On Structural Properties of Argumentation Frameworks: Lessons from ICCMA

O. Rodrigues [a,1], E. Black [a], M. Luck [a], J. Murphy [a]
[a] *Department of Informatics, King's College London, London, UK*

Abstract. It is well known that the computation of solutions to decision and enumeration problems in argumentation can be very hard. In this work, we analyse some of the results of the 2017 International Competition on Computational Models of Argumentation. Our analysis shifts the focus from the performance of individual solvers to how well/badly they can collectively tackle different classes of abstract argumentation frameworks. In so doing, we were able to identify the instances that were particularly difficult for all/most solvers and look into their particular structural properties.

Keywords. argumentation semantics, argumentation competition, data analysis

1. Introduction

It is well known that the computation of argument acceptability and the enumeration of extensions of abstract argumentation frameworks (AAFs) can be very hard [9,11]. Nevertheless, advances in techniques for the computation of argumentation semantics [2,8,7,14,16] have been matched by the development of software systems (solvers). The International Competition on Computational Models of Argumentation (ICCMA, `http://argumentationcompetition.org`) provides a platform for the evaluation of the performance of the solvers against a collection of benchmark problems [18]. These problems consist of: the *enumeration* of either one or all of the extensions of an AAF; and the *decision* of whether a given argument is a member of one (resp. all) extensions of the AAF under a particular semantics (a *track* in the competition's terminology).

In order to evaluate the solvers against a range of problems, ICCMA uses AAFs taken from a number of *domains* with particular structural properties. The results of the 2nd ICCMA were announced late in 2017 and ranked the solvers in the different tracks of the competition with an overall winner. It is clear from the results that different problems present different levels of challenges for the solvers: for example, the AAF `massachusetts_vineyardfastferry_2015-11-13.gml.50` had all of its complete extensions enumerated by 14 solvers in an average time of 0.004s while the extensions of the AAF `WS_300_32_50_70` could only be enumerated by 2 solvers in an average time of 514.91s. In this paper, we shift the focus of the analysis of the results from the solvers themselves to the domains and the AAFs within them. In particular, we investigate structural properties of the AAFs and discuss the results of the complete enumeration track of

[1]Corresponding Author. E-mail: odinaldo.rodrigues@kcl.ac.uk

the 2nd ICCMA, taking each individual domain into account. Using this information, we aim to provide answers to the following questions: 1) In which domains are the AAFs easier or more difficult to solve? 2) Are there any domains that are beyond the current capabilities of all solvers? 3) Is there any correlation between some structural properties of an AAF and the ability and speed of the solvers to solve it?

Although our analysis is restricted to the complete enumeration track, it reveals some very interesting patterns in the results. Nearly 23% of all AAFs could not be enumerated by any solver. In the domain SemBuster, 15 out of the 16 AAFs (93.75%) could not be enumerated at all. The next most challenging domains were Planning2AF, Traffic and Barabasi-Albert, in this order. At the other end of the spectrum, the domains SccGenerator and GroundedGenerator were the easiest to tackle, having only a small proportion of AAFs whose extensions could not be enumerated by *all* solvers. These results not only confirm the initial findings in [3] about the sensitivity of argumentation tools to particular graph models, but also provide a proxy for the complexity of the models in general. In addition, our analysis seeks characteristics in the structure of the AAFs likely to make them more challenging.

The rest of the paper is organised as follows. In Section 2, we describe the methodology we used in the evaluation of the results. This is followed in Section 3 by an analysis of the results by domain. In Section 4, we discuss the domains whose extensions were particularly hard to enumerate, and analyse their structural properties. In Section 5, we suggest an indicator that attempts to predict the enumeration complexity of an AAF based on its structural properties. In Section 6, we conclude pointing out some directions for future work.

2. Methodology

It is worth beginning by summarising how the competition was organised, the results of the benchmarking available, and how we used it in the analysis. The competition was divided into eight main tracks, one for each of the *grounded*, *complete*, *preferred*, *stable*, *semi-stable*, *stage* and *ideal* semantics, plus a special track called "Dung's Triathlon" requiring solvers to enumerate all extensions of all standard Dung-semantics (the first four aforementioned) [1,10]. A public call was made for submission of benchmark problems (i.e., AAFs) and 11 different *domains* were selected. For space limitations, we cannot describe each individual domain, but they are listed in Table 3. For more details please check `http://argumentationcompetition.org/2017/benchmark_selection_iccma2017.pdf`.

In total 350 AAFs were used in the complete enumeration track. Table 3 lists the domains, the number of AAFs taken from each domain, and an "identifier" that can be used to identify the domain of an AAF in this track. For example, the string "BA" can be used to identify that the AAF `BA_80_80_2` belongs to the Barabasi-Albert domain; the string "gml" can be used to infer that the AAF `arlington_va_2016-01.gml.20` belongs to the Traffic domain; and so forth.

