Graph is the union of direct graph (DG) and undirect graph (UG) features.
Values in bold indicate the best results, also considering hidden decimals.

Also in this case the number of extensions has a great variance (1989.8) across the considered testing set[10]. We should remark that successfully analyzed AFs provide the ground truth for the correct evaluation of models.

Table 10 shows the results of classification and regression models against AFs with identifiable clusters of information. It clearly indicates that the classification model is very accurate also on AFs that are significantly different from the training ones. On the other hand, it is worth noting that the considered AFs derived from SAT formulae always have more than one preferred extension. Thus, they are all from the same class, according to the exploited classification model. For this reason, we are not showing precision results. Nevertheless, the predictive model is able to correctly identify all of them. Remarkably, the trend shown in Table 8 is confirmed by results in Table 10: features derived from the UG are not as informative as those from the other sets for the classification task. Also, a model based on the three features selected by using the CFS technique only, is able to correctly classify 100% of AFs.

Although the set of AFs with identifiable clusters of information is very unbalanced—only 4.0% of the frameworks has less than $\overline{\chi}$ extensions—the classification models perform very well, with an accuracy of 94.4% and an average recall of 96% from the features derived from the DG, which appears to be dominating also in the set of *Graph* as the results are identical. Considering only the six features selected by the greedy forward search on the CFS attribute selector, the accuracy is still quite high, 91.9%. Results presented in Table 10 also highlight the hardness of correctly identifying the (very few) frameworks that has less than $\overline{\chi}$ extensions: UG, Matrix, and *All* sets of features show a 0.0 precision value, which indicates they are not able to correctly identify any of such frameworks. This possibly indicates that the structure of frameworks from this specific class is very different from the structure of those, from the same class, included in the training set.

Table 11 shows the performance of the regression model aimed at predicting the log-number of preferred extensions (i.e. $\log_{10}\ \lvert\mathcal{E}_{\mathrm{PR}}(\Gamma)\rvert$). Results are significantly different from those observed in Table 8:

[10] Average number of extensions: 1024.2; variance: 1989.8.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

the *All* set of features seems to include a lot of noise, and does not allow to make any prediction. Given the results on the Matrix and DG sets, it is reasonable to derive that noise is mainly from these two sets.

6.6 Discussion

In summary, according to the results in Tables 8 and 10, the proposed two-level approach shows good overall performance. This demonstrates that the predictive hierarchical model built on the considered training AFs (i.e. those described in Section 6.1) is able to address a testing set including differently structured frameworks, although the relative importance of the considered set of features changes, as will be shown below.

Interestingly, the classification models have generally better performance on this testing set than on the training one. This is probably due to the different structure of the AFs in training and testing sets. Moreover, the informativeness of the sets of features is similar in both the considered sets of instances for the first level of the hierarchical model, but not for the second level classifier, where it is evident that the features from the DG representation are much more informative for the testing (structured) AFs (Table 10).

As to the regression task, features extracted by considering the UG are very informative, and lead to accurate regression model. The regression model relying on the five features selected by the CFS method shows a RMSE of 2.9. This is better than the performance achieved by considering the *All* set of features, but slightly worse than those obtained by exploiting the UG features only. For confirming the importance of UG features, we applied the CFS technique directly on the testing set. Four features have been selected, two from the UG set, and two from the matrix encoding. Even though this selection might be affected by overfitting, given the number of considered AFs, we believe that the importance of the UG features for the regression task on testing frameworks is confirmed. Intuitively, in the regression task structured AFs have a topology which can effectively be synthesized by considering the UG representation only. Moreover, information added by considering the DG, or the matrix encoding, are misleading with regards to the task of predicting the number of preferred extensions.

