On the Impact of Configuration on Abstract Argumentation Automated Reasoning

Federico Cerutti[a], Mauro Vallati[b], Massimiliano Giacomin[c]

[a] *Cardiff University*
[b] *University of Huddersfield*
[c] *Università degli Studi di Brescia*

Abstract

In this paper we consider the impact of configuration of abstract argumentation reasoners both when using a single solver and choosing combinations of framework representation–solver options; and also when composing portfolios of algorithms.

To exemplify the impact of the framework–solver configuration we consider one of the most configurable solvers, namely **ArgSemSAT**—runner-up of the last competition on computational models of argumentation (ICCMA-15)—for enumerating preferred extensions. We discuss how to configure the representation of the argumentation framework in the input file and show how this coupled framework–solver configuration can have a remarkable impact on performance.

As to the impact of configuring differently structured portfolios of abstract argumentation solvers, we consider the solvers submitted to ICCMA-15, which provided the community with a heterogeneous panorama of approaches for handling abstract argumentation frameworks. A superficial reading of the results of ICCMA-15 is that reduction-based systems (either SAT-based or ASP-based) are always the most efficient. Our investigation, concerning the enumeration of stable and preferred extensions, shows that this is not true in full generality and suggests the areas where the relatively under-developed non reduction-based systems should focus more to improve their performance. Moreover, it also highlights that the state-of-the-art solvers are very complementary and can be successfully combined in portfolios.

Keywords: Abstract Argumentation, Solvers for Argumentation Problems, Automatic Configuration, Portfolios methods for Argumentation

1. Introduction

An abstract argumentation framework (*AF*) consists of a set of arguments and a binary *attack* relation between them. In [1] four semantics were introduced, namely *grounded*, *preferred*, *complete*, and *stable* semantics: each of them leads to a single or to multiple *extensions* (or possibly no extensions in the case of stable semantics) where an *extension* is intuitively a set of arguments which can "survive the conflict together." We refer the reader to [2] for a detailed survey. Moreover, for each semantics, several *decision* and *enumeration* problems have been identified. In this paper we focus on the enumeration of preferred and stable extensions because: (i) the solution to the problem of enumerating extensions allows to easily derive the answer to the other significant problems in abstract argumentation; (ii) the problems of enumerating preferred and stable extensions are among the hardest in abstract argumentation.

One of the solvers that took part in ICCMA-15 was **ArgSemSAT** [3] which, despite a not perfect implementation—e.g. a bug that heavily affected problems related to stable semantics was found after the competition—performed remarkably well, being among the best three solvers for preferred and stable semantics problems, and scoring overall second at a single Borda count point from the winner **CoQuiAAS** [4].

ArgSemSAT is a rather configurable solver: it allows to select different ways for encoding abstract argumentation problems in SAT, and it is able to exploit external SAT solvers. Following the common practice, we manually tuned its parameters on a large set of benchmarks, before submitting it to ICCMA-15; however, the question naturally arises: *is it possible to improve the chosen configuration?*

We investigated whether automatic algorithm configuration systems [5, 6, 7] can address such a question. In this work we extend [8] by (1) including a comprehensive description of configuration possibilities; (2) extending the considered benchmarks with an additional, interesting, structure of argumentation frameworks; and (3) completing the study with an elaborated post-hoc analysis. We exploit the sequential model-based algorithm configuration method SMAC [9], which represents the state of the art of configuration tools. SMAC uses predictive models of algorithm performance [10] to guide its search for performant—according to a chosen metric—configurations. We then experimentally evaluate the impact of automated configuration on the performance of **ArgSemSAT** in enumerating the preferred extensions, testing in particular the following two hypotheses:

1. *Overall SMAC configuration outperforms default configuration.*

2. *Argumentation framework configuration plays a significant role.*

Our findings suggest that much improvement can still come from better engineering of solvers in conjunction with representation of argumentation frameworks. This is a remarkable finding that has been proved only (and very recently) in classical planning [11]. We believe this is an important element that future organisers of competitions should be aware of and take into serious consideration.

A second contribution of the paper is to evaluate, by means of an empirical investigation, whether some take-home messages possibly arising from the results of ICCMA-15[1] are general enough. In particular, the competition results may suggest the following two contradicting hypotheses:

3. *Reduction-based solvers [12] constantly outperforms others.* Indeed, according to the results of the competition the best solvers for enumerating stable and preferred extensions are either SAT-based or ASP-based.
4. *Solvers show complementary performance.* **CoQuiAAS**—that scored first among all for each semantics considered in ICCMA-15—uses a variety of approaches.

Here, complementing our previous work [13] by including an extensive discussion on the used features, we test how general such conclusions are with a large empirical investigation focused on enumeration of stable and preferred extensions using the solvers submitted to ICCMA-15. By adopting different metrics, we identify avenues for improvement that we hope will be valuable for solvers' authors and for the argumentation community. In particular, we have hard evidence contradicting a superficial reading of the results of ICCMA-15, namely that reduction-based systems have consistently higher performance than non reduction-based. This is not always the case, although it is the case that they have better coverage (i.e. they are able to solve more problem instances before the established cutoff time), and ICCMA-15 privileged coverage against speed.

Since solvers prove to be very complementary (i.e. exploiting a mixture of techniques can be fruitful), we also consider different portfolio approaches in order to highlight (relative) strengths and weaknesses of solvers. As testified by experiences in other research areas in artificial intelligence, such as planning [14], SAT [15], and ASP [16], portfolios and algorithm selection techniques [17] are very useful tools for understanding the importance of solvers, evaluate the improvements, and effectively combine solvers for increasing overall performance.

[1] `http://argumentationcompetition.org/2015/results.html`

Existing works in the abstract argumentation field [18, 19] either focus on algorithm selection for enumerating preferred extensions, with a very small number of solvers and of instances; or on theoretical complementariness of algorithms.

In this paper we consider the enumeration of stable and preferred extensions, and we develop both static portfolios, i.e. generated once independently of the specific instance at hand, and per-instance portfolios, i.e. adjusted on the basis of the input AF. We then evaluate the following two hypotheses:

5. *Static portfolios are more efficient than basic solvers.* Given the complementarity between ICCMA solvers, we expect that static portfolios are more efficient than the best ICCMA solver.
6. *Per-instance portfolios are more efficient than static portfolios.* This hypothesis seems sensible, since per-instance portfolios exploit more information than static ones.

As will be shown below, the analysis of portfolio techniques—and their generalisation capabilities—highlight that, by combining solvers, it is possible to increase the coverage by 13% (resp. 3%) and the speed of 51% (resp. 53%) against the best single solver for enumerating preferred (resp. stable) extensions.

The paper is organized as follows. After providing some necessary background in Section 2, we introduce the configuration problem when using the **ArgSemSAT** solver in Section 3, and present the relevant experimental analysis in Section 4. Section 5 empirically evaluates the solvers submitted to ICCMA-15. Motivated by complementarity shown between different approaches, Section 6 introduces the techniques used to combine multiple solvers into portfolios, which are then experimentally evaluated in Section 7. Finally, Section 8 draws some conclusions and highlights avenues for future work.

2. Dung's Argumentation Framework

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

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* $\boldsymbol{b} \rightarrow \boldsymbol{a}$*.*

[2] In this paper we consider only *finite* sets of arguments: see [20] for a discussion on infinite sets of arguments.

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.* $\boldsymbol{a} \rightarrow \boldsymbol{b}$*;*
- *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.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$*,* $\exists\, \boldsymbol{c} \in S$ *s.t.* $\boldsymbol{c} \rightarrow \boldsymbol{b}$*;*
- *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*, namely a set of sets of arguments satisfying the conditions dictated by σ.

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

- complete extension *of* Γ *iff S is an admissible set of* Γ *which includes all the acceptable arguments with respect to S*
- preferred extension *of* Γ *iff S is a maximal (w.r.t. set inclusion) admissible set of* Γ*;*
- stable extension *of* Γ *iff S is a conflict–free set of* Γ *and* $\mathscr{A} \setminus S = \{\boldsymbol{a} \in \mathscr{A} \mid \boldsymbol{b} \rightarrow \boldsymbol{a}$ *and* $\boldsymbol{b} \in S\}$*.*

3. Automated Configuration

After providing some background on automated configuration, this section describes the space of configurations of *AF* representations, and show how they can be combined with the parameters of **ArgSemSAT** and the external SAT solver exploited in it.

3.1. Automated Configuration Techniques

Many algorithms have parameters that can be adjusted to optimise performance (in terms of e.g., solution cost, or runtime to solve a set of instances). Formally, this problem can be stated as follows: given a parameterised algorithm with possible configurations $\mathscr{C}$, a benchmark set Π, and a performance metric $m(c, \pi)$ that measures the performance of a configuration $c \in \mathscr{C}$ on an instance

$\pi \in \Pi$ (the lower the better), find a configuration $c \in \mathscr{C}$ that minimises m over Π, i.e., that minimises

$$f(c) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} m(c, \pi). \tag{1}$$

The AI community has recently developed dedicated algorithm configuration systems to tackle this problem [5, 6, 7]. In this work we exploit the sequential model-based algorithm configuration method SMAC [9], which represents the state of the art of configuration tools and, differently from other existing tools, can handle continuous parameters. SMAC uses predictive models of algorithm performance [10] to guide its search for good configurations. It uses previously observed $\langle$configuration, performance$\rangle$ pairs $\langle c, f(c)\rangle$ and supervised machine learning (in particular, random forests [21]) to learn a function $\hat{f} : \mathscr{C} \to \mathbb{R}$ that predicts the performance of arbitrary parameter configurations, and is used to select a good configuration. Random forests are collections of regression trees, which are similar to decision trees but have real values (here: CPU-time performance) rather than class labels at their leaves. Regression trees are known to perform well for categorical input data. Random forests share this benefit and typically yield more accurate predictions; they also allow to quantify the uncertainty of a prediction. The performance data to fit the predictive models are collected sequentially.

In a nutshell, after an initialisation phase, SMAC iterates the following three steps: (1) use the performance measurements observed so far to fit a random forest model $\hat{f}$; (2) use $\hat{f}$ to select a promising configuration $c \in \mathscr{C}$ to evaluate next, trading off exploration of new parts of the configuration space and exploitation of parts of the space known to perform well; and (3) run c on one or more benchmark instances and compare its performance to the best configuration observed so far.

In order to save time in evaluating new configurations, SMAC first evaluates them on a single training instance; additional evaluations are only carried out (using a doubling schedule) if, based on the evaluations to date, the new configuration appears to outperform SMAC's best known configuration. Once the same number of runs has been evaluated for both configurations, if the new configuration still performs better then SMAC updates its best known configuration accordingly.

