Algorithm Selection for Preferred Extensions Enumeration

Federico CERUTTI [a,1], Massimiliano GIACOMIN [b] and Mauro VALLATI [c]
[a] *Department of Computing Science, King's College, University of Aberdeen, UK*
[b] *Department of Information Engineering, University of Brescia, Italy*
[c] *School of Computing and Engineering, University of Huddersfield, UK*

Abstract. Enumerating semantics extensions in abstract argumentation is generally an intractable problem. For preferred semantics four algorithms have been recently proposed, **AspartixM**, **NAD-Alg**, **PrefSAT** and **SCC-P**, with significant runtime variations. This work is a first comprehensive exploration of the graph features and of their impact on the execution time of state-of-the-art preferred extensions enumeration algorithms. Following other areas of AI, we exploit *empirical performance models*, predictive models that relate instance features and algorithms performance. The result is an approach able to select the "best" algorithm for any Dung's argumentation framework with an accuracy, on the average, of the 80%. Moreover, we show that an algorithm selection approach based on classification can select 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 (**PrefSAT**).

Keywords. argumentation semantics, argumentation features, algorithm selection

1. Introduction

Dung's theory of abstract argumentation frameworks [1] 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. 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*, whose definitions and properties are an active investigation subject since two decades [2, 3].

In [1] four "traditional" semantics were introduced, namely *complete*, *grounded*, *stable*, and *preferred* semantics. Associated to each semantics, there are several computational problems and they include *decision* and *construction* problems, which turn out to be computationally intractable for most of argumentation semantics [4]. In this paper we focus on the *extension enumeration* problem, i.e. constructing *all* extensions prescribed for a given *AF*: its solution provides complete information concerning the justification

[1]Corresponding Author.

status of arguments and subsumes the solutions to the other problems. In particular, we consider preferred semantics, which represents the main contribution in Dung's theory, 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 enumeration problem associated to this semantics is at the second level of the polynomial hierarchy [4]: this justifies the search for efficient mechanisms for solving it.

Among others, four *solvers* for the preferred extensions enumeration have been recently proposed: **AspartixM** [5], **NAD-Alg** [6], **PrefSAT** [7] and **SCC-P** [8]. Previous empirical analyses have shown significant runtime variations between them, arising even among *AF*s with the same number of arguments [7]. Moreover, in some cases different solvers require dramatically different amount of CPU-times given the same *AF*. In short, according to our exploration, there is not a "gold" solver among the four considered.

The aim of this paper is to provide an approach to predict how a given solver will perform on a given *AF*, and thus automatically select the "best solver". Such predictions are possible using so-called *empirical performance models* (EPMs) [9] — successfully applied for instance in SAT, MIP, TSP and Automated Planning [10, 11, 12, 13, 14, 15] — which require the following steps to be constructed. First, each solver is run on a large number of *AF*s and, for each *AF*, each solver's performance (e.g. the CPU-runtime) is recorded. Furthermore, a set of instance features is computed for each *AF*: an instance feature summarises a property of the *AF*. A predictive model is then learnt as a mapping from instance features to solvers performance.

This work establishes an extensive set of features for Argumentation Frameworks, and investigates its effectiveness in predicting algorithm performance and performing an efficient algorithm selection. Recently, we discussed a preliminary restricted set of features [16] considering only **PrefSAT** and **SCC-P**. In this paper we significantly extend both the set of features, and the set of solvers, and we also discuss the implication related to the algorithm selection. The importance of this contribution is twofold. From a theoretical point of view, it provides some explanation of the features causing the runtime variations between different approaches, an issue which is still unexplained. From a practical perspective, the methodology proposed in the paper can be extended to include any further solver, and allows to combine the solvers so as to improve the performance of each of them used in isolation.

The paper is organized as follows. In Section 2 we review Dung's abstract framework theory, and we briefly describe the functioning of each of the four solvers mentioned above. Section 3 discusses the chosen features for *AF*s, and Section 4 presents the algorithm-selection experimental results given the chosen set of features. Section 5 concludes the paper.

2. Background

2.1. Dung's Argumentation Framework

An argumentation framework [1] consists of a set of arguments and a binary attack relation between them.[2]

[2] In this paper we consider only *finite* sets of arguments: see [17] for a discussion on infinite sets of arguments.