Although the competition also included the partial enumeration of extensions as well as skeptical and credulous decision problems in the seven semantics mentioned above, in order to keep our analysis focussed and succinct, we have restricted it to the enumeration of all extensions of the complete semantics (i.e., the `EE-CO` track, in ICCMA's terminology). Each solver was awarded a *score* for the enumeration of all complete extensions of

Solver	**Total**	**ABA**	**Adm**	**BA**	**ER**	**GG**	**PL**	**Sem**	**SCC**	**ST**	**Tra**	**WS**
pyglaf	**238**	**32**	**13**	26	30	**24**	**18**	**1**	**29**	**21**	20	24
argmat-dvisat	221	**32**	7	21	27	**24**	**18**	0	**29**	**21**	15	27
argmat-mpg	198	**32**	7	**27**	30	**24**	**18**	**1**	27	5	**21**	6
cegartix	189	31	10	11	25	**24**	7	0	**29**	20	6	26
argmat-sat	187	**32**	7	8	27	**24**	7	0	**29**	**21**	6	26
ArgSemSAT	182	**32**	12	7	25	**24**	6	0	**29**	19	5	23
conarg	182	21	**13**	18	**35**	21	15	**1**	12	-8	18	**36**
heureka	178	**32**	12	22	30	**24**	**18**	**1**	12	3	20	4
CoQuiAAS	174	**32**	12	2	25	**24**	4	0	**29**	18	5	23
goDIAMOND	171	**32**	-5	3	33	18	5	0	**29**	**21**	5	30
ArgTools	157	**32**	9	10	32	23	6	**1**	19	8	5	12
EqArgSolver	124	**32**	7	8	8	**24**	5	0	28	2	6	4
argmat-clpb	20	7	0	5	0	1	0	**1**	0	0	5	1
gg-sts	-304*	-160	65	8	16	-102	-9	0*	15	**21**	-23	27

Table 1. Total scores of all solvers in each domain for the `EE-CO` track.

an AAF within the time limit of 600s. The score 1 was awarded for delivering the correct and complete set of extensions of the AAF within 600s; the score -5 was given for delivering an incorrect extension of the AAF within 600s; and the score 0 was given otherwise (this included results delivered after 600s; the abnormal termination of a solver; and the delivery of an incomplete set of extensions). The total scores by domain for the `EE-CO` track are shown in Table 1. The results of all problem instances for all solvers are available at `http://argumentationcompetition.org/2017/results.html`.

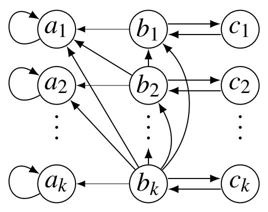
Using the above available data, we undertook a number of calculations to obtain the average enumeration time per AAF, the number of solvers successfully enumerating all extensions within 600s, and several other statistics grouping AAFs by domain. We also investigated several structural properties of the AAFs, such as their number of arguments and attacks, how many strongly connected components (SCCs) they had, the density of attacks of the AAF, etc., by analysing all AAFs ourselves.

Since all enumerations completed after 600s were awarded the score 0 regardless of whether they were correct or not, in the analysis of the average execution time and the number of solvers successfully completing an enumeration, we only considered those results whose scores were 1. We also examined instances that were not successfully completed by any solver in the time allowed, since they may point to structural properties of interest.

In Section 3, we consider the results from the perspective of each individual domain, but first describe some adjustments we made to the results to make our analysis clear.

Anomalies in the Results

Domain SemBuster. The AAFs in the SemBuster domain were proposed by Caminada and Verheij [6,5]. Their set of arguments is divided into three partitions A, B and C, each with a given cardinality k, arranged as in the picture below on the left.



For a given value of k, the AAF in this domain is unique and has exactly $n = 3k$ arguments. Using the SCC-recursive schema [2], these arguments can be arranged in $k+1$ layers: the top layer with b_k and c_k; followed by a_k, b_{k-1} and c_{k-1}; and so forth until the last layer with a_1 alone. The pattern of attacks is also very regular, so we can easily count them as follows.

Proposition 1. *An AAF in the SemBuster domain with* $3k$ *arguments has* k^2+3k *attacks.*

Proof. It is easy to see that for every $i > 1$ and every $j < i$, every b_i argument attacks every a_j and b_j argument. Therefore, for each row $1 < i \leq k$, we have $2 \times (i-1)$ attacks. Summing them all up gives us double the sum of the numbers in the sequence $1,2,\dots,k-1$. Furthermore, within each row a_i, b_i, c_i there are exactly 4 attacks. This gives us # attacks $= 2 \times (\sum_{i=1}^{k-1} i) + 4k = 2 \times \frac{(k-1)\times k}{2} + 4k = k^2 + 3k$. □

Thus, for $k = 20$, we have an AAF with 60 arguments and $20^2 + 60 = 460$ attacks arranged in 21 layers. Some properties of the AAFs in this domain are given in Table 2.