SMAC is an *anytime algorithm* (or *interruptible algorithm*) that interleaves the exploration of new configurations with additional runs of the current best configuration to yield both better and more confident results over time. As all anytime algorithms, SMAC improves performance over time, and for finite configuration spaces it is guaranteed to converge to the optimal configuration in the limit of infinite time.

```
arg(a1).
arg(a2).
arg(a3).
att(a1,a3).
att(a2,a2).
att(a3,a1).
att(a3,a2).
```

Listing 1: Γ_E described in Aspartix format.

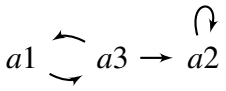


Figure 1: The *AF* Γ_E.

3.2. Configuration of Abstract Argumentation Frameworks

The description of an abstract argumentation framework consists in listing all the arguments and all the attacks of the framework. Currently, three main formats for describing frameworks are used: Trivial Graph Format, Aspartix Format and the CNF Format. Here we focus on the most used one, the Aspartix Format [22].

As a running example, let us consider the framework $\Gamma_E = \langle \mathscr{A}_E, \mathscr{R}_E \rangle$ where $\mathscr{A}_E = \{a1, a2, a3\}$ and $\mathscr{R}_E = \{(a1,a3),(a2,a2),(a3,a1),(a3,a2)\}$: Figure 1 depicts Γ_E. A valid description of Γ_E in the Aspartix Format is provided in Listing 1.

Here, we are investigating a particular aspect of the abstract argumentation frameworks description: *in which order should arguments and attacks be listed so as to maximize the performance of a solver?*

Since this *configuration* of the input file should be performed *online* to lead to improvements of the overall system, we are interested only in order criteria based on information about the *AF* that can be quickly obtained.

In particular, we considered the possibility to list arguments ordered according to the following five criteria: (1) the number of attacks received; (2) the number of attacks to other arguments; (3) the involvement in a self-attack; (4) the difference between the number of received attacks and the number of attacks to other arguments; and (5) being an argument in a mutual attack. For each of the five mentioned criteria, arguments can be listed following a direct or inverse order.

To order the list of attacks, these five criteria can be applied either to the attacking or to the attacked argument. For instance, let us consider Γ_E (Fig. 1)

```
arg(a2).
arg(a3).
arg(a1).
att(a2,a2).
att(a3,a2).
att(a3,a1).
att(a1,a3).
```

Listing 2: Γ_E described in Aspartix format, with the list of arguments according to the number of received attacks and, subsequently, the number of outgoing attacks; and the list of attacks ordered prioritising self-attacks and, subsequently, the number of outgoing attacks.

again. We may be interested in ordering its arguments according to the number of received attacks, breaking ties by considering the number of outgoing attacks from each argument; and in ordering the list of attacks by prioritising self-attacks and breaking ties according to the number of outgoing attacks of the attacking arguments (Listing 2).

There are different ways for encoding the degrees of freedom in *AF* descriptions as parameters, mainly because orders are not natively supported by general configuration techniques. Following [11], we generate 10 continuous parameters, which correspond to the aforementioned possible orderings of arguments and attacks in frameworks. Each continuous parameter has associated a real value in the interval $[-1.0, +1.0]$. A negative value for a criterion indicates that inverse ordering is used, while the absolute value represents the *weight* or *precedence* given to an ordering criterion: the criterion corresponding to the parameter with the highest absolute value is considered first in the ordering. Ties of such an ordering are then broken by referring to the criterion associated to the next parameter of high absolute value. In the case of two criteria having exactly the same absolute value, they are applied according to their alphabetical ordering.

As to ordering the list of attacks, since a criterion can be applied either on the attacking or the attacked argument, an additional categorical selector determines how continuous parameters are exploited for attacks. In particular, the selector allows to decide among 6 alternatives. The first 4 alternatives exploit the 5 continuous parameters for arguments, i.e. applying the same ordering (resp. the inverse ordering) used for listing the arguments to the first argument (resp. the second argument) of the pair of attacks, hence the four alternatives. The other 2 alternatives allows to order the list of attacks independently from the choice of criteria for ordering the list of arguments, i.e. on the basis of additional 5 continuous

parameters for attacks applied on the first or second argument of the attack pair (hence the two additional alternatives). For further details, please see Appendix A.

Summing up, the configuration space is $\mathscr{C} = [-1.0, +1.0]^{10} * 6$, where 6 are the possible values of the categorical parameter describing the order of the list of attacks.

In order to automatically re-order an argumentation framework according to the specified configuration, we developed a wrapper in Python. On the *AF*s considered in our experimental analysis, composed by hundreds of arguments and few hundreds of thousands of attacks, the re-ordering of the Aspartix format description takes less than 1 CPU-time second.

3.3. Joint AF-Solver Configuration

As mentioned in the introduction, we consider **ArgSemSAT** [23]—the runner-up of ICCMA 2015—as a case study for investigating the synergies of re-ordering a given argumentation framework and of selecting the most appropriate solver's parameters. To this purpose, we focus on the enumeration of preferred extensions.

ArgSemSAT is a SAT-based solver which, among other tasks, enumerates stable and preferred extensions performing a search in the space of complete extensions. In particular, the constraints corresponding to complete extensions are encoded into a propositional formula in conjunctive normal form, and an external SAT solver is called to identify a model of this formula, which is directly associated to a complete extension. In this work we exploit the Glucose SAT solver [24], which showed very good performance in recent SAT competitions.

On the one hand, **ArgSemSAT** exposes a single—critical—parameter which allows to select the specific encoding for translating the constraints corresponding to complete extensions into a propositional formula, with remarkable impact on its size and structure, and on the CPU-time required to enumerate all the extensions. On the other hand, Glucose SAT solver has a large number of parameters that can be tuned and controlled for modifying its behaviour, from decay level of variables and clauses, to the number of restarts. Configuring **ArgSemSAT** together with Glucose requires to tune 20 parameters (2 categorical and 18 continuous). Please see the full list of parameters, including default value and valid value range, in Appendix B.

In order to maximise the impact of automated configuration on solvers' performance and thus exploiting unforeseen synergies between solver behaviour and specific knowledge descriptions, we use SMAC for configuring at the same time the *AF* description and the configuration of **ArgSemSAT**. Taking into account the

11 parameters that determine the configuration of the input framework, the total number of configurable parameters is 31, i.e. 3 categorical and 28 continuous.

4. Experimental Analysis of Automated Configuration

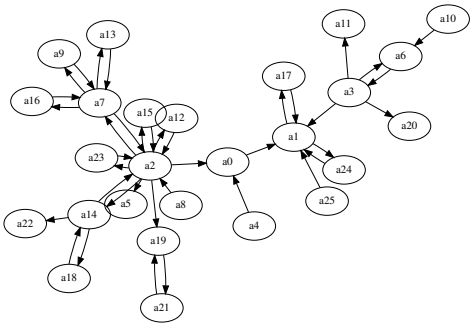
Our experimental analysis aims to evaluate the impact that *AF*s configuration and algorithm configuration, as described in previous sections, has on the state-of-the-art solver **ArgSemSAT**.

4.1. Settings

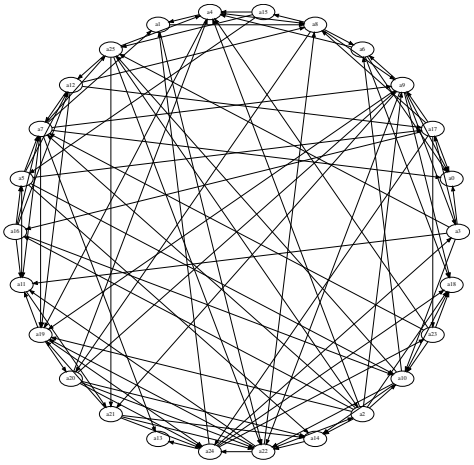
We randomly generated 10,000 *AF*s, divided into 5 sets of 2,000 *AF*s each. Three of such sets include only frameworks based on three corresponding graph models, i.e. Barabasi-Albert [25], Erdös-Rényi [26] and Watts-Strogatz [27]. A fourth set includes graphs featuring a large number of stable extensions—and clearly of preferred extensions as well—hereinafter named StableM (Figure 2d).[3] The fifth set includes mixed-structured *AF*s generated by considering graphs of all the mentioned models: we refer to this set as "General." With the aim of providing a set that allows to easily measure the impact of the configuration process, the *AF*s of the General set have been selected by considering the performance of the default configuration: approximately 60% of the selected instances are successfully analysed.

Erdös-Rényi graphs [26] are generated by randomly selecting attacks between arguments according to a uniform distribution (Figure 2b). On the other hand, Watts and Strogatz [27] show that many biological, technological and social networks are neither completely regular nor completely random, but something in the between. They thus explore simple models of networks that can be tuned through this middle ground: regular networks *rewired* to introduce increasing amounts of disorder. These systems can be highly clustered, like regular lattices, yet have small characteristic path lengths, like random graphs, and they are named *small-world* networks by analogy with the small-world phenomenon (Figure 2c). Finally, as discussed by Barabasi and Albert [25], a common property of many large networks is that the node connectivities follow a scale-free power-law distribution. This is generally the case when: (i) networks expand continuously by the addition of new nodes, and (ii) new nodes attach preferentially to sites that are already well

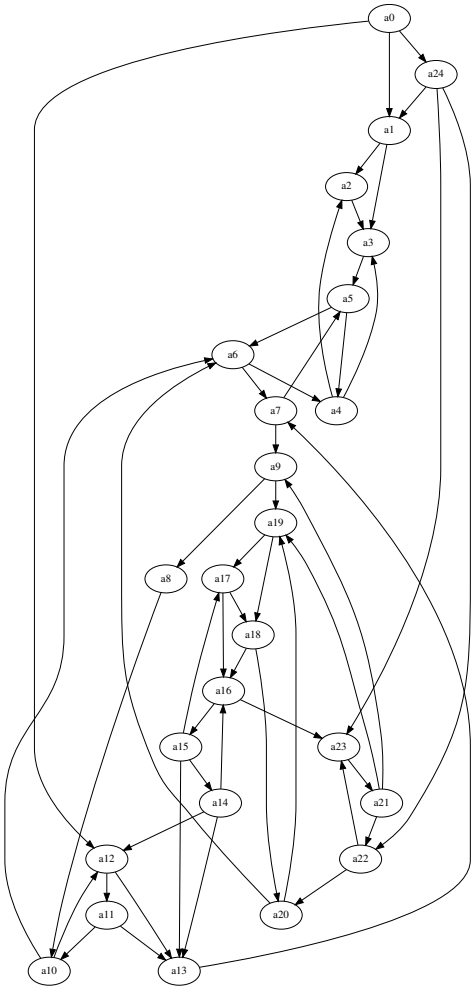
[3] Frameworks of the Barabasi-Albert, Erdös-Rényi, Watts-Strogatz, and StableM sets are available at: `http://helios.hud.ac.uk/scommv/storage/AFs.zip`