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

An argument $\mathbf{a}$ without attackers, i.e. such that $\mathbf{a}^- = \emptyset$, is said *initial*. The *neighbour* of an argument $\mathbf{a}$ is $\mathbf{a}^- \cup \mathbf{a}^+$. Moreover, each argumentation framework has an associated 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 $\Gamma = \langle \mathscr{A}, \mathscr{R} \rangle$:*

- *a set $S \subseteq \mathscr{A}$ is a* conflict–free *set of Γ if $\nexists\, \boldsymbol{a}, \boldsymbol{b} \in S$ s.t. $\boldsymbol{a} \to \boldsymbol{b}$;*
- *an argument $\boldsymbol{a} \in \mathscr{A}$ is* acceptable *with respect to a set $S \subseteq \mathscr{A}$ of Γ if $\forall \boldsymbol{b} \in \mathscr{A}$ s.t. $\boldsymbol{b} \to \boldsymbol{a}$, $\exists\, \boldsymbol{c} \in S$ s.t. $\boldsymbol{c} \to \boldsymbol{b}$;*
- *a set $S \subseteq \mathscr{A}$ 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 $\mathscr{E}_\sigma(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by σ. Here we need to recall the definitions of complete (denoted as $\mathscr{CO}$), and preferred (denoted as $\mathscr{PR}$) semantics only.

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

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

2.2. Algorithms for Solving the Preferred Extensions Enumeration Problem

In this paper we consider four state-of-the-art approaches for solving the problem of preferred extensions enumeration: **AspartixM**, **NAD-Alg**, **PrefSAT**, and **SCC-P**.

AspartixM [5] 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`.[3]

NAD-Alg [6] 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.[4]

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

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

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 used 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 states are excluded from further exploration by adding specific constraints to the encoding of complete extensions.

SCC-P is an instantiation of the meta-algorithm proposed in [8] which exploits the SCC-recursiveness schema [18]: it recursively decomposes a framework so as to compute semantics extensions on restricted sub-frameworks, in order to reduce the computational effort. At the beginning, the extensions of the frameworks restricted to the initial Strongly Connected Components (SCCs), i.e. 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 is recursively applied. For every SCC, the local computation is performed as follows. 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 in order to compute extensions in restricted sub-frameworks, e.g. a variation of **PrefSAT** as proposed in [8].

3. Features

As mentioned in Section 1, accurate EPMs rely on a "good" set of features. A feature is a real number that summarises a property of *AF*s. Our feature set includes 50 values, which exploit the representation of *AF*s both as direct (loss-less) or undirect (lossy) graphs.

We are able to extract 26 features from the representation of *AF*s as direct graphs (DG). Each feature belongs to one of the following four classes: *graph size* (5 features), *degree* (4), *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 (NT – non trivial).[5]
- *Degree features* (overall NT): average, standard deviation, maximum, minimum degree values across the nodes in the graph.
- *SCC features* (overall NT): number of SCCs, average, standard deviation, maximum and minimum size.
- *Graph structure*: presence of auto-loops, number of isolated vertices (NT), flow hierarchy (NT) and results of test on Eulerian (NT) and aperiodic structure of the graph (NT).
- *CPU-times*: the needed CPU-time for extracting NT features and overall NT classes.

[5]We consider as trivial the extraction of features that requires less than 0.001 seconds. Such features are, for instance, those requiring only elements count (e.g., number of edges) or easy calculations (e.g., ratios vertices–edges).

Direct Graph Features (DG)				**Undirect Graph Features (UG)**			
Class	**CPU-Time**		**# feat**	**Class**	**CPU-Time**		**# feat**
	Mean	stdDev			Mean	stdDev	
Graph Size	0.001	0.009	5	Graph Size	0.001	0.003	4
Degree	0.003	0.009	4	Degree	0.002	0.004	4
SCC	0.046	0.036	5	Components	0.011	0.009	5
Graph Structure	2.304	2.868	5	Graph Structure	0.799	0.684	1
				Triangles	0.787	0.671	5

Table 1. Summary of proposed features grouped by class. Mean indicates the average CPU-time needed for extracting all the features of the class, "stdDev" is the standard deviation and "# feat" the number of features of the class.

The features (UG) extracted by the undirect graph representation of *AF*s — i.e. 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), *triangles* (5), *graph structure* (1), *times* (5).

- *Graph size features*: number of edges, ratios vertices–edges and inverse, and graph density (NT).
- *Degree features* (overall NT): average, standard deviation, maximum, minimum degree values across the nodes in the graph.
- *Components features* (overall NT): number of connected components, average, standard deviation, maximum and minimum size.
- *Graph structure*: transitivity of the graph (NT).
- *Triangles features* (overall NT): 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 NT features and overall NT classes.