7 Conclusions

This work provides an extensive investigation of the exploitability of automatically generated predictions in dealing with high-complexity problems. In order to perform such investigation, we introduced the largest set of features for AFs ever proposed in literature which analyses them both from a graph and matrix perspective. Results presented in Section 5 demonstrate that EPMs can be used to determine the 'best' algorithm—relatively to the CPU-time—with an accuracy of about 80%. Moreover, via an empirical investigation, we derive an algorithm selection approach, based on classification, which can identify the fastest algorithm in about the double of the number of cases where the most efficient algorithm outperforms the other ones (**SCC-P**), and about three times the number of cases of the second most efficient algorithm (**ArgSemSAT**). It is worth mentioning that the obtained results are consistent with previous empirical investigations (Cerutti *et al*., 2013). For instance, according to Table 5, **ArgSemSAT** seems to be affected by the number of vertices (feature B1) much more than **Aspartix** and **ArgTools**. In Cerutti *et al*. (2013) we show that both **Aspartix** and **ArgTools** show behaviours independent from the cardinality of the set of arguments, while they depend on the percentage of attacks.

Results discussed in Section 6 confirms that we develop a model able to predict, with an overall accuracy of 91.4%, whether or not an AF has a unique empty preferred extension, a problem which is known to be NP-COMPLETE (Dunne & Wooldridge, 2009). Moreover, inspired by the work of Baumann and Strass (2014), where an approach for computing the maximum and average number of stable extensions is provided, using a hierarchical predictive model we can distinguish when there is one or more preferred extensions. Finally, if more than one preferred extension is predicted, predictive models are able to predict their log-number. Furthermore, we show how it is possible to discriminate around a pivotal number of extension with an accuracy $\geq$90.0% both using our randomly generated training set as well as the AFs with identifiable clusters of information.

This work opens the road to a large set of interesting future work and applications of the developed techniques. Future work can be envisaged both from a theoretical point of view—that is, determining

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

potential inter-dependencies between set of features and semantics for the tasks identified in this paper—as well as from an 'engineering' perspective. A natural extension to the proposed algorithm selection investigation would be the analysis of *portfolios*: portfolio approaches are more complex than algorithm selection ones, since they have to (i) select a subset of solvers; (ii) order the solvers and; (iii) allocating CPU-time to each solver. Such increased complexity can potentially lead to further performance improvements: see, for instance, Cerutti *et al*. (2018). We also envisage an analysis of the exploitability of predictive models for supporting different, and notably less computationally complex, reasoning problems. We are also interested in extending the set of considered features as well as of benchmarks, to include those of the recent International Competitions on Computational Models of Argumentation (Thimm *et al*., 2016; Thimm & Villata, 2017). From a solvers' perspective, it is important to assess the usefulness and impact of *probing* features: features that are computed by briefly running an existing algorithm on the given AF and extracting characteristics from that algorithm's trajectory (Hutter *et al*., 2014). Also, different aspects of the structure of AFs can be considered for extracting more features, such as: information related to symmetric attacks, treewidth (Dunne, 2007; Dvořák *et al*., 2012), acyclicity. Moreover, Dunne *et al*. (2015) provides characterization theorems for *realizing* AFs given a set of extensions under different semantics, including preferred semantics. (Dunne *et al*., 2015, Proposition 10) indeed shows that if two (maximal) sets of arguments are involved, preferred semantics that satisfies incomparability of extensions is capable to deliver such extensions. Linking the study of features with realizability of AFs is an existing interesting line of future work.

Finally, we will further investigate regressive approaches in order to predict the number of extensions: an investigation in the context of stable semantics in order to provide a comparison with Baumann and Strass (2014) is already envisaged, as well as for other semantics.

Among the applications mentioned in Section 1, we point out that the investigated predictive models can be exploited in existing argumentation-based decision support systems (Tamani *et al*., 2015; Toniolo *et al*., 2015). For instance, CISpaces (Toniolo *et al*., 2015) is an argumentation-based research-grade prototype for supporting intelligence analysts in their sense-making process, now under consideration for transitioning into a commercial product by the US Army Research Laboratory. It supports intelligence analysts in sense-making via compelling hypotheses, where each hypothesis is a preferred extension. Being able to identify the best algorithm to compute preferred extensions is clearly an important feature, especially when the probabilistic engine of CISpaces is enabled, which requires to solve an argumentation problem—for example, the enumeration of preferred extensions—several times evaluating different probabilistic worlds. At the same time, if each preferred extension is an hypothesis that an analyst needs to assess, the more the extensions the more the cognitive burden on the analysts. In this respect, it becomes important to know in advance the number of extensions, and in particular if such number exceeds a given threshold. Moreover, this information can be exploited in environments where the structure of frameworks change frequently, and operators need to quickly obtain an overview of the structure of the solution, for example, in order to know in advance what to change in the knowledge base to reduce the expected number of extensions. We believe that it would be of extreme interest to evaluate how analysts will leverage the proposed tools.