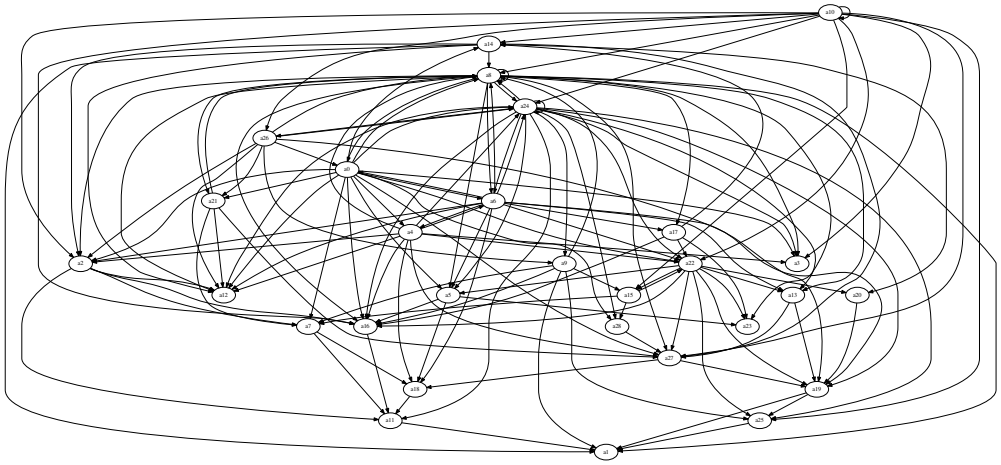
(a) Barabasi-Albert



(b) Erdös-Rényi



(c) Watts-Strogatz



(d) StableM

Figure 2: Examples of randomly generated graph structures

connected (Figure 2a). The *AF*s have been generated by using an improved version of `AFBenchGen` [28], while the StableM set has been generated using the code provided in `Probo` [29] by the organisers of ICCMA-15.[4] It is worth to emphasise that Watts-Strogatz and Barabasi-Albert produce undirected graphs. Each edge of the undirected graph is then associated with a direction following a probability distribution, that can be provided as input to `AFBenchGen`.

In order to identify challenging frameworks—i.e., neither trivial nor too complex to be successfully analysed in the given CPU-time— we followed the protocol suggested in [30] which leads to the selection of *AF*s with a number of arguments between 250 and 650, and number of attacks between (approximately) 400 and 180,000.

Each set of *AF*s has been split into a training set (1,800 *AF*s) and a testing set (200 *AF*s) in order to obtain an unbiased estimate of generalisation performance to previously unseen *AF*s from the same distribution.

Configuration was done using SMAC version 2.10. The performance metric we optimised is the Penalized Average Runtime with a penalty equal to ten times the cutoff time (PAR10). This metric trades off coverage and runtime for successfully analysed *AF*s: runs that do not solve the given problem get ten times the cutoff time, other runs get the actual runtime. The PAR10 score of a solver on a set of *AF*s is the average of the associated scores.

The Wilcoxon sign-rank test [31] is used for comparing performance in terms of PAR10. It exploits the T-distribution which, given a sufficiently large number of samples, is an approximation of a normal distribution, and it is characterised by the *z-value* and the *p-value*. The higher the *z-value*, the more significant the difference of the performance of the compared systems is. The *p-value* indicates the required level of significance of the performance gap. In our analysis we considered that the null-hypothesis, i.e. the performance of compared solvers is statistically similar, is accepted when $p\text{-}value > 0.05$. Otherwise, the null-hypothesis is rejected, and therefore the compared systems' performance is statistically different. For the purposes of this analysis, the Wilcoxon sign-rank test is appropriate because it does not require any knowledge about the sample distribution, and makes no assumption about the distribution.

Experiments were performed on Dual Xeon X5660-2.80GHz with 48GB DDR3 RAM. Each configuration run (i.e. training and testing on one of the 5 sets of 2000 *AF*s) was limited to a single core, and was given an overall runtime of 5 days and

[4] `http://argumentationcompetition.org/2015/results.html`

4 GB of RAM, for ensuring re-usability of results also on less equipped machines. The cutoff time for enumerating the preferred extensions of a single *AF* was 500 seconds.

In the following, also coverage and the International Planning Competition (IPC) score are used for comparing different configurations performance. Coverage corresponds to the percentage of the *AF*s that are correctly solved below the cutoff time. As to IPC score, for a solver $\mathscr{S}$ and a problem p, $Score(\mathscr{S}, p)$ is 0 if p is unsolved, and $1/(1+\log_{10}(T_p(\mathscr{S})/T_p^*))$ otherwise (where $T_p(\mathscr{S})$ is the amount of time required by the solver with the configuration $\mathscr{S}$ to solve the enumeration problem p, and T_p^* is the minimum amount of time required by any compared system). The IPC score on a set of instances is given by the sum of the scores achieved on each considered instance. It is worth noting that the IPC score is designed to highlight the fastest option and heavily penalise the others, even if their performance is not very dissimilar. Instead PAR10 is highly dependent on the coverage, but it does not heavily penalise slower systems. Since the IPC score depends on the set of systems compared, in the following we will consider the set including the default configuration and the one identified by SMAC.

4.2. Hypothesis 1: Overall SMAC Configuration Outperforms Default Configuration

Table 1 compares the performance of **ArgSemSAT** using the default configuration (cf. Appendix B), and the specific joint configuration of *AF* description and **ArgSemSAT**, obtained by running SMAC. Remarkably, the joint configuration of *AF* description and **ArgSemSAT** leads to a general performance improvement. In particular, on the Barabasi-Albert, StableM, and General sets the performance of the configured system are statistically significantly better than the performance achieved by using the default configuration, according to the Wilcoxon test. The significant performance improvement achieved on the General set is of particular interest: it indicates that it is possible to identify a configuration able to improve the performance across differently-structured graphs. In other words, this is an indication that the default configuration of the solver, that was considered as the best possible configuration for solving any general *AF*, can be improved.

Conversely, the configuration process does not significantly improve the default performance on the Watts-Strogatz set. According to the Wilcoxon test, performance of default and tuned configurations are statistically undistinguishable even though IPC score show slight improvements. This is possibly due to the fact that the default configuration is already showing very good performance. In that scenario, it may be the case that only small portions of the configuration space lead

Set	Configuration	IPC Score	PAR10	Coverage
Barabasi-Albert	Default	78.0	1921.0	62.0
	Configured	**125.2**	**1863.1**	**63.0**
Erdös-Rényi	Default	56.8	3426.5	32.0
	Configured	**60.4**	**3329.2**	**34.0**
StableM	Default	90.8	2188.1	57.0
	Configured	**125.7**	**1892.9**	**63.0**
Watts-Strogatz	Default	116.6	**1967.3**	61.0
	Configured	**118.1**	1967.9	61.0
General	Default	110.0	1665.4	68.0
	Configured	**143.0**	**1376.8**	**73.5**

Table 1: Comparison between the default and tuned configuration, in terms of IPC score, PAR10, and coverage (percentage), on the considered *AF*s sets. In bold the best results.

to a significant performance improvement over the default configuration. Given the limited CPU-time made available to the configuration process, SMAC did not identify such portions of the vast configuration space.

Finally, the results on the Erdös-Rényi set deserves a more detailed discussion. On the one hand, the Wilcoxon test indicates that there is not a statistically significant performance improvement. On the other hand, *AF*s from the Erdös-Rényi set are extremely hard for **ArgSemSAT**, as testified by the coverage values. Moreover, those that can be solved are usually solved quickly, i.e. in few CPU-time seconds. This makes the evaluation of configurations' performance, and the exploration of the space of configurations, hard and slow. Despite such issues, SMAC was able to identify a configuration that is able to improve the performance both in terms of runtime (better IPC score) and coverage. Overall, this is a remarkable result, that shows the ability of automated configuration in improving performance also in unfavourable cases.

4.3. Hypothesis 2: Argumentation Framework Configuration Plays a Significant Role

In order to shed some light on the usefulness of algorithm and *AF* tuning, we assessed the importance of parameters in the considered graph-specific configurations. We used fANOVA [32], a recently-released tool for assessing parameter importance after each configuration. fANOVA exploits predictive models of the performance of each configuration for assessing the importance of each parame-

Set	1st	2nd	3rd
Barabasi-Albert	S-ExtEnc (011111)	G-firstReduceDB (1528)	G-cla-decay (0.32)
Erdös-Rényi	F-autoFirst (-1.00)	G-rnd-freq (0.00)	G-K (0.26)
StableM	G-var-decay (0.98)	G-firstReduceDB (2000)	G-incReduceDB (267)
Watts-Strogatz	S-ExtEnc (101010)	G-Grow (0)	G-rnd-freq (0.08)
General	S-ExtEnc (101010)	G-R (2.09)	G-cla-decay (0.99)

Table 2: Most important single parameters (configured value) for SMAC runs on the considered *AF* sets. F-, S- and G- stand for, respectively, Framework, **ArgSemSAT** and Glucose parameters.

ter, regardless of the value of the others, and the interaction between parameters' values. Table 2 shows the three most important parameters for each configuration. Unsurprisingly, the encoding used by **ArgSemSAT** for generating the SAT formulae is usually the most important parameter (see Appendix B). Default value of this parameter is 101010, proven to be the best choice for *AF*s belonging to Watts-Strogatz and General sets, but not for *AF*s in the Barabasi-Albert set. After that, the parameters that control the behaviour of Glucose are those with the highest impact on performance, notably: decay value of clauses and size of the DB of learnt clauses are among the aspects with the strongest impact on the performance of **ArgSemSAT**.

One parameter used for controlling the *AF* description has a significant impact on performance on *AF*s belonging to the Erdös-Rényi set, according to the fANOVA tool. This is the parameter that orders arguments according to the presence of self-attacks. However, `AFBenchGen` does not allow the generation of any self-attacking argument for Erdös-Rényi *AF*s. Interestingly, due to the way in which the configurator has been implemented, setting the `autoFirst` parameter value to -1.0, when no self-attacking arguments are included, leads to listing the arguments in a complete inverse order than the one provided by the generator. This result suggests that `AFBenchGen` orders the arguments in a peculiar way, that has a detrimental impact on the solver's performance.