It was claimed in [6] that these AAFs would have $k+1$ complete extensions, but in fact, they have a much larger number t of complete extensions: $t = 2^k + k$.

Proposition 2. *A SemBuster AAF with* $3k$ *arguments has* $2^k + k$ *complete extensions.*

Proof. Note that no extension can contain an argument from the A partition, so all complete labellings label all a_i arguments either **out** or **und**. The number $2^k + k$ comes from the following observations: *i)* There are exactly k complete labellings with exactly one B argument labelled **in**: for each b_i that is labelled **in**, all arguments b_j such that $j \neq i$ must be labelled **out**, and, consequently all corresponding arguments c_j must be labelled **in**. In addition, for all $j \leq i$, a_j is labelled **out**, and for all $j > i$, a_j is labelled **und**; and *ii)* The remaining 2^k labellings give all complete extensions that are subsets E_C of the C partition (obviously, without a B element). They can be constructed as follows:

$$\lambda(a_i) = \textbf{und}, \text{ for } i \leq k \quad \lambda(b_i) = \begin{cases} \textbf{out}, \text{ if } c_i \in E_C \\ \textbf{und}, \text{ otherwise} \end{cases} \quad \lambda(c_i) = \begin{cases} \textbf{in}, \text{ if } c_i \in E_C \\ \textbf{und}, \text{ otherwise} \end{cases}$$ □

Note that case *ii)* in the proof above includes the empty extension (when $E_C = \varnothing$ and all arguments are labelled **und**). Incidentally, this means that not every complete labelling is also a preferred labelling in this domain (cf. [6]). In particular, all non-maximal subsets of C in case *ii)* above are *not* preferred.

AAFs of this type pose a challenge beyond the actual computation of the complete labellings since the number of complete extensions grows exponentially to the number of arguments. For example, the smallest SemBuster AAF in this domain, `sembuster_60`, has only 60 arguments but $1,048,596$ extensions. Thus, the amount of data generated in the solutions becomes problematic and, for a sufficiently large value of n, it cannot be enumerated in any reasonable time. In fact, this is not the only domain where a relatively low number of arguments generates a very large number of extensions. The number of complete extensions of the so-called *bi-directed cycle graphs* is even larger [16]. Whereas a SemBuster framework with 36 arguments only has $4,108$ complete extensions, a bi-directed cycle graph with 36 arguments has $7,095,675$.

The results of the competition for the SemBuster domain include the ones listed in Table 2 (recall, all results presented here are for the `EE-CO` track). The solver

AAF	**Args**	**Attacks**	**NT SCCs**	**Max SCC size**	**Layers**	**Solvers Awarded Score 1**	**Avg Time (secs)**
sembuster_60	60	460	40	2	21	argmat-clpb/mpg ArgTools, conarg heureka, pyglaf	13.08
sembuster_150	150	2650	100	2	51	-	-
sembuster_300	300	10300	200	2	101	-	-
sembuster_600	600	40600	400	2	201	-	-
sembuster_900	900	90900	600	2	301	-	-
sembuster_1200	1200	161200	800	2	401	-	-
sembuster_1500	1500	251500	1000	2	501	gg-sts	32.47
sembuster_1800	1800	361800	1200	2	601	gg-sts	30.29
sembuster_2400	2400	642400	1600	2	801	gg-sts	32.51
sembuster_3000	3000	1003000	2000	2	1001	gg-sts	33.58
sembuster_3600	3600	1443600	2400	2	1201	gg-sts	33.12
sembuster_4200	4200	1964200	2800	2	1401	gg-sts	33.52
sembuster_4800	4800	2564800	3200	2	1601	gg-sts	27.91
sembuster_5400	5400	3245400	3600	2	1801	gg-sts	29.77
sembuster_6000	6000	4006000	4000	2	2001	gg-sts	32.29
sembuster_7500	7500	6257500	5000	2	2501	-	-

Table 2. Details of the AAFs in the SemBuster domain.

gg-sts was awarded scored 1 for the instances 1500–6000, the smallest of which was sembuster_1500, with 1500 arguments and therefore $2^{500} + 500$ complete extensions. Clearly, these cannot have been computed and enumerated, so these results cannot be correct. We have contacted the organisers of the competition who, at the time of writing, are investigating. For the purposes of our analysis, we disregarded the results provided by gg-sts in the above instances.