Table 1 summarises the list of features, and reports the average and standard deviation of the CPU-time associated with the extraction of each feature group. The overall feature extraction process takes around 3 CPU-time seconds per *AF* (apart from the aperiodicity of the graph which requires up to 20 seconds). Each *AF* is solved in tens of hundreds of seconds: therefore we can consider the feature extraction CPU-time as negligible compared to the CPU-time required for solving an *AF*.

4. Experimental Analysis

In this section we describe the protocol and present the results of a large experimental study examining **AspartixM** [5], **NAD-Alg** [6], **PrefSAT** [7] and **SCC-P** [8].

4.1. Experimental Protocol

We randomly generated a set of 10,000 *AF*s, varying the number of SCCs between 0 and 100, the number of arguments between 10 and 5,000, and considering different uniformly distributed probabilities of attacks, leading to *AF*s with a number of attacks between 35 and (approximately) 270,000.

Solver	B1	B2	B3
AspartixM	number of nodes	density of directed graph	size of max. SCC
PrefSAT	density of directed graph	number of SCCs	aperiodicity
NAD-Alg	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

Table 2. Best features selected for each solver, by the greedy approach, for predicting runtime.

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$^{\text{TM}}$, 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 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. 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.

We first assessed the performance of various classification and regression models, using the WEKA tool [19]. We considered well-known machine learning techniques: linear regression, neural networks, Gaussian processes, decision trees and rule-based techniques, and we observed that random forests performed best in classification, and M5-Rules in regression.

The EPMs have been generated considering 7 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 directed graph (DG), the set including those extracted from undirected graph (UG), and the set with all the extracted features (*All*). It turns out that B1, B2 and B3 include only features from the directed graph. The subsets of features are selected according to a greedy forward search based on the Correlation-based Feature Selection (CFS) attribute evaluator [20]. The search starts from an empty subset. At each iteration, the feature which increases the most the evalution 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. Given the results shown in [8], SCC-related information are believed to be extremely informative, thus expected to play a significant role in the EPMs performance.

4.2. Runtime Prediction

Table 3 shows the results, in terms of root mean square error (RMSE), for regression using the best performing models, 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

Regression (Lower is better)							
	B1	B2	B3	DG	UG	SCC	*All*
AspartixM	0.66	0.49	0.49	0.48	0.49	0.52	**0.48**
PrefSAT	1.39	0.93	0.93	0.89	0.92	0.94	**0.89**
NAD-Alg	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**

Table 3. Cross-validated RMSE of log_{10}(runtime) for M5-rules models using different features subsets. Values in bold indicate the best results, also considering hidden decimals.

900 CPU seconds, we trained our regression models to predict log-runtime rather than absolute runtime: this have been effective in similar circumstances [9]. The three best features selected by the greedy approach are summarised in Table 2.

This analysis provides several insights on features that affect solvers performance. First, the density of directed graph — i.e. how close it is to a complete graph — is strongly related to the performance of all the solvers since it is always among the best 3 features. Second, we observe that also SCC-related information, mainly the number of SCCs and the size of the maximum one, are often selected.

Results shown in Table 3 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 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 **AspartixM**, **SCC-P** and **PrefSAT**. Interestingly, this is not true for **NAD-Alg**. 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 **NAD-Alg**, and this is confirmed also by the selected B3 features. Finally, we noted that the behaviours of **PrefSAT** 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 **NAD-Alg** ran out of time on most of the *AF*s, and **AspartixM** solved a significant number of them in 800-899 CPU-time seconds. **PrefSAT** and **SCC-P** show different, and more variegated, CPU-time distributions.

4.3. Classification

Table 4 shows the results, in terms of overall accuracy and per-algorithm precision, of the classification EPM, evaluated with 10-fold cross validation. In this approach, an *AF* is member of the class corresponding to the solver which has been the fastest in enumerating all the preferred extensions. From the set of 10,000 *AF*s, those which were not solved by any solver were removed. 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.

The three best features selected for the classification EPM are, in the order, (1) the number of vertices, (2) the density of the directed graph and (3) the minimum degree value of the directed graph.

Several considerations can be drawn from Table 4:

- the whole set of features usually allows the EPM to achieve the best predicting performance;

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 **AspartixM**	35.0%	64.6%	63.7%	74.5%	74.9%	42.2%	**76.1%**
Precision **PrefSAT**	53.7%	67.8%	68.1%	79.6%	**80.5%**	60.4%	80.1%
Precision **NAD-Alg**	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%**

Table 4. Evaluation of classification EPMs for different features subsets. A precision value smaller than 50% indicates that most of the *AF*s have been mistakenly classified as members of the class.