However, the interaction of parameters controlling the shape of *AF*s with reasoning-related parameters do have a remarkable impact, i.e. the best performance depends on two or more parameters. Parameters used for controlling the order of arguments have strong interactions with the parameter that controls the encoding of **ArgSemSAT**, as well as with parameters of Glucose controlling the number and type of clauses learnt. Figure 3 (coloured) shows the average PAR10 performance of **ArgSemSAT** on the Barabasi-Albert set as a function of two in-

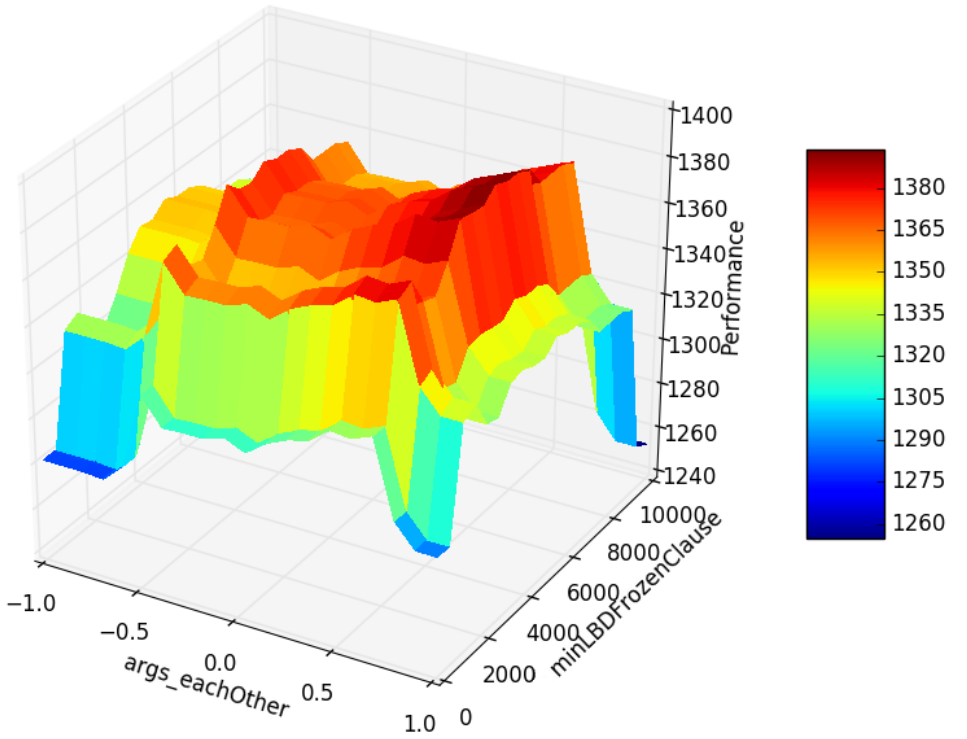


Figure 3: (Coloured) The average PAR10 performance of **ArgSemSAT** on the Barabasi-Albert set, as a function of the ordering of arguments according to the fact that they attack each other (`args_eachOther`) and of number of clauses stored by Glucose during the search. Lower PAR10 values correspond to better performance.

teracting parameters. `args_eachOther` is used for listing earlier in the *AF* description arguments that are attacking each other, the other parameter is used for controlling the number of Glucose learnt clauses, according to their heuristic LBD (Literal Block Distance) evaluation. Figure 3 shows that, in order to achieve better performance in terms of PAR10, both arguments have to be taken into account. Specifically, arguments attacking each other should be listed very late (-1.0 value of the parameter) or very early ($+1.0$ value) and either few or many clauses should be kept (respectively, low and high value of Glucose parameter). The peculiar shape of the impact figure is probably due to the way in which the parameter for ordering the arguments works: the same value can result in an ascending (positive) or descending (negative) order, according to the sign.

Parameters that control the order in which arguments are listed tend to have a

stronger impact on overall performance—either singularly (Table 2) or as a result of their interaction with other parameters (Fig. 3)—than parameters controlling the order in which attacks are listed. At a first sight, this may be seen as counter-intuitive, since the number of attacks in a typical benchmark *AF* is significantly higher than the number of arguments. However, this difference can be due to the data structure used by **ArgSemSAT**. The set of arguments of the *AF* is stored in a list which is populated according to the order in which the arguments are listed in the input file. Each argument has an associated data structure with pointers to two other lists of arguments: one for the attacked arguments; and one for the arguments that attack it. Then the list representing the set of arguments of the *AF* is navigated several times when creating CNFs to be evaluated by the SAT solver. Let us also reiterate here that we considered the problem of enumerating preferred extensions which requires iterative calls to a SAT solver with significant manipulations [33]. These results suggest that not only the encoding of complete labellings in CNF, but also the order of clauses have a remarkable impact on the performance.

4.4. Post-Hoc Analysis: Robustness of SMAC Configuration

We can identify three typologies of argumentation frameworks from the sets we used in our experimentation:

T1: frameworks generated obeying extension properties: StableM;

T2: frameworks generated obeying graph-theoretical properties: Barabasi-Albert, Erdös-Rényi, and Watts-Strogatz;

T3: a mixture of all of those: General.

In this section we investigate, as a post-hoc analysis, the robustness of the found configurations. We therefore proceed with two post-hoc analyses, one *intra-typology* (e.g. comparing the configurations within the same typology set); and the other *inter-typology*. For the latter, we are in particular interested in understanding the generality of the results, i.e. whether the General set of *AF*s carries enough information to support the claim that its best configuration can be directly applied to frameworks belonging to different typologies.

4.4.1. Intra-Typology Post-Hoc Analysis

Since both T1 and T3 have one member each only, these cases are already discussed in Table 1. Let us then focus on T2, namely the set of frameworks

Test Set	Training Set	IPC Score	PAR10	Coverage
Barabasi-Albert	Barabasi-Albert	**119.5**	1836.1	63.0
	Erdös-Rényi	92.4	1870.1	63.0
	Watts-Strogatz	116.5	**1795.9**	**64.5**
Erdös-Rényi	Barabasi-Albert	7.1	4569.4	9.0
	Erdös-Rényi	**60.5**	**3329.2**	**34.0**
	Watts-Strogatz	55.1	3430.8	32.0
Watts-Strogatz	Barabasi-Albert	36.9	2581.2	49.0
	Erdös-Rényi	108.3	**1922.1**	**62.0**
	Watts-Strogatz	**119.3**	1968.0	61.0

Table 3: Intra-T2-typology post-hoc analysis on the generalisability of the configuration obtained training on frameworks belonging to the T2 typology. IPC Score is evaluated by considering all the configurations on each single test set: therefore, IPC scores cannot be directly compared between different test sets. In bold the best results, with respect to each specific test set.

generated obeying graph-theoretical properties, such as Barabasi-Albert, Erdös-Rényi, and Watts-Strogatz.

Table 3 shows the results of this comparison within the T2 typology of argumentation frameworks. Leaving apart the configuration derived from Barabasi-Albert training *AF*s, the configurations derived from Erdös-Rényi and Watts-Strogatz *AF*s exhibit performance on differently structured testing sets that are similar to performance on *AF*s with the same structure. This possibly indicates that there are some parameters' values that can boost performance on differently-structured *AF*s.

On the other hand, the configuration derived from Barabasi-Albert training does not generalise well on differently-structured *AF*s. Such a behaviour is possibly due to some parameter's value that helps increasing the performance on the Barabasi-Albert test set, but has a detrimental effect on different graph structures. For instance, among considered structures, Barabasi-Albert is the only set of *AF*s with a large number of preferred extensions (several tens) per *AF*.

Finally, it is worth noticing that only in the case of testing on Erdös-Rényi frameworks the configuration derived from a training over frameworks structured as the testing ones provides consistently the best performance. In the case of Barabasi-Albert and Watts-Strogatz frameworks, depending on the chosen metric there are two different "optimal" configurations. In the case of testing on Barabasi-Albert frameworks, the best configuration according to PAR10 (and, unsurprisingly the coverage as well) is training on Watts-Strogatz frameworks; ac-

Test Set	Training Set	IPC Score	PAR10	Coverage
StableM	General	100.2	2018.4	60.5
	StableM	**125.3**	**1892.9**	**63.0**
Barabasi-Albert	General	87.7	1917.6	62.0
	Barabasi-Albert	**119.8**	1863.1	63.0
	Watts-Strogatz	116.6	**1795.9**	**64.5**
Erdös-Rényi	General	**61.6**	**3301.6**	**34.5**
	Erdös-Rényi	61.6	3329.2	34.0
Watts-Strogatz	General	113.5	1966.5	61.0
	Erdös-Rényi	105.3	**1922.1**	**62.0**
	Watts-Strogatz	**115.5**	1968.0	61.0

Table 4: Inter-typology post-hoc analysis on the generalisability of the configuration obtained training on General set compared to the best configuration identified in the intra-typology post-hoc analysis, cf. Tables 1 and 3. In case from Table 3 is not possible to identify a single clear winner, we considered all the winner "candidates."

cording to IPC is training on Barabasi-Albert frameworks. In the case of testing on Watts-Strogatz frameworks instead for PAR10 and coverage it is better to train over Erdös-Rényi frameworks, but not for IPC, for which it is better to train over Watts-Strogatz frameworks. This is due to the fact that PAR10 is highly coupled with the coverage measurement.

4.4.2. Inter-Typology Post-Hoc Analysis

Table 4 shows the results of the inter-typology post-hoc analysis aimed at identifying the degree of generalisibility of configurations obtained on the General training set on testing sets belonging to homogeneous typologies.

As to the inter-typology analysis with T1, namely frameworks generated obeying to argumentation properties (i.e. StableM), Table 4 seems to be fairly conclusive that the configuration obtained using General as training set is not competitive against the one obtained training over StableM. This is also due to the fact that the General set is not perfectly balanced, as only one fourth of its frameworks belongs to the T1 typology, versus three fourth of frameworks belonging to the T2 typology.

As to the inter-typology analysis with T2, the configuration identified by training on the General set is usually able to obtain performance that are close to—or even better than—those of the specific configuration, on each considered test set. This seems to support the hypothesis that there are some parameters' values that

Set	Configuration	IPC Score	PAR10	Coverage
General	Default	113.4	1866.0	63.5
	Configured	**144.4**	**1204.5**	**77.0**

Table 5: Comparison between the default and tuned configuration, in terms of IPC score, PAR10, and coverage (percentage), on the considered *AF*s sets for **ArgSemSAT** exploiting MiniSAT. In bold the best results.

can help improving the general performance. Remarkably, results presented in Table 4 highlight a potential limit of the configuration approach. As previously observed, *AF*s from the Erdös-Rényi set are extremely challenging for **ArgSemSAT**; this is true for both training and testing instances. It is therefore reasonable to assume that the configuration approach did not assess a large number of configurations due to the frequent timeouts of the solver and to the limited overall CPU-time available for the learning process. This may result—as it is the case for the General configuration on the Erdös-Rényi test set—in having other configurations, optimised by successfully analysing a much larger number of training instances, that performs better on the specific test set.