Other results not fully verified. A number of other instances outside the SemBuster domain were also enumerated by only one solver. At the time of writing, our understanding is that although there was some scrutiny of the results provided by a single solver, they were not fully verified. Thus, there may well be other instances where a score 1 was incorrectly awarded. In our analysis, we only modified the results above since they are mathematically impossible in the time given. This means that the analysis presented in Section 4.2 may need to be adjusted should the results change. However, the overall methodology we have employed is justified with the data given. For a discussion on the difficulties of verifying the competition results, please see Section 6.

3. Challenging Domains

The EE-CO track evaluated the overall performance of the solvers over the whole set of 350 AAFs. It is interesting to look at how these results fare within each individual domain. Table 1 shows how each solver performed in each domain, with the best results highlighted in bold. The overall results are displayed in the column "Total". The results with a $*$ have been adjusted as described in Section 2. In the table, the term ABA is used for the domain ABA2AF, Adm for the domain AdmBuster, BA for the domain Barabasi-Albert, ER for the domain Erdös-Rényi, GG for the domain GroundedGenerator, PL for the domain Planning2AF, Sem for the domain SemBuster, SCC for the domain SccGenerator, ST for the domain StableGenerator, Tra for the domain Traffic, and WS for the domain Watts-Strogatz.

Table 1 shows that every solver obtained the highest score in at least one domain, confirming a similar finding in [4], which analysed a subset of the 1st ICCMA's solvers on a smaller number of domains.[2] If we shift focus from solver to domain, we can define a proxy for complexity by examining how the solvers performed overall in each domain. In particular, we consider how many of the domain instances could not be solved by any solver. Table 3 contains the average execution time of the successful enumerations as well as the percentage of instances in each domain left unsolved by all solvers (within 600s). Four domains had a particularly high proportion of unsolved instances. As expected from our analysis in Section 2, the domain SemBuster had nearly all instances left unsolved (93.75%). This was followed by the domain Planning2AF, which had 58.13% of its instances left unsolved; followed by the domain Traffic with 43.90% of instances; and then the domain Barabasi-Albert with 34.14% of instances left unsolved. Table 3 also shows the number of instances solved by at most one solver. In general, the proportion of problems that could be solved by only a few solvers was also very high.

Domain	**Identifier**	**AAFs**	**Unsolved (%)**		**≤ 1 solver**	**Avg time**
ABA2AF	`afinput`	32	0	(0%)	0	12.71s
AdmBuster	`admbuster`	13	0	(0%)	0	22.10s
Barabasi-Albert	`BA`	41	14	(34.15%)	15	59.40s
Erdös-Rényi	`ER`	40	0	(0%)	8	93.28s
GroundedGenerator	`grd`	24	0	(0%)	0	18.91s
Planning2AF	`.cnf`	43	25	(58.14%)	25	41.98s
SemBuster	`sembuster`	16	15	(93.75%)	15	13.09s
SccGenerator	`scc`	30	0	(0%)	1	35.54s
StableGenerator	`stb`	30	5	(16.67%)	9	93.86s
Traffic	`.gml.`	41	18	(43.90%)	21	41.29s
Watts-Strogatz	`WS`	40	1	(2.50%)	10	146.73s
Total		350	78		104	57.50s

Table 3. Distribution of "hard" problems by domain.

Figure 1 generalises this further. It provides an overview of the percentage of the AAFs left unsolved by a given number of solvers by domain. The x-axis in the plot represents the number of solvers that were unable to solve problems in a domain and the y-axis represents the percentage of problems left unsolved by those solvers. So for example, for $x = 16$, y gives the percentage of problems left unsolved by 16 solvers (i.e., all) in each domain (compare this with Table 3).

At the other end of the spectrum, the domains GroundedGenerator, SccGenerator and ABA2AF were the easiest to tackle, with only a small proportion of AAFs left unsolved by a few solvers.

4. Challenging AAFs

The extensions of 78 out of the 350 AAFs (22.28%) in the `EE-CO` track could not be fully enumerated within 600s. In addition, the full enumeration of the extensions of 26 other AAFs could only be completed by a single solver. For the purposes of this analysis,

[2]The domains considered in [4] were Barabasi-Albert, Erdös-Rényi, Kleinberg [13], and plain tree-graphs.