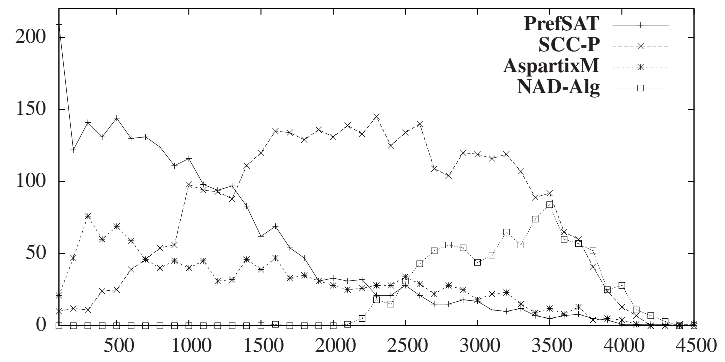

Figure 1. The number of time each solver has been the fastest one w.r.t. the cardinality of the set of arguments.

- using the best 2 features instead of the single best one guarantees better performance;
- SCC-related features are not as informative as we believed;
- features derived from undirected graphs are usually better than those generated by considering directed graphs. This counter-intuitive result (UG are lossy) is further addressed in Section 5. However, no features from the UG set are included in the B3 one: this seems to suggest that undirected graph features are useful while 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 1 shows how the number of vertices of *AF*s affect the performance of the considered solvers. Given the precision performance reported in Table 4, and the shape of the graph in Figure 1, we can argue that this feature is quite informative for selecting between **SCC-P** and **PrefSAT**, which are the fastest systems on most of the considered *AF*s, but does not provide very reliable information about the other solvers.

4.4. 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 on-line algorithm selection. Algorithm selection done by exploiting the classification

Metric Fastest (max. 1007)		Metric IPC (max. 1007, log. scale)	
AspartixM	106	**NAD-Alg**	210.1
NAD-Alg	170	**AspartixM**	288.3
PrefSAT	278	**PrefSAT**	546.7
SCC-P	453	**SCC-P**	662.4
EPMs Regression	755	EPMs Regression	887.7
EPMs Classification	788	EPMs Classification	928.1

Table 5. Algorithm selection performance — w.r.t. single solver (upper part) — using two metrics: the number of time each selection approach and solver has been the fastest one on the testing set of *AF*s (Fastest); and the IPC value (IPC). The approaches are ordered from the worse to the best (according to each metric).

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 extemely 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 *AF*s in separate training and testing instances; the 1,040 testing instances have been randomly selected across all the different sizes. This allows a more objective comparison between the techniques. The time needed for extracting on-line the *AF* features is again negligible w.r.t. the CPU-time needed by the solvers to enumerate the preferred extensions (cf. Section 3). All the solvers failed to enumerate the preferred extensions in the given time on 33 *AF*s over the 1,040 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 shown in the left part of Table 5. 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 *AF*s (78%, clearly consistent with Table 4), 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 worthy noting that there is a remarkable difference between the performance of algorithm selection approaches and the single solvers.

It can be argued that the metric used, the number of time a system has been the fastest, albeit is good for evaluating the algorithm selection performance, might not be very informative from the performance gain perspective. Thus, we analysed the IPC score, borrowed from the 2008 International Planning Competition.[6] 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 solved instances. Again,

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

the maximum IPC score achievable on the testing *AF*s is 1007, which corresponds to a system that always selects the best solver on the instances where at least a solver succeeded. Clearly, on *AF*s 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. **SCC-P** and **PrefSAT** score respectively 662.4 and 546.7; finally **AspartixM** scores 288.3 and **NAD-Alg** 210.1 (right part of Table 5). Also by comparing the systems using the IPC score, a significant difference can be found between algorithm selection methods and single solvers.

5. Conclusions

This paper is the first complete study on EPMs [9] applied to argumentation problems, in particular to the problem of enumerating the preferred extensions. To this aim, we considered the four most recent relevant approaches in the literature, namely **AspartixM** [5], **NAD-Alg** [6], **PrefSAT** [7] and **SCC-P** [8].

One of the contributions of this paper is to introduce 50 features of *AF*s exploited in EPMs development, thus significantly extending our preliminary proposal [16]. In this paper we show that these features allow us to determine the "best" algorithm — relatively to the CPU-time — with an accuracy of about 80%. Moreover, via an empirical investigation over 10,000 *AF*s, 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 (**PrefSAT**). It is worth mentioning that the obtained results are consistent with previous empirical investigations [7]. For instance, from Table 4, the performance of **PrefSAT** depends on the number of vertices (feature B1) much more than **AspartixM** and **NAD-Alg**. In [7] we show that both **AspartixM** and **NAD-Alg** show behaviours independent from the cardinality of the set of arguments, while they depend on the percentage of attacks. Finally, from Table 2, we can observe that the density of directed graph is the most informative feature, according to the CFS attribute evaluator, for predicting runtimes.