4.5. Post-Hoc Analysis: Joint Configuration and a Different SAT Solver

In this section we show how the configuration process leads to a significant performance improvement, in line with the results shown in Section 4.4, even when **ArgSemSAT** uses MiniSAT instead of Glucose as SAT solver. We considered MiniSAT [34], given its wide availability, and the fact that it is commonly used as a framework for developing SAT solvers. As training and testing benchmarks, we focused on the General set, which has been shown to provide a good training set for configurations that generally increase the performance of **ArgSemSAT**.

Table 5 compares the performance of **ArgSemSAT** using MiniSAT with the default configuration, and the specific joint configuration of *AF* description and MiniSAT, obtained by running SMAC on the General benchmark set. This suggests that the joint configuration plays a significant role in the increase of performance.

The analysis of parameters' importance for the obtained configuration, performed using the fANOVA tool, confirmed that—beside parameters controlling the internal behaviour of the solver—parameters used for controlling the *AF* description play a significant role. In particular, the value of the `args_eachOther`

parameter—used for listing earlier in the *AF* description arguments that are attacking each other—can improve **ArgSemSAT**'s performance by 4%. This parameter plays a significant role also on the performance of **ArgSemSAT** running with the Glucose SAT solver. Moreover, fANOVA identified parameters `args_ingoingFirst` and `args_differenceFirst`—used respectively for listing first the arguments that show a high number of incoming attacks and the arguments with a high difference between incoming and outgoing attacks—as having a noticeable impact on the overall performance.

5. Experimental Analysis of ICCMA-15 Solvers

For this experimental analysis we reused the same benchmark frameworks described in Section 4.1, extracting from them 2,000 *AF*s based on four different graph models: Barabasi-Albert [25], Erdös-Rényi [26], Watts-Strogatz [27] and graphs featuring a large number of stable extensions (StableM).[5]

The set of *AF*s has been divided into training and testing sets. For each graph model, we randomly selected 200 *AF*s for training, and the remaining 300 for testing. Therefore, out of the 2,000 *AF*s generated, 800 have been used for training purposes, while the remaining 1,200 have been used for testing and comparing the performance of trained approaches.

We considered all the solvers that took part in the EE-PR and EE-ST tracks of ICCMA-15 [35], respectively 15 and 11 systems. For the sake of clarity and conciseness, we removed from the analysis single solvers that did not correctly solve at least one *AF* within cutoff time or which were always outperformed by another solver. The interested reader is referred to [36] for detailed descriptions of the solvers. Hereinafter, we will refer to such systems as *basic solvers*, regardless of the approach they exploit for solving argumentation-related problems (this is to distinguish them from portfolios of algorithms, described in the next sections, which use basic solvers as components).

Experiments have been run on a cluster with computing nodes equipped with 2.5 Ghz Intel Core 2 Quad Processors, 4 GB of RAM and Linux operating system. A cutoff of 600 seconds was imposed to compute the extensions—either preferred or stable—for each *AF*. For each solver we recorded the overall result: success (if it solved the considered problem), crashed, timed-out or ran out of memory.

In ICCMA, solvers have been evaluated by considering only coverage (and

[5] Frameworks are available at: `http://helios.hud.ac.uk/scommv/storage/AFs.zip`

in case of ties the overall runtime on solved instances). Here we also evaluate solvers' performance by considering the PAR10 score.

5.1. Hypothesis 3: Reduction-based Solvers Constantly Outperform Others

Table 6 shows the results of this analysis in terms of coverage, PAR10 scores, and number of instances on which a given solver has been the fastest. We considered runtimes below 1 CPU-time second as equally fast.

Each *basic solver* for the EE-PR problem has at least one instance (i.e. one *AF*) on which it is the fastest (cf. **F.t**). We note that, when considering performance achieved on the whole testing set (**All**) by solvers, there can be a significant discrepancy between results shown in the coverage and fastest columns. One would expect that the higher the coverage, the larger the possibility of a solver to be the fastest. Interestingly, we observed that some of the solvers with low coverage tend to be fast on the (few) instances they are able to analyse. For instance, **ArgTools** (a non reduction-based system) achieves low overall coverage, but it is the best solver for handling *AF*s of the Erdös-Rényi set. This contradicts the hypothesis—endorsed by a superficial reading of the results of ICCMA-15—that reduction-based systems constantly outperform others.

The best *basic solver* for solving the EE-PR problem on the StableM set of *AF*s is **Cegartix**, which is able to solve 77.0% of the instances. This is approximately 10% more than the coverage of the second best solver on such set, **ASPARTIX-D**. The **prefMaxSAT** solver has shown the best performance on the Watts-Strogatz *AF*s. From an (empirical) complexity perspective, we observe that the set with the lowest average coverage is the Barabasi-Albert set of *AF*s. This is possibly due to the very large number (up to few thousands, in some cases) of preferred extensions of such testing frameworks. In this set **GRIS** is able to obtain a very good coverage and a very low PAR10, probably because it exploits a decomposition of the *AF* into strongly connected components [37]. Conversely, the Erdös-Rényi set is the less complex for the considered *basic solvers* when solving the EE-PR problem. Moreover we can derive that even though there is usually a *basic solver* with best coverage performance on each testing set, such solver is not always the fastest (see the column **F.t**).

As for the EE-ST problem, the results in Table 6 show another interesting scenario. **ArgTools** is able to achieve the best PAR10 and coverage performance on two of the four considered sets, namely StableM and Watts-Strogatz. The best PAR10 score on the Erdös-Rényi set is obtained by **LabSATSolver** but four of the considered *basic solvers* successfully analyse each of the 300 *AF*s in such a set. The winner of the EE-ST track of ICCMA-15, **ASPARTIX-D**, has been the

EE-PR

	All			Barabasi-Albert		Erdös-Rényi		StableM		Watts-Strogatz	
Solver	PAR10	Cov.	F.t	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
Cegartix	**1350.4**	**79.1**	229	1662.6	74.2	1266.6	81.0	**1439.2**	**77.0**	1028.6	84.2
ArgSemSAT	1916.2	69.1	35	3532.3	41.9	433.7	94.2	2530.9	58.7	1171.1	81.5
LabSATSolver	2050.3	66.8	9	3430.7	43.5	261.3	96.5	2869.5	53.0	1657.5	73.9
prefMaxSAT	2057.2	66.8	273	3482.1	42.9	444.0	94.2	3625.2	40.3	**697.5**	**89.4**
DIAMOND	2417.0	61.0	1	3447.8	43.2	1366.7	79.0	2831.8	53.7	2026.0	68.0
ASPARTIX-D	2728.6	56.1	4	4101.5	32.6	3067.8	51.6	2068.8	66.7	1630.3	74.3
ASPARTIX-V	2772.2	55.2	21	3646.6	40.3	3292.6	47.1	2340.7	62.0	1772.4	71.9
CoQuiAAS	3026.4	50.5	78	3736.1	38.4	2873.4	53.5	2836.4	53.3	2645.1	57.1
ASGL	3477.3	43.2	1	4809.7	20.3	96.1	**100.0**	4475.4	26.0	4585.5	25.4
ConArg	3696.3	39.3	158	1128.7	81.6	2813.9	55.8	4934.6	18.3	6000.0	0.0
ArgTools	3906.2	35.2	**322**	3694.4	39.0	**45.2**	**100.0**	6000.0	0.0	6000.0	0.0
GRIS	4543.7	24.4	174	**254.6**	**96.1**	6000.0	0.0	6000.0	0.0	6000.0	0.0

EE-ST

	All			Barabasi-Albert		Erdös-Rényi		StableM		Watts-Strogatz	
Solver	PAR10	Cov.	F.t	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.
ArgTools	440.7	94.5	245	1328.6	78.4	47.4	**100.0**	**144.1**	**100.0**	**230.5**	**100.0**
LabSATSolver	641.6	90.0	352	396.2	93.9	**22.7**	**100.0**	1497.6	76.0	684.9	90.7
ASPARTIX-D	829.7	87.1	**395**	412.2	93.5	1194.4	81.6	1187.2	81.0	535.0	93.0
CoQuiAAS	1477.2	76.2	372	1453.3	76.5	1485.1	76.5	1879.0	69.3	1106.5	83.3
DIAMOND	1555.4	75.2	42	2527.1	58.7	692.2	89.7	1887.2	69.7	1127.1	83.7
ArgSemSAT	1826.6	70.5	70	4019.0	33.5	408.9	94.5	1970.0	68.0	900.8	87.0
ConArg	1976.4	67.8	292	**261.4**	**96.1**	33.6	**100.0**	3742.1	38.3	4010.0	35.3
ASGL	2647.6	57.3	11	2737.4	56.1	85.2	**100.0**	3723.8	38.7	4152.8	33.7

Table 6: PAR10 score and coverage (cov.)—percentage of *AF*s successfully analysed—of the considered *basic solvers* for solving the preferred enumeration (upper table) and stable enumeration (lower table) problems on the complete testing set (All) of 1,200 *AF*s, and on testing sets including *AF*s generated by specific graph models. Solvers are ordered according to PAR10 on the All testing set. F.t column indicates the number of times a solver has been the fastest among considered. Best results in bold.

fastest solver on 395 of the testing frameworks, but it did never excel in any of the 4 considered subsets. The *AF*s of the StableM set are (empirically) the most complex to solve for the considered systems, which is unsurprising if we take into account that they have been constructed in order to obtain a large number of stable extensions.

5.2. Hypothesis 4: Basic Solvers show Complementary Performance

Table 6 indicates that there is not a *basic solver* that is always the best selection on the vast majority of the testing frameworks. This is evidence that the *basic solvers* are substantially complementary, thus supporting the claim that a mixture of approaches can be fruitful, and justifying the search for improvements via portfolios.

6. Generation of Portfolios of Multiple Solvers

In this section we describe the techniques we used for combining multiple solvers into sequential portfolios. Every approach requires as input a set of solvers, a set of training *AF*s, and measures of performance of solvers on the training set. Solvers are treated as black-boxes, and no communication is allowed between different solvers. As in Section 4, solvers' performance are measured in terms of Penalised Average Runtime (PAR10) score. Although PAR10 largely emphasises the coverage, it also gives a clear indication on runtime effective performance, thus resulting in an interesting and useful measure.

6.1. Static Portfolios

Static portfolios—as the name suggests—are generated once, according to the performance of the considered solvers on training instances, and never adjusted. Static portfolios are defined by: (i) the selected solvers; (ii) the order in which solvers will be run, and (iii) the runtime allocated to each solver.