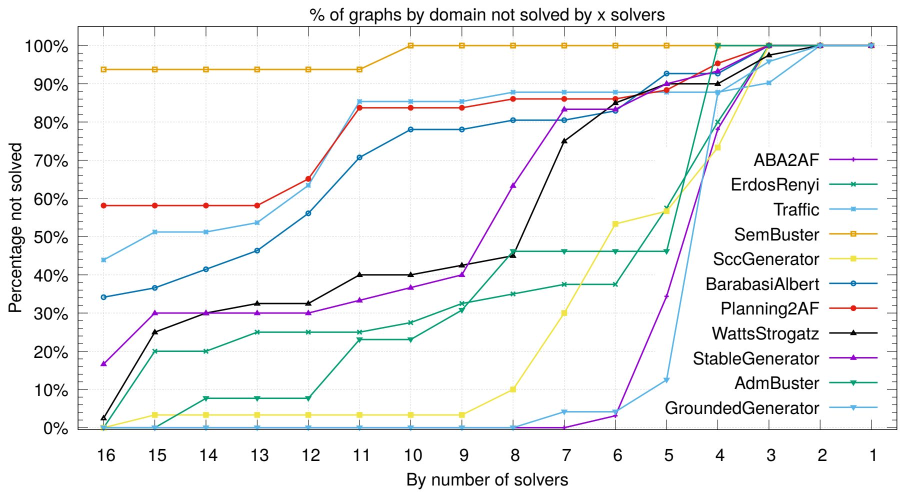


Figure 1. Percentage of instances left unsolved by number of solvers and domain

we refer to these particularly challenging 104 AAFs as the "hard" AAFs. Together they represent 29.71% of the total number of AAFs in the `EE-CO` track.

Our objective in the identification of the hard AAFs is to shed some light into specific structural properties that may make them difficult to solve. In addition, in future work this may help to draw some correlation between the techniques that were used by the solvers managing to solve them and the AAFs' structural properties. The tables in Appendix A give further structural details about the challenging AAFs in some of the hardest domains.

4.1. AAFs Not Solvable By Any Solver

We have already concluded from the anomalies of the SemBuster domain presented in Section 2, that the difficulty in the enumeration of the complete extensions of an AAF can arise not only because of the complexity of the attack configuration of the framework, but also from the number of extensions itself. Besides SemBuster, the domain Planning2AF was clearly the hardest to tackle, containing 32.05% of all unsolvable AAFs (58.14% of the AAFs in the domain); followed by the Traffic domain with 23.07% of all unsolvable AAFs (43.90% of the domain), and then by the Barabasi-Albert domain with 17.94% of all unsolvable AAFs (34.15% of the domain). We discuss some structural properties of the AAFs in these domains in Section 4.3. Details of some of the individual AAFs can be found in Appendix A.

Looking at the structure of the AAFs in the SemBuster domain, we can see that even a configuration with a small number of small non-trivial SCCs arranged in a particular way can hinder the ability of the solvers to tackle them. In the case of the SemBuster AAFs, the k binary non-trivial SCCs (containing the loop b_i–c_i) accept at least two possible solutions one labelling c_i **in** and the other labelling it **out** or **und**. These possibilities need to be propagated down to the other layers and this is essentially what increases the number of solutions exponentially to the number of layers in the AAF, giving the $O(2^k)$ number of complete extensions. In consequence, we should expect that AAFs with many non-trivial SCCs arranged in many layers are also likely to be hard to enumerate, but this is not obviously the only indicator of complexity. A single large non-trivial SCC or a relatively small non-trivial SCC with a sparse distribution of attacks (such as the bi-directed acyclic AAFs described in Section 2) are also problematic. In Section 5, we attempt to combine these factors into a single metric to obtain a proxy for complexity.

4.2. AAFs solvable by at most one solver

The extensions of 26 AAFs could only be enumerated by one solver. This represents 7.42% of the total number of AAFs in the `EE-CO` track.

The solver `conarg` was able to solve 76.92% (20 out of 26) of the problems solved by only one solver, spending around 597s in each enumeration. Here, `conarg`'s strategy in the competition was to provide whatever extensions it managed to compute within the time limit.[3] This explains why, in each of these 20 instances, it completed just before 600s. There were also 3 instances solved only by the solver `gg-sts`, all in under 41s; one instance solved only by `goDIAMOND` (in 468.6s); and 2 instances solved only by `argmat-mpg` – one of which, namely the AAF `massachusetts_srta_2014-11-13.gml.50`, we will come back to in the discussion in Section 6.

Since the answers provided by a single solver may not have been fully verified (especially for *completeness*), it is possible that these results need to be adjusted further. However, if it turns out that all of these instances are indeed correct *and complete*, it would be interesting to investigate further the superior effectiveness of these solvers in these demanding AAFs.

4.3. Structural Properties of the Hard Domains

We already discussed the domain SemBuster in Section 2. In this section, we look at the next three hardest domains in more detail. For brevity, we only summarise the most important characteristics of their AAFs. Further details can be found in Appendix A.

Domain Planning2AF The domain Planning2AF was the second hardest domain to tackle. One of the most distinguishing characteristics of the AAFs here is that their non-trivial SCCs are arranged in a relatively high number of layers (35 layers per AAF on average). In the cases containing the three lowest number of layers (3, 7 and 11, resp.), the AAFs had non-trivial SCCs with the largest sizes (421, 219 and 245, resp.). So there is a pattern of large SCCs or smaller SCCs arranged in a large number of layers.