Acknowledgement

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work. The authors also thank the anonymous reviewers for their helpful comments.

References

Audemard, G. & Simon, L. 2014. Lazy clause exchange policy for parallel sat solvers. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 197–205. Springer.

Barabasi, A. & Albert, R. 1999. Emergence of scaling in random networks. *Science* **286**(5439), 509–512.

Baroni, P., Caminada, M. & Giacomin, M. 2011. An introduction to argumentation semantics. *Knowledge Engineering Review* **26**(4), 365–410.

Baroni, P., Cerutti, F., Dunne, P. & Giacomin, M. 2013. Automata for infinite argumentation structures. *Artificial Intelligence* **203**, 104–150.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Baroni, P., Dunne, P. E. & Giacomin, M. 2010. On extension counting problems in argumentation frameworks. In *Proceedings of the 3rd International Conference on Computational Models of Argument (COMMA)*, **216**, 63–74. IOS Press.

Baroni, P. & Giacomin, M. 2007. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence (Special issue on Argumentation in A.I.)* **171**(10/15), 675–700.

Baroni, P., Giacomin, M. & Guida, G. 2005. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence* **168**(1–2), 165–210.

Baumann, R. & Strass, H. 2014. On the maximal and average numbers of stable extensions. In *Proceedings of the 2nd International Workshop on Theory and Applications of Formal Argumentation (TAFA)*, 111–126. Springer.

Besnard, P. & Hunter, A. 2014. Constructing argument graphs with deductive arguments: a tutorial. *Argument & Computation* **5**(1), 5–30.

Bistarelli, S., Rossi, F. & Santini, F. 2015a. A comparative test on the enumeration of extensions in abstract argumentation. *Fundamenta Informaticae* **140**(3–4), 263–278.

Bistarelli, S., Rossi, F. & Santini, F. 2015b. Testing credulous and sceptical acceptance in smallworld networks. In *Proceedings of the 22nd RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA)*, 39–46.

Brewer, E. A. 1994. *Portable High-Performance Supercomputing: High-Level Platform-Dependent OptiMization*. PhD thesis, MIT.

Cabrio, E. & Villata, S. 2014. Node: A benchmark of natural language arguments. In *Proceedings of the Fifth International Conference on Computational Models of Argument (COMMA)*, 449–450. IOS Press.

Cerutti, F., Dunne, P., Giacomin, M. & Vallati, M. 2013. Computing preferred extensions in abstract argumentation: a SAT-based approach. In *Proceedings of the 2nd International Workshop on Theory and Applications of Formal Argumentation (TAFA)*, 176–193. Springer.

Cerutti, F., Giacomin, M. & Vallati, M. 2014a. Algorithm selection for preferred extensions enumeration. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA)*, 221–232. IOS Press.

Cerutti, F., Giacomin, M. & Vallati, M. 2014b. Generating challenging benchmark AFs. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA)*, 457–458. IOS Press.

Cerutti, F., Giacomin, M., Vallati, M. & Zanella, M. 2014c. A SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press.

Cerutti, F., Vallati, M. & Giacomin, M. 2017. An efficient Java-based solver for abstract argumentation frameworks: jArgSemSAT. *International Journal on Artificial Intelligence Tools* **26**(2), 1750002.

Cerutti, F., Vallati, M. & Giacomin, M. 2018. On the impact of configuration on abstract argumentation automated reasoning. *International Journal of Approximate Reasoning* **92**, 120–138.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence* **77**(2), 321–357.