We considered two different approaches for configuring static portfolios. A first approach consists in static portfolios of exactly k components, called *Shared-k*. Each component solver has been allocated the same amount of CPU-time, equal to $maxRuntime/k$ seconds. Solvers are selected and ordered according to overall PAR10 score achieved by the resulting portfolio. We considered values of k between 2 and 5. In fact, $k = 1$ would be equivalent to select the single solver with the best PAR10 score on training instances, which is not relevant for our investigation. For $k > 5$, the CPU-time assigned to each solver tends to be too short hence drastically reducing portfolio performance.

For our second static portfolio approach, named *FDSS*, we adapted the Fast Downward Stone Soup technique [38]. We start from an empty portfolio, and iteratively add either a new solver component, or extend the allocated CPU-time[6] of a solver already added to the portfolio, depending on what maximises the increment of the PAR10 score of the portfolio. We continue until the time limit of the portfolio has been reached, or it is not possible to further improve the PAR10 score of the portfolio on the training instances.

6.2. Per-instance Portfolios

Per-instance portfolios rely on instance features for configuring an instance-specific portfolio. For each *AF* a vector of features is computed; each feature is a real number that summarises a potentially important aspect of the considered *AF*. Similar instances should have similar feature vectors, and, on this basis, portfolios are configured using empirical performance models [10].

6.2.1. Abstract Argumentation Features

In this investigation we consider the largest set of features available for *AF*s [18]. Such set includes 50 features, extracted by exploiting the representation of *AF*s both as directed (loss-less) or undirected (lossy) graphs.

We are able to extract 26 features from the representation of *AF*s as direct graphs. Each feature belongs to one of the following four classes: *graph size* (5), *degree* (4), *SCC* (5), *graph structure* (5), *times* (7).

- *Graph size features*: number of vertices, number of edges, ratios vertices–edges and inverse, and graph density (NT – non trivial).[7]
- *Degree features* (overall NT): average, standard deviation, maximum, minimum degree values[8] across the nodes in the graph.
- *SCC features* (overall NT): number of SCCs, average, standard deviation, maximum and minimum size of SCCs.

[6] A granularity of 5 CPU-time seconds is considered.

[7] 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). A class is called overall NT if all of its features are NT.

[8] The degree value of a node is the sum of its indegree and its outdegree.

- *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.

The features extracted by considering the undirect graph representation of *AF*s — i.e. replacing each directed attack with an undirected edge — are 24, belonging to six classes: *graph size* (4), *degree* (4), *components* (5), *graph structure* (1), *triangles* (5), *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 of connected components.

- *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.

The features extraction process is usually quick (less than 2 CPU-time seconds on average) and is done by exploiting a wrapper written in Python.

6.2.2. Classification-based approach

The classification-based (hereinafter *Classify*) approach exploits the technique introduced in [18]. It trains a random decision forest classification model to perform algorithm selection. It classifies a given *AF* into a single category which corresponds to the single solver predicted to be the fastest. The difference between solvers' performance is ignored: all the available CPU-time is then allocated to the selected solver.

6.2.3. Regression-based approaches

For regression-based approaches, deciding which solver to execute and its runtime depends on the empirical hardness models learned from the available training data, in particular a M5-Rules [39] model generated for each solver. When executed on a fresh AF, the predictive model estimates the CPU-time required by each solver to successfully terminate.

We exploit the regression-based model in two different ways. First, for performing algorithm selection (hereinafter *1-Regression*): given the predicted runtime of each solver, the solver predicted to be the fastest is selected and it is allocated all the available CPU-time. However, such use of the model does not fully exploit the available predicted runtimes. Therefore, we designed a different way for using the regression-based approach, referred to as *M-regression*. As in 1-Regression, we initially select the solver predicted to be the fastest, but we allocate only its predicted CPU-time (increased by 10% in order to mitigate the impact of negligible prediction mistakes). If the selected solver is not able to successfully analyse the given AF in the allocated time, it is stopped and no longer available to be selected, and the process iterates by selecting a different solver. The M-regression approach stops when either a solver has successfully analysed the AF, or the runtime budget has been exhausted.

With regards to existing well-known portfolio-based solver approaches, it is worthy to remark that SATZilla [15] is a regression-based approach similar to the 1-regression we introduced. However, since it was developed for competition purposes, SATZilla also exploits pre and backup solvers. These are undoubtedly useful for improving coverage, but not when the main point is to evaluate to which extent solvers composition/selection can improve results, as in our investigation.

7. Experimental Analysis of Portfolios

First of all, we generated the Virtual Best Solver (*VBS*) as the (virtual) oracle which always select the best solver (as to PAR10) for the given problem and instance (i.e. argumentation framework). This provides the upper bound of performance achievable by combining considered solvers.

As to static solvers, for the preferred semantics the basic solvers included in the Shared-5 portfolio, ordered following their execution order, are: **Cegartix**, **ArgSemSAT**, **prefMaxSAT**, **LabSATSolver** and **DIAMOND**. Smaller static portfolios include subsets of those 5 solvers, not necessarily in that order. FDSS static portfolio includes **ArgSemSAT** and **GRIS**, only.

For the stable semantics, the solvers included in the Shared-5 portfolio, ordered following their execution order, are: **LabSATSolver**, **ArgTools**, **ASPARTIX-D**, **CoQuiAAS** and **DIAMOND**. Smaller portfolios include subsets of the listed solvers, not necessarily in that order. The FDSS portfolio includes **LabSATSolver** and **ASPARTIX-D**.

As to per-instance portfolios, we generated the three per-instance (per-problem) portfolios that exploit predictive models in order to map the features of the given *AF* to a solver selection or combination: *Classify*, *1-Regression*, and *M-Regression*.

We trained all the portfolio approaches using our training set of 800 *AF*s, 200 *AF*s from each set. The runtime cutoff once again was 600 CPU-time seconds. Table 7 shows the coverage and PAR10 scores of all portfolios, *basic solvers* and the *VBS* on the testing frameworks.

7.1. Hypothesis 5: Static Portfolios are more Efficient than Basic Solvers

Results for the static portfolios vary between stable and preferred semantics. When dealing with the EE-PR problem, the FDSS approach is the only technique which is able to outperfom the best *basic solver*. *Shared-2* and *Shared-3* achieve performance close to those of the best *basic solver*, while *Shared-4* and *Shared-5* are undistinguishable from average *basic solvers*. FDSS portfolio performs better than *Shared-k* static portfolios because it includes **GRIS**. **ArgSemSAT** has good coverage, and **GRIS** excels on the Barabasi-Albert set (Table 6), while *Shared-k* portfolios do not include any solver able to efficiently solve the EE-PR problem on the Barabasi-Albert set.

Conversely, the right part of Table 7 shows that on the EE-ST problem, both *Shared-2* and *Shared-3* are able to achieve better performance than any *basic solver*, and the FDSS portfolio. Shared portfolios performance are boosted by the inclusion of **ArgTools**, which is able to achieve the best performance on three of the considered benchmark set structures, and **CoQuiAAS**—that is the second best *basic solver* in terms of number of *AF*s quickly analysed. Moreover, the EE-ST problems are usually quickly solved by the *basic solvers*, therefore 2 or 3 solvers can be easily executed within the 600 CPU-time seconds limit. When more than three solvers are combined by the Shared approach—i.e. the CPU-time allocated to each *basic solver* is less than 200 seconds—performance drops.

7.2. Hypothesis 6: Per-Instance Portfolios are more Efficient than Static Portfolios

When considering per-instance portfolios, Table 7 indicates that they are all able to outperfom the best *basic solver* on the considered testing frameworks. This

EE-PR			EE-ST		
System	**Cov.**	**PAR10**	**System**	**Cov.**	**PAR10**
VBS	91.4	562.9	*VBS*	100.0	39.3
Classify	89.7	665.2	*1-Regression*	97.4	206.9
1-Regression	88.6	734.7	*Classify*	97.1	217.5
M-Regression	82.8	1068.3	*Shared-2*	97.7	262.3
FDSS	80.0	1311.4	*M-Regression*	94.7	378.4
Cegartix	79.1	1350.4	*Shared-3*	94.0	420.1
Shared-2	73.2	1678.0	**ArgTools**	94.5	440.7
Shared-3	69.4	1892.0	**LabSATSolver**	90.0	641.6
ArgSemSAT	69.1	1916.2	*FDSS*	89.4	677.4
LabSATSolver	66.8	2050.3	**ASPARTIX-D**	87.1	829.7
prefMaxSAT	66.8	2057.2	*Shared-5*	86.3	867.4
Shared-4	65.7	2105.5	*Shared-4*	86.0	873.8
Shared-5	63.3	2240.3	**CoQuiAAS**	76.2	1477.2
DIAMOND	61.0	2417.0	**DIAMOND**	75.2	1555.4
ASPARTIX-D	56.1	2728.6	**ArgSemSAT**	70.5	1826.6
ASPARTIX-V	55.2	2772.2	**ConArg**	67.8	1976.4
CoQuiAAS	50.5	3026.4	**ASGL**	57.3	2647.6
ASGL	43.2	3477.3			
ConArg	39.3	3696.3			
ArgTools	35.2	3906.2			
GRIS	24.4	4543.7			

Table 7: Coverage (Cov.) and PAR10 of the systems considered in this study for solving the EE-PR problem (left part) and the EE-ST problem (right part) on the complete set of 1,200 testing *AF*s. VBS indicates the performance of the virtual best solver. Systems are ordered according to PAR10.

	EE-PR		EE-ST	
System	Class.	M-Reg.	Class.	M-Reg.
ArgSemSAT	0	253	0	212
ArgTools	**311**	305	138	428
ASGL	6	36	0	35
ASPARTIX-D	2	80	**305**	409
ASPARTIX-V	1	99		
Cegartix	221	**403**		
ConArg	157	122	231	337
CoQuiAAS	43	44	288	193
DIAMOND	0	65	33	138
GRIS	153	278		
LabSATSolver	13	208	228	**548**
prefMaxSAT	297	301		

Table 8: Number of times each solver has been selected by the Classify (Class.) or M-Regression (M-Reg.) approaches for solving EE-PR (left part) and EE-ST (right part) problems on the testing frameworks. *Basic solvers* are alphabetically ordered. Highest numbers in bold. Empty cells indicate that the corresponding solver is not able to handle the considered problem.

comes as no surprise, since per-instance approaches should be able to select the most promising—ideally, the fastest—algorithm for solving the considered problem on the given AF. For both EE-PR and EE-ST problems, the performances of *Classify* and *1-Regression* are very similar, but the *M-Regression* approach performance is always worse. Such results indicate that: (i) the 50 features considered are informative for both EE-PR and EE-ST problems, and allow to effectively select solvers; (ii) classification and regression predictive models have similar performance when used for selecting a single solver to run; and (iii) the regression predictive model tends to underestimate the CPU-time needed by algorithms for solving the considered problem on the given AF.