Domain Traffic The Traffic domain was composed of 41 AAFs created from mass transit data, and the configuration of SCCs here was very variable. Some of the unsolvable AAFs had very large SCCs or a very large number of smaller non-trivial SCCs.[4] On average, the AAFs had 100 non-trivial SCCs, with an average size of 8 arguments, arranged over 8 layers on average. Three AAFs were solved by at most one solver: `massachusetts_srta_2014-11-13.gml.50` (by `argmat-mpg`); `longueuil_qc_2016-01.gml.20` (by `gg-sts`); and `orange-county-transportation-authority_20151202_1534.gml.20` (by `gg-sts`). The AAF `massachusetts_srta_2014-11-13.gml.50` has

[3]Personal communication with S. Bistarelli, 8/6/18.

[4]For instance, the largest SCC in the AAF `ruter.no_20151217_2005.gml.80` had 11,459 nodes, and the AAF `ecobici.stats_20141007_2248.gml.50` had 680 non-trivial SCCs.

34,307,712 complete extensions (generating roughly 4.3Gb of data), so representing and verifying sets of this cardinality can be challenging (see discussion in Section 6).

Domain Barabasi-Albert Finally, the AAFs in the Barabasi-Albert domain were relatively smaller than the ones in the two previous domains. They had on average 13 non-trivial SCCs with an average of 8 nodes arranged over 5 layers (on average).

Other Noteworthy Results The solver `conarg` did very well in the domains Watts-Strogatz and StableGenerator. It was the only solver capable of solving 13 instances in these domains with a typical execution time of roughly 597s. Assuming these results are correct, this points to `conarg`'s special ability of dealing with large SCCs. The domain Watts-Strogatz is composed of AAFs with a single non-trivial SCC with between 300 and 500 nodes. The AAFs in the StableGenerator domain contained up to 2 non-trivial SCCs of sizes of up to 723 nodes and a few trivial SCCs, all arranged in up to 5 layers.

5. Challenging Structural Properties

Here, we attempt to devise a measure based on structural properties of the AAFs that gives a good indication of the likely complexity of the enumeration of their extensions.

Solvers that compute extensions by decomposing the original argumentation framework into SCCs [2,12] work on each SCC individually and then combine the solutions appropriately (see `http://argumentationcompetition.org/2017/submissions.html`), e.g., `EqArgSolver`, `argmat-dvisat`. Simply put, a *trivial* SCC is a single argument that is not involved in any loops and a *non-trivial* SCC is a maximal set of arguments in which every argument in the set is reachable from every argument in the set via the attack relation.

A complete extension can be seen as the colouring of a direct graph satisfying some constraints: *i)* A node can be coloured **in** if and only if all of its attackers are coloured **out**;[5]; *ii)* If any of the attackers of a node is coloured **in**, then the node must be coloured **out**; and *iii)* a node is coloured **und** if and only if one of its attackers is coloured **und** and no attacker is coloured **in**. A *solution* is then any colouring of all nodes satisfying these constraints. The solution of a trivial SCC is uniquely determined by the colours given to its attackers, and once these are known this can be easily computed. However, in a non-trivial SCC there may be many configurations that satisfy the constraints *i)–iii)*. Each solution to a non-trivial SCC needs to be propagated to the nodes they attack, so the number of configurations to check can grow very quickly.

In any evaluation of complexity, we must differentiate the trivial components of the AAF from its non-trivial components. Each trivial SCC T only adds a very small amount to the complexity of the AAF, except in cases where the trivial SCCs form a very long chain. For this we can take the length of the chain up to T into account.[6] So we can define the individual complexity of solving a trivial SCC T, $tc(T)$, as $tc(T) = depth(T) * \varepsilon$, where $depth(T)$ is the length of the longest chain of attacks leading to T in its layer and ε is a small constant. In our calculations, we used $\varepsilon = 5 \times 10^{-10}$.

For a non-trivial SCC N, we must consider a number of factors. Experiments undertaken in [17] suggest that the complexity of the solution of an AAF $G = \langle Args, Atts \rangle$

[5]Note that this includes the non-attacked nodes, vacuously.

[6]For example, the longest chain in the problem `admbuster_2000000` is $1,000,000$ nodes deep.

is inversely proportional to its density of attacks. The density of G is defined as $density(G) = \frac{|Atts|}{|Args|^2}$, and so its *sparsity* is $sparsity(G) = 1 - density(G)$. However, as we intend to use this measure on a non-trivial SCC N with nodes $nodes(N) \subseteq Args$, we must only consider the number of internal attacks starting and terminating in N, i.e., $atts(N) = nodes(N)^2 \cap Atts$. Furthermore, given that the labels of the nodes in N with an external attacker *may* be *at least* partially determined by the label of the attacker, we are more interested in N's "loose" nodes, i.e., those without an external attacker: $loose(N)$. So for a non-trivial SCC N, the measure that interests us is $sparsity(N) = 1 - \frac{|loose(N)|}{|atts(N)|^2}$.