Dunne, P. & Wooldridge, M. 2009. Complexity of abstract argumentation. In *Argumentation in Artificial Intelligence*, 85–104. Springer.

Dunne, P. E. 2007. Computational properties of argument systems satisfying graph-theoretic constraints. *Artificial Intelligence* **171**(10–15), 701–729.

Dunne, P. E., Dvořák, W., Linsbichler, T. & Woltran, S. 2015. Characteristics of multiple viewpoints in abstract argumentation. *Artificial Intelligence* **228**, 153–178.

Dvořák, W., Gaggl, S., Wallner, J. & Woltran, S. 2011. 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)*, 114–133. Springer.

Dvorák, W., Järvisalo, M., Wallner, J. P. & Woltran, S. 2015. Cegartix v0. 4: A SAT-based counterexample guided argumentation reasoning tool. In *System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA15)*, 12.

Dvořák, W., Pichler, R. & Woltran, S 2012. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence* **186**, 1–37.

Erdös, P. & Rényi, A. 1959. On random graphs. *I. Publicationes Mathematicae Debrecen* **6**, 290–297.

Fawcett, C., Vallati, M., Hutter, F., Hoffmann, J., Hoos, H. & Leyton-Brown, K. 2014. Improved features for runtime prediction of domain-independent planners. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 355–359. AAAI Press.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M. T. & Ziller, S. 2011. A portfolio solver for answer set programming: preliminary report. In *Proceedings of the 11th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 352–357. Springer.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Gebser, M., Kaufmann, B., Neumann, A. & Schaub, T. 2007. clasp: A conict-driven answer set solver. In *Proceedings of the 9th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 260–265. Springer.

Gomes, C. P., Sabharwal, A. & Selman, B. 2009. Model counting. In *Handbook of Satisfiability*, Biere, A., Heule, M., van Maaren, H. & Walsh, T. (eds). 633–654. IOS Press.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. 2009. The WEKA data mining software: an update. *SIGKDD Explorations* **11**(1), 10–18.

Hall, M. A. 1998. *Correlation-Based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Department of Computer Science, Hamilton, New Zealand.

Holmes, G., Hall, M. & Prank, E. 1999. *Generating Rule Sets from Model Trees*. Springer.

Hoos, H., Lindauer, M. T. & Schaub, T. 2014. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming* **14**(4–5), 569–585.

Hutter, F., Xu, L., Hoos, H. & Leyton-Brown, K. 2014. Algorithm runtime prediction: methods & evaluation. *Artificial Intelligence* **206**, 79–111.

Kohavi, R. 1995. The power of decision tables. In *8th European Conference on Machine Learning*, 174–189. Springer.

Kröll, M., Pichler, R. & Woltran, S 2017. On the complexity of enumerating the extensions of abstract argumentation frameworks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, 1145–1152.

Leyton-Brown, K., Nudelman, E. & Shoham, Y. 2009. Empirical hardness models: methodology and a case study on combinatorial auctions. *Journal of the ACM* **56**(4), 1–52.

Luo, J. & Magee, C. L. 2011. Detecting evolving patterns of self-organizing networks by flow hierarchy measurement. *Complexity* **16**(6), 53–61.

Maratea, M., Pulina, L. & Ricca, F. 2014. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming* **14**(6), 841–868.

Marquardt, D. W. & Snee, D. 1975. Ridge regression in practice. *The American Statistician* **29**(1), 3–20.

Matos, P., Planes, J., Letombe, F. & Marques-Silva, J. 2008. A MAX-SAT algorithm portfolio. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, 911–912. IOS Press.

Miller, G. A. 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review* **63**, 81.

Nofal, S., Atkinson, K. & Dunne, P. 2014. Algorithms for decision problems in argument systems under preferred semantics. *Artificial Intelligence* **207**, 23–51.

Nudelman, E., Leyton-Brown, K., Devkar, A., Shoham, Y. & Hoos, H. 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP)*, 438–452. Springer.

Pulina, L. & Tacchella, A. 2007. A multi-engine solver for quantified Boolean formulas. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, 574–589. Springer.