Let us notice here that the EPMs which consider features derived from undirected graphs — i.e. without considering the directionality of attacks — are usually better than those generated by considering directed graph ones (cf. Table 4). Since preferred semantics satisfies the *directionality* principle [21], one could see this as a counter-intuitive result. However, it is worth mentioning that both **AspartixM** and **NAD-Alg** 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 undirected graphs.

Nevertheless, this fails to explain the behaviour of **PrefSAT** and **SCC-P**. As to **PrefSAT**, it has to be remarked that its performance depends on the performance of the SAT-solver used.Therefore, further investigations on the translation into SAT problems and its dependency from the *AF* structure should be considered. Moreover, in [7] 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 undirected graph seems to be consistent with the meta-algorithm based on the SCC-recursiveness schema [18], which allows to apply the SAT-solver just into single SCCs.

We believe that this discussion motivates the search for additional features specifically designed for exploring the characteristics of *AF*s in relation with the semantics we are analysing. In fact, with this work we start an investigation whose importance goes beyond the theoretical study of the features: EPMs have been successfully applied to algorithms for solving other types of problems in artificial intelligence [10, 11, 12, 13, 14, 15]. Therefore, similarly to other fields, also industrial applications based on argumentation (e.g. [22]) can take advantage by the EPMs developed in this work — or by this methodology — in order to determine the "best" algorithm for their purpose. We also believe that the results provided can guide the development of more sophisticated solvers.

Future work can be envisaged both from a theoretical point of view — i.e. determining relevant semantics-dependent features — as well as from an "engineering" perspective, by combining algorithms in *portfolios* and considering *probing* features. 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. Probing features are computed by briefly running an existing algorithm on the given *AF* and extracting characteristics from that algorithm's trajectory [9].

References

[1] Phan M Dung. On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games. *Artificial Intelligence*, 77(2):321–357, 1995.

[2] Pietro Baroni and Massimiliano Giacomin. Semantics of Abstract Argumentation Systems. In *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

[3] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Engineering Review*, 26(4):365–410, 2011.

[4] Paul E. Dunne and Michael Wooldridge. Complexity of abstract argumentation. In *Argumentation in AI*, chapter 5, pages 85–104. Springer, 2009.

[5] Wolfgang Dvořák, Sarah Alice Gaggl, Johannes Wallner, and Stefan Woltran. 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)*, pages 114–133, 2011.

[6] Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for decision problems in argument systems under preferred semantics. *Artificial Intelligence*, 207:23–51, 2014.

[7] Federico Cerutti, Paul E. Dunne, Massimiliano Giacomin, and Mauro Vallati. Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach. In *Proceedings of Theory and Applications of Formal Argumentation (TAFA)*, pages 176–193, 2013.

[8] Federico Cerutti, Massimiliano Giacomin, Mauro Vallati, and Marina Zanella. 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)*, 2014. to appear.

[9] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79 – 111, 2014.

[10] Eric A. Brewer. *Portable high-performance supercomputing: high-level platform-dependent optimization*. PhD thesis, Massachusetts Institute of Technology, 1994.

[11] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4):1–52, 2009.

[12] Eugene Nudelman, Kevin Leyton-Brown, Alex Devkar, Yoav Shoham, and Holger Hoos. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Principles and Practice of Constraint Programming - CP 2004*, pages 438–452, 2004.

[13] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[14] Kate Smith-Miles, Jano van Hemert, and . Xin Yu Lim. Understanding TSP difficulty by learning from evolved instances. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization (LION)*, pages 266–280, 2010.

[15] Chris Fawcett, Mauro Vallati, Frank Hutter, Jörg Hoffmann, Holger H. Hoos, and Kevin Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 2014. to appear.

[16] Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Argumentation Frameworks Features: an Initial Study. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 2014.

[17] Pietro Baroni, Federico Cerutti, Paul E. Dunne, and Massimiliano Giacomin. Automata for Infinite Argumentation Structures. *Artificial Intelligence*, 203:104–150, 2013.

[18] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168(1-2): 165–210, 2005.

[19] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[20] Mark A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[21] Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence (Special issue on Argumentation in A.I.)*, 171(10/15):675–700, 2007.

[22] Pietro Baroni, Marco Romano, Francesca Toni, Marco Aurisicchio, and Giorgio Bertanza. An Argumentation-Based Approach for Automatic Evaluation of Design Debates. In *Workshop on Computational Logic in Multi-Agent Systems*, pages 340–356, 2013.