Taking these factors into account, by trial and error, we defined the following measure for the complexity of a non-trivial SCC N, $nt\_c(N) = (\sqrt{2})^{(sparsity(N)*\sqrt{loose(N)})}$.

Our aim with the above measure is to give the chosen structural aspects of a non-trivial SCC the correct weight in the approximation of the overall complexity of the AAF.

Now let $t\_sccs(G)$ be the set of all trivial SCCs of G and $nt\_sccs(G)$ its set of non-trivial SCCs. We define $m_x(G)$ as

$$m_x(G) = \Sigma_{T \in t\_sccs(G)} t\_c(T) + \Sigma_{N \in nt\_sccs(G)} nt\_c(N)$$

Note that m_x has two distinct components for the complexities of the trivial and non-trivial SCCs of an AAF G, with the non-trivial part weighing much more heavily. In addition,

(M1) If $t\_sccs(G) = \varnothing$, then $m_x(G) = \Sigma_{N \in nt\_sccs(G)} nt\_c(N)$ and if $nt\_sccs(G) = \varnothing$, then $m_x(G) = \Sigma_{T \in t\_sccs(G)} t\_c(T)$.
This means that a higher number of arguments in an AAF G itself is not sufficient to dramatically increase the value of $m_x(G)$. This is important in AAFs in domains such as AdmBuster, which have a very high number of nodes (up to 2,000,000), but do not have any non-trivial SCCs, and are therefore relatively simple to solve.

(M2) The higher the incidence of non-trivial SCCs in an AAF G, the higher the value of $m_x(G)$.

(M3) The less sparse a non-trivial SCC is, the lower its contribution to the value of $m_x(G)$.

Table 4 shows the Pearson product-moment correlation coefficients between m_x or the density or the sparsity of the AAFs and the number of solvers to return a correct solution. A value of ~ -0.5 or more indicates a negative correlation between the aspects and ~ -0.75 or more indicates a strong negative correlation [15].

We observe that for five domains (Erdös-Rényi, SccGenerator, StableGenerator, AdmBuster and Barabasi-Albert) there is a fair correlation between m_x and the number of solvers returning a correct solution. This suggests that m_x is a good indicator of the difficulty of a particular framework in these domains. On the other hand, the density/sparsity of the AAFs was a good indicator for their complexity in the ABA2AF, SemBuster, Traffic and Planning2AF domains. There was no strong positive or negative correlation in the domains GroundedGenerator and Watts-Strogatz. This needs further investigation.

Currently, m_x is just an *initial* attempt at capturing the complexity of an AAF based on a few of its graph theoretical properties. By incorporating more properties into the measurement, and fine-tuning how each individual component is weighted, it may be possible to obtain stronger correlations over a larger number of domains. Ideally, we would have a measurement that strongly indicates the difficulty of the framework across most domains. However, even if it is possible to theoretically derive such a measurement, it may be unfeasible to compute it in a satisfactory time. Moreover, the configuration of attacks in certain domains may be *designed* to have a particular effect on its number of extensions and/or the complexity of the enumeration, so no measure can be expected to work well in all domains.

Domain	Measure	Correlation
Erdös-Rényi	m_x	-0.63
SccGenerator	m_x	-0.79
StableGenerator	m_x	-0.66
AdmBuster	m_x	-0.74
Barabasi-Albert	m_x	-0.50
ABA2AF	density	-0.61
SemBuster	sparsity	-0.78
Traffic	sparsity	-0.82
Planning2AF	sparsity	-0.50

Table 4. Correlation between measures and the number of solvers that returned a correct solution per domain.

6. Conclusions and Discussion

It is quite clear from our analysis that the performance of the solvers in ICCMA varied quite dramatically according to the domain of the AAFs. In Section 3, we saw that some domains have instances whose extensions are particularly hard to enumerate, confirming and extending similar findings for a smaller number of domains initially reported in [3,4]. In addition, our analysis revealed several results, particularly in the cases that were solvable by only one solver, that deserve further investigation.

We saw that besides the intrinsic computational complexity associated with checking the constraints of a complete labelling, some domains can generate a very large number of extensions and this can also be problematic. Take, for example, the problem `massachusetts_srta_2014-11-13.gml.50` from the Traffic domain, which has 34,307,712 complete extensions. The solution set for this instance alone comprises over 4.3Gb of data. This can become a problem not only for individual solvers, but also for the benchmarking both in terms of storage as well as verification. In terms of verification, only problem instances whose complete set of solutions have been generated and verified should be employed in future competitions. This will eliminate any problems arising from the inability to verify results provided by a single solver. In the cases where the solutions cannot be generated automatically from the theoretical properties of the domain,[7] the solutions can be obtained by running specific solvers for a generous amount of time and then checking the solutions against each other and/or by running some sanity checks, such as counting the number of extensions, checking for admissibility properties, and so on. Those instances where the size of the solutions may become untractable or impractical should be discarded or dealt with separately.