Table 8 shows the number of times each *basic solver* has been executed by either the *Classify* or the *M-Regression* portfolio. *1-Regression* executed solvers are not shown, because they are a subset of the *M-regression* selections. Table 8 shows some remarkable differences in the algorithm selected by the classification and regression approaches, and also those included in the static portfolios. For instance, *Classify* never selects **ArgSemSAT**, while it is largely exploited by *M-regression*, and included in static portfolios generated for solving EE-PR problems. This is because **ArgSemSAT**, and a few other *basic solvers*, has rarely been the fastest: therefore the classification approach—which only focuses on the fastest solver—ignores its performance. On the contrary, solvers like **ArgTools** (EE-PR) and **ASPARTIX-D** (EE-ST) are usually the fastest, and are often selected by both *Classify* and *M-Regression* approaches.

Finally, by looking at Table 7, it can be noted that the largest performance improvement can be achieved when exploiting portfolio approaches for solving the problem of enumerating preferred extensions of an AF: the use of portfolio-based techniques allows to solve up to 10.6% more instances than the best *basic solver*, **Cegartix**. Such margin is reduced to 2.9% when solving the EE-ST problem. This is due to the higher computational complexity of the EE-PR problem, and to the higher complementarity between *basic solvers* able to handle the EE-PR problem.

7.3. Post-Hoc Analysis: Generalisation of Performance

To assess the ability of our portfolios on testing instances that are dissimilar from instances used for training we generated four different new training sets as follows: starting from the original training set composed by 800 AFs, we removed all the frameworks corresponding to one set at a time, and randomly oversampled frameworks from the remaining three sets—in order to have again approximately 800 frameworks for training. We then tested our portfolios on the complete testing set of 1,200 AFs, so that performance can be compared with those of portfolios

EE-PR								
	Barabasi-Albert		**Erdös-Rényi**		**StableM**		**Watts-Strogatz**	
System	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10
Classify	**78.9**	**1321.4**	**88.6**	**745.0**	74.4	1574.3	**89.5**	**677.8**
1-Regression	76.3	1479.0	63.0	2255.2	76.5	1453.9	83.0	1079.9
M-Regression	70.4	1828.4	67.3	2039.7	77.0	1434.7	79.6	1267.6
FDSS	69.1	1916.2	80.9	1245.5	**79.1**	**1341.9**	78.6	1380.0
Shared-2	73.2	1678.0	73.2	1678.0	74.2	1620.4	73.2	1678.0
Shared-3	69.4	1892.0	67.3	2007.9	69.5	1896.7	69.4	1892.0
Shared-4	65.7	2106.2	65.7	2101.1	65.7	2108.1	65.7	2103.9
Shared-5	63.3	2240.9	63.4	2235.8	63.3	2242.9	63.3	2242.9

EE-ST								
	Barabasi-Albert		**Erdös-Rényi**		**StableM**		**Watts-Strogatz**	
System	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10	Cov.	PAR10
1-Regression	88.6	756.9	92.6	508.7	**98.6**	**149.9**	81.6	1153.0
Classify	93.0	470.4	92.4	519.6	91.2	575.6	93.4	439.3
Shared-2	**97.7**	**262.3**	97.3	285.2	97.7	220.9	**97.7**	**262.3**
M-Regression	96.2	297.4	**96.4**	**282.2**	95.6	334.9	90.3	636.5
Shared-3	94.0	420.1	94.0	435.5	94.0	420.1	94.0	476.6
FDSS	89.4	677.4	87.1	829.7	89.4	677.4	88.7	714.7
Shared-4	85.9	878.2	86.0	887.5	86.0	873.8	86.8	833.8
Shared-5	86.3	867.4	86.3	870.8	86.3	862.3	84.3	973.4

Table 9: Coverage (Cov.) and PAR10 of the systems considered in this study on the complete testing set, when trained on a training set not containing *AF*s of that structure (leave-one-set-out scenario). Systems are ordered according to results shown in Table 7. Best results in bold.

trained on the original set (Table 7). This can be seen as a leave-one-out scenario. The results of such generalisation analysis are shown in Table 9.

Unsurprisingly, static portfolios—particularly *Shared-k*—show the best generalisation performance: their behaviour does not change much with the new training sets. On the other hand, per-instance approaches do not show good generalisation capabilities: their performance varies significantly when the training set is not fully representative of the testing instances. This is true for both EE-PR and EE-ST problems, despite the fact that gaps are smaller in the EE-ST case, although it is true that also the performance of *basic solvers* on EE-ST tends to be closer.

Remarkably, *Classify* (covering up to 89.7%, cf. Table 7) is very sensible to the absence of Barabasi-Albert (-10.8%, cf. Table 9) or StableM (-15.3%, cf. Table 9) frameworks from the training set for EE-PR, while regression-based approaches show scarse generalisation abilities when the Erdös-Rényi frameworks are removed from the training set. On the contrary, *Classify* is very generalisable on the EE-ST set, and the *1-Regression* method is very sensitive when Watts-Strogatz *AF*s are removed. *M-Regression* is more generalisable than *1-Regression* when dealing with the EE-ST problem: this indicates that when testing instances are dissimilar from training ones, the exploitation of more than one solver can be fruitful.

8. Conclusion

We evaluated the impact of configuration of abstract argumentation solvers both at the level of *AF*-solver configuration, and at the level of combining solvers into portfolios, thus testing six experimental hypotheses.

We, indeed, extended the evaluation first proposed in [8] of an approach for the joint automatic configuration of *AF* descriptions and argumentation solvers. Specifically, we designed a method to automatically order the list of arguments and the list of attacks in argumentation frameworks by tuning 11 parameters, considering as a test-case the widely used Aspartix format. We focused our investigation on **ArgSemSAT**—runner-up of the ICCMA 2015—using Glucose as a SAT solver: they export together a further set of 20 parameters.

We demonstrate: (i) that joint AF-solver configuration has a statistically significant impact on the performance of **ArgSemSAT**; (ii) the synergies between *AF*s configuration and SAT solvers behaviour. We also open new, exciting possibilities in the area of learning for improving performance of abstract argumenta-

tion solvers. We believe this work would be particularly beneficial for the participants of the forthcoming competition ICCMA 2017 [40].[9]

We also exploit the ICCMA-15 legacy by combining state-of-the-art solvers, able to handle EE-PR and EE-ST problems, using portfolio-based techniques. In particular, we tested static and per-instance portfolios, exploiting the largest available set of argumentation features [18].

The results of our extensive empirical analysis showed that: (i) the claim that reduction-based solvers always outperform non reduction-based systems—a possible superficial reading of the results of ICCMA-15—is not always the case; (ii) the solvers at the state of the art show a high level of complementarity (specially those able to deal with EE-PR problems), thus they are suitable to be combined in portfolios; (iii) portfolio systems generally outperform *basic solvers*; (iv) if the training instances are representative of testing *AF*s, the existing set of features is informative for selecting most suitable solvers; (v) classification-based portfolios show good generalisation performance; (vi) static portfolios are usually the approaches which are less sensitive to different training sets.

We see several avenues for future work. We plan to evaluate the proposed joint *AF*-solver configuration approach on different solvers and on different semantics. Moreover, we are interested in exploiting the configuration approach for combining different argumentation and SAT solvers into portfolios. Finally, we are considering investigating the presence of *AF* configurations that are able to improve—on average—the performance of all the existing state-of-the-art argumentation solvers. This would provide powerful guidelines for the encoding of frameworks as well as explaining the connections between *AF* configurations and overall performance. We are also interested in further investigating the generalisation capabilities of portfolios performance by considering significantly differently-structured *AF*s, including complex frameworks generated by real-world scenarios. We will also extend the portfolio methods considering SATZilla [15] like approaches, or more sophisticated model-based techniques. Finally, given the significant results obtained for the extension enumeration problem, a natural future work is to test portfolio methods also in other argumentation problems, e.g. credulous and skeptical acceptance of a single argument.

[9] `http://www.dbai.tuwien.ac.at/iccma17/`

Acknowledgements

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

This work was performed using the computational facilities of the Advanced Research Computing @ Cardiff (ARCCA) Division, Cardiff University.

References

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

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

[3] F. Cerutti, M. Giacomin, M. Vallati, ArgSemSAT: Solving Argumentation Problems Using SAT, in: Proc. of COMMA, 2014, pp. 455–456.

[4] J. Lagniez, E. Lonca, J. Mailly, Coquiaas: A constraint-based quick abstract argumentation solver, in: Proc. of ICTAI, 2015, pp. 928–935.

[5] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stützle, Paramils: An automatic algorithm configuration framework, Jouranl of Artificial Intelligence Research 36 (2009) 267–306.

[6] C. Ansótegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: Proc. of CP, 2009, pp. 142–157.

[7] Z. Yuan, T. Stützle, M. Birattari, Mads/f-race: Mesh adaptive direct search meets f-race, in: Proc. of IEA/AIE, 2010, pp. 41–50.

[8] F. Cerutti, M. Vallati, M. Giacomin, On the Effectiveness of Automated Configuration in Abstract Argumentation Reasoning, in: Proc. of COMMA, IOS Press, 2016, pp. 199–206.

[9] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proc. of LION, 2011, pp. 507–523.

[10] F. Hutter, L. Xu, H. H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: Methods & evaluation, Artificial Intelligence 206 (2014) 79–111.

[11] M. Vallati, F. Hutter, L. Chrpa, T. L. McCluskey, On the effective configuration of planning domain models, in: Proc. of IJCAI, 2015.

[12] G. Charwat, W. Dvořák, S. A. Gaggl, J. P. Wallner, S. Woltran, Methods for solving reasoning problems in abstract argumentation - a survey, Artificial Intelligence 220 (2015) 28–63.

[13] F. Cerutti, M. Vallati, M. Giacomin, Where Are We Now? State of the Art and Future Trends of Solvers for Hard Argumentation Problems, in: Proc. of COMMA, IOS Press, 2016, pp. 207–218.

[14] M. Vallati, L. Chrpa, D. E. Kitchin, Portfolio-based planning: State of the art, common practice and open challenges, AI Communications 28 (4) (2015) 717–733.

[15] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: portfolio-based algorithm selection for sat, Journal of Artificial Intelligence Research (2008) 565–606.

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

[17] J. R. Rice, The algorithm selection problem, Advances in Computers 15 (1976) 65–118.

[18] F. Cerutti, M. Giacomin, M. Vallati, Algorithm selection for preferred extensions enumeration, in: Proc. of COMMA, 2014, pp. 221–232.

[19] R. Brochenin, T. Linsbichler, M. Maratea, J. P. Wallner, S. Woltran, Proc. of TAFA workshop, 2015, Ch. Abstract Solvers for Dung's Argumentation Frameworks, pp. 40–58.