Sideris, A. & Dimopoulos, Y. 2010. Constraint propagation in propositional planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 153–160. AAAI Press.

Smith-Miles, K., van Hemert, J. & Lim, X. 2010. Understanding TSP difficulty by learning from evolved instances. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization (LION)*, 266–280. Springer.

Tamani, N., Mosse, P., Croitoru, M., Buche, P., Guillard, V., Guillaume, C. & Gontard, N. 2015. An argumentation system for eco-efficient packaging material selection. *Computers and Electronics in Agriculture* **113**, 174–192.

Thimm, M. & Villata, S. 2017. The first international competition on computational models of argumentation: results and analysis. *Artificial Intelligence* **252**, 267–294.

Thimm, M., Villata, S., Cerutti, F., Oren, N., Strass, H. & Vallati, M. 2016. Summary report of the first international competition on computational models of argumentation. *AI Magazine* **37**(1), 102.

Toniolo, A., Norman, T. J., Etuk, A., Cerutti, F., Ouyang, R. W., Srivastava, M., Oren, N., Dropps, T., Allen, J. A. & Sullivan, P. 2015. Agent support to reasoning with different types of evidence in intelligence analysis. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 781–789. ACM.

Valiant, L. 1979. The complexity of computing the permanent. *Theoretical Computer Science* **8**(2), 189–201.

Vallati, M., Cerutti, F. & Giacomin, M. 2014. Argumentation frameworks features: an initial study. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 1117–1118. IOS Press.

Watts, D. J. & Strogatz, S. H. 1998. Collective dynamics of 'small-world' networks. *Nature* **393**(6684), 440–442.

Wyner, A., Bench-Capon, T., Dunne, P. & Cerutti, F. 2015. Senses of argument in instantiated argumentation frameworks. *Argument & Computation* **6**(1), 50–72.

Xu, L., Hutter, F., Hoos, H. & Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* **32**, 565–606.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

Appendix A: Matrix-related Features

Any AF with N arguments can be encoded as an $N \times N$ matrix. Given this encoding we are able to extract 98 features, divided in:

- *Basic matrix* (9):

1. Rank.
2. Sparsity of the matrix.
3. Number of non-zero elements.
4. Standard deviation of the matrix elements.
5. Average of the matrix elements.
6. Variance of the matrix elements.
7. Norm.
8. Determinant.
9. Sum of matrix diagonals.

- *Sum* the following statistics are computed on vectors resulting from sum of **either** columns or rows (14):

1. Standard deviation.
2. Minimum value.
3. Maximum value.
4. Average.
5. Variance.
6. Number of non-zero elements.
7. Norm.

- *Cumulative sum* the following statistics are computed on vectors resulting from cumulative sum of columns, rows, and diagonals (21):

1. Standard deviation.
2. Minimum value.
3. Maximum value.
4. Average.
5. Variance.
6. Number of non-zero elements.
7. Norm.

- *Difference* the following statistics are computed on vectors resulting from differences of cells on columns, rows, and diagonals (21):

1. Standard deviation.
2. Minimum value.
3. Maximum value.
4. Average.
5. Variance.
6. Number of non-zero elements.
7. Norm.

- *Diagonals* the following statistics are extracted by considering the first and second diagonal of the matrix (7):

1. Standard deviation.
2. Minimum value.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press

3. Maximum value.
4. Average.
5. Variance.
6. Number of non-zero elements.
7. Norm.

- *Gradient* the following statistics are extracted by considering the gradient of the vector resulting by (i) the sum of the rows, (ii) the sum of the columns, and (iii) the sum of the diagonals of the matrix (21):

1. Standard deviation.
2. Minimum value.
3. Maximum value.
4. Average.
5. Variance.
6. Number of non-zero elements.
7. Norm.

- Correlation the correlation value of the following couple of vectors (4):

1. Sum of columns, sum of rows.
2. Sum of columns, first diagonal.
3. Sum of rows, first diagonal.
4. gradient of sum of columns, gradient of sum of rows.

https://doi.org/10.1017/S0269888918000036 Published online by Cambridge University Press