A new competition track could be introduced whereby solvers are awarded points according to the number of extensions they correctly enumerate within the timeout period. The solvers in turn can adapt to display a solution as soon as it is becomes available.

Clearly, there is much more work to do. An important question is how realistic the domains used in the competition are in practical argumentative scenarios. It may well be

[7]See Section 2 for an explanation on how to do this in the SemBuster domain.

the case that certain configurations are not likely from the purely argumentative point of view, so developing argumentation solvers for all possible configurations of direct graphs may be unwise. Indeed, there are also other possible interesting decision problems to consider besides credulous and skeptical acceptance.

Finally, for space limitations, we only analysed the *complete* semantics. Given all complete extensions, it is possible to provide an answer for both skeptical and credulous decision problems under several different semantics, by analysing the elements in these extensions. However, the full enumeration of these extensions is not always required and the direct analysis of the results of the other competition tracks would also be useful.

Acknowledgements We would like to thank the organisers of the 2nd ICCMA for clarifications on the calculation of the competition results; Stefano Bistarelli for discussions about `conarg`; and the anonymous reviewers for their suggestions to improve this paper.

References

[1] P. Baroni, M. Caminada, and M. Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26:365–410, 12 2011.

[2] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168(1):162 – 210, 2005.

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

[4] S. Bistarelli, F. Rossi, and F. Santini. Not only size, but also shape counts: abstract argumentation solvers are benchmark-sensitive. *Journal of Logic and Computation*, 28(1):85–117, 2018.

[5] M. Caminada and B. Verheij. On the existence of semi-stable extensions. In *Proceedings of the 22nd Benelux Conference on Artificial Intelligence*, 2010.

[6] M. Caminada and B. Verheij. Sembuster: a benchmark example for semi-stable semantics. `http://argumentationcompetition.org/2017/SemBuster.pdf`, 2017.

[7] F. Cerutti, M. Vallati, and M. Giacomin. Where are we now? State of the art and future trends of solvers for hard argumentation problems. In P. Baroni, T.F. Gordon, and T. Scheffler, editors, *Proceedings of COMMA*, volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 207–218. IOS Press, 2016.

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

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

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

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

[12] B. Liao. *Efficient Computation of Argumentation Semantics*. Elsevier, 2014.

[13] C. Martel and V. Nguyen. Analyzing Kleinberg's (and other) small-world models. In *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, pages 179–188, New York, NY, USA, 2004. ACM.

[14] S. Nofal, K. Atkinson, and P. E. Dunne. Looking-ahead in backtracking algorithms for abstract argumentation. *International Journal of Approximate Reasoning*, 78:265 – 282, 2016.

[15] K. Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

[16] O. Rodrigues. A forward propagation algorithm for the computation of the semantics of argumentation frameworks. In *Theory and Applications of Formal Argumentation*, TAFA'17, pages 120–136. Springer International Publishing, 2018.

[17] O. Rodrigues. An investigation into reduction and direct approaches to the computation of argumentation semantics. In *Festschrift in Honour of Tarcisio Pequeno*, Tributes. College Publications, To appear.

[18] M. Thimm, S. Villata, F. Cerutti, N. Oren, H. Strass, and M. Vallati. Summary report of the first international competition on computational models of argumentation. *AI Magazine*, 37(1):102–104, 2016.

Appendix A. Some Structural Properties of the Hard Domains

Domain Planning2AF (all unsolved)

AAF	Args	Atts	NT SCCs	Avg nt SCC size	Max SCC size	Layers
`ferry2.pfile-L3-C2-02.pddl.2.cnf`	528	1012	51	3.29	26	43
`bw2.pfile-3-04.pddl.2.cnf`	958	1960	74	3.88	24	49
`bw2.pfile-3-06.pddl.2.cnf`	825	1680	58	2.86	39	53
`bw3.pfile-3-01.pddl.2.cnf`	1224	2479	60	4.55	93	51
`ferry2.pfile-L3-C4-01.pddl.3.cnf`	1180	2156	100	3.64	88	62
`bw2.pfile-3-06.pddl.6.cnf`	970	1768	73	4.3	6	43
`bw2.pfile-4-03.pddl.1.cnf`	908	1899	87	2.0	2	61
`bw3.pfile-3-01.pddl.1.cnf`	1644	3778	110	2.0	2	79
`bw3.pfile-3-02.pddl.5.cnf`	515	931	14	26.5	245	11
`bw3.pfile-3-08.pddl.6.cnf`	