[20] P. Baroni, F. Cerutti, P. E. Dunne, M. Giacomin, Automata for Infinite Argumentation Structures, Artificial Intelligence 203 (0) (2013) 104–150.

[21] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[22] U. Egly, S. A. Gaggl, S. Woltran, Aspartix: Implementing argumentation frameworks using answer-set programming, in: Logic Programming, 2008, pp. 734–738.

[23] F. Cerutti, M. Vallati, M. Giacomin, Argsemsat-1.0: Exploiting sat solvers in abstract argumentation, System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA) (2015) 4.

[24] G. Audemard, L. Simon, Lazy clause exchange policy for parallel sat solvers, in: Proc. of SAT, 2014, pp. 197–205.

[25] A. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 11.

[26] P. Erdös, A. Rényi, On random graphs. I, Publ. Math-Debrecen 6 (1959) 290–297.

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

[28] F. Cerutti, M. Giacomin, M. Vallati, Generating challenging benchmark AFs, in: Proc. of COMMA, 2014, pp. 457–458.

[29] F. Cerutti, N. Oren, H. Strass, M. Thimm, M. Vallati, A benchmark framework for a computational argumentation competition, in: Proc. of COMMA, 2014, pp. 459–460.

[30] M. Vallati, L. Chrpa, M. Grzes, T. L. McCluskey, M. Roberts, S. Sanner, The 2014 international planning competition: Progress and trends, AI Magazine 36 (3) (2015) 90–98.

[31] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (6) (1945) 80–83.

[32] F. Hutter, H. Hoos, K. Leyton-Brown, An efficient approach for assessing hyperparameter importance, in: Proc. of ICML, 2014, pp. 754–762.

[33] F. Cerutti, P. E. Dunne, M. Giacomin, M. Vallati, Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach, in: Proc. of TAFA workshop, 2014, pp. 176–193.

[34] N. Eén, N. Sörensson, An extensible sat-solver, in: Proc. of SAT, 2003, pp. 502–518.

[35] M. Thimm, S. Villata, F. Cerutti, N. Oren, H. Strass, M. Vallati, Summary Report of The First International Competition on Computational Models of Argumentation, AI Magazine 37 (1) (2016) 102.

[36] M. Thimm, S. Villata, System descriptions of the first international competition on computational models of argumentation (ICCMA), arXiv preprint arXiv:1510.05373.

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

[38] J. Seipp, M. Braun, J. Garimort, M. Helmert, Learning portfolios of automatically tuned planners, in: Proc. of ICAPS, 2012, pp. 369–372.

[39] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The WEKA data mining software: An update, SIGKDD Exploration 11 (1) (2009) 10–18.

[40] S. A. Gaggl, T. Linsbichler, M. Maratea, S. Woltran, Introducing the second international competition on computational models of argumentation, in: Proc. of (SAFA) workshop, 2016, pp. 4–9.

[41] M. Caminada, On the Issue of Reinstatement in Argumentation, in: Proc. of JELIA, 2006, pp. 111–123.

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

[43] M. Caminada, D. M. Gabbay, A Logical Account of Formal Argumentation, Studia Logica (Special issue: new ideas in argumentation theory) 93 (2–3) (2009) 109–145.

Appendix A. Configuration Parameters for Argumentation Frameworks

Parameter	Domain	Default
args_ingoingFirst	[-1.0,1.0]	0
args_outgoingFirst	[-1.0,1.0]	0.2
args_autoFirst	[-1.0,1.0]	-1
args_eachOther	[-1.0,1.0]	-1
args_differenceFirst	[-1.0,1.0]	-1
atts_ingoingFirst	[-1.0,1.0]	0
atts_outgoingFirst	[-1.0,1.0]	0
atts_autoFirst	[-1.0,1.0]	0.2
atts_eachOther	[-1.0,1.0]	0
atts_differenceFirst	[-1.0,1.0]	0
atts_orders	{0,1,2,3,4,5}	0

Regarding atts_orders:

0: Same ordering applied to the first argument of the attack pair;

1: Same ordering applied to the second argument of the attack pair;

2: Inverse ordering applied to the first argument of the attack pair;

3: Inverse ordering applied to the second argument of the attack pair;

4: Attack-specific ordering applied to the first argument of the attack pair;

5: Attack-specific ordering applied to the second argument of the attack pair;

Appendix B. Configuration Parameters for ArgSemSAT

Parameter	Domain	Default
SOLVER-ExtEnc	{001111, 010101, 010111, 011101, 011111, 101010, 101011, 101110, 101111, 110011, 110101, 110111, 111010, 111011, 111100, 111101, 111110, 111111}	101010
GLUCOSE-gc-frac	[0.0, 500.0]	0.2
GLUCOSE-rnd-freq	[0.0, 1.0]	0.0
GLUCOSE-cla-decay	[0.0, 1.0]	0.999
GLUCOSE-max-var-decay	[0.0, 1.0]	0.95
GLUCOSE-var-decay	[0.0, 1.0]	0.8
GLUCOSE-phase-saving	0,1,2	2
GLUCOSE-ccmin-mode	0,1,2	2
GLUCOSE-K	[0.0, 1.0]	0.8
GLUCOSE-R	[1.0, 5.0]	1.4
GLUCOSE-szTrailQueue	[10,10000] (int)	5000
GLUCOSE-szLBDQueue	[10,10000] (int)	50
GLUCOSE-simp-gc-frac	[0.0, 5000.0]	0.5
GLUCOSE-sub-lim	[-1,10000] (int)	20
GLUCOSE-cl-lim	[-1,10000] (int)	1000
GLUCOSE-grow	[-10000,10000] (int)	0
GLUCOSE-incReduceDB	[0,10000] (int)	300
GLUCOSE-firstReduceDB	[0,10000] (int)	2000
GLUCOSE-specialIncReduceDB	[0,10000] (int)	1000
GLUCOSE-minLBDFrozenClause	[0,10000] (int)	30

For a detailed description of GLUCOSE parameters, the interested reader is referred to [24], and the GLUCOSE's website.[10]

To explain the parameter SOLVER-ExtEnc, let us recall that each extension S implicitly defines a three-valued *labelling* of arguments, as follows: an argument **a** is labelled `in` iff $\mathbf{a} \in S$, is labelled `out` iff $\exists\ \mathbf{b} \in S$ s.t. $\mathbf{b} \rightarrow \mathbf{a}$, is labelled `undec` if neither of the above conditions holds. In the light of this correspondence, argumentation semantics can equivalently be defined in terms of labellings

[10] `http://www.labri.fr/perso/lsimon/glucose/`

rather than of extensions (see [41, 42]). In particular, the notion of *complete labelling* [43, 42] provides an equivalent characterization of complete semantics, in the sense that each complete labelling corresponds to a complete extension and vice versa. Complete labellings can be (redundantly) defined as follows.

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

- $\mathscr{L}ab(\boldsymbol{a}) = \mathtt{in} \Leftrightarrow \forall \boldsymbol{b} \in \boldsymbol{a}^- \mathscr{L}ab(\boldsymbol{b}) = \mathtt{out}$;
- $\mathscr{L}ab(\boldsymbol{a}) = \mathtt{out} \Leftrightarrow \exists \boldsymbol{b} \in \boldsymbol{a}^- : \mathscr{L}ab(\boldsymbol{b}) = \mathtt{in}$;
- $\mathscr{L}ab(\boldsymbol{a}) = \mathtt{undec} \Leftrightarrow \forall \boldsymbol{b} \in \boldsymbol{a}^- \mathscr{L}ab(\boldsymbol{b}) \neq \mathtt{in} \wedge \exists \boldsymbol{c} \in \boldsymbol{a}^- : \mathscr{L}ab(\boldsymbol{c}) = \mathtt{undec}$;

In [33] it is proved that the requirement of Definition 4 can equivalently be expressed in different ways, corresponding to the conjunction of different specific subsets of the 6 terms $C^{\rightarrow}_{\mathtt{in}}$, $C^{\leftarrow}_{\mathtt{in}}$, $C^{\rightarrow}_{\mathtt{out}}$, $C^{\leftarrow}_{\mathtt{out}}$, $C^{\rightarrow}_{\mathtt{undec}}$, $C^{\leftarrow}_{\mathtt{undec}}$, where

- $C^{\rightarrow}_{\mathtt{in}} \equiv (\mathscr{L}ab(\mathbf{a}) = \mathtt{in} \Rightarrow \forall \mathbf{b} \in \mathbf{a}^- \mathscr{L}ab(\mathbf{b}) = \mathtt{out})$;
- $C^{\leftarrow}_{\mathtt{in}} \equiv (\mathscr{L}ab(\mathbf{a}) = \mathtt{in} \Leftarrow \forall \mathbf{b} \in \mathbf{a}^- \mathscr{L}ab(\mathbf{b}) = \mathtt{out})$;
- $C^{\rightarrow}_{\mathtt{out}} \equiv (\mathscr{L}ab(\mathbf{a}) = \mathtt{out} \Rightarrow \exists \mathbf{b} \in \mathbf{a}^- : \mathscr{L}ab(\mathbf{b}) = \mathtt{in})$;
- $C^{\leftarrow}_{\mathtt{out}} \equiv (\mathscr{L}ab(\mathbf{a}) = \mathtt{out} \Leftarrow \exists \mathbf{b} \in \mathbf{a}^- : \mathscr{L}ab(\mathbf{b}) = \mathtt{in})$;
- $C^{\rightarrow}_{\mathtt{undec}} \equiv (\mathscr{L}ab(\mathbf{a}) = \mathtt{undec} \Rightarrow \forall \mathbf{b} \in \mathbf{a}^- \mathscr{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^- : \mathscr{L}ab(\mathbf{c}) = \mathtt{undec})$;
- $C^{\leftarrow}_{\mathtt{undec}} \equiv (\mathscr{L}ab(\mathbf{a}) = \mathtt{undec} \Leftarrow \forall \mathbf{b} \in \mathbf{a}^- \mathscr{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^- : \mathscr{L}ab(\mathbf{c}) = \mathtt{undec})$.

The parameter SOLVER-ExtEnc considers a sequence of six binary numbers whose position is in one-to-one correspondence with the sequence: $C^{\rightarrow}_{\mathtt{in}}$, $C^{\leftarrow}_{\mathtt{in}}$, $C^{\rightarrow}_{\mathtt{out}}$, $C^{\leftarrow}_{\mathtt{out}}$, $C^{\rightarrow}_{\mathtt{undec}}$, $C^{\leftarrow}_{\mathtt{undec}}$. For instance, SOLVER-ExtEnc=101010 is equivalent to select the encoding $C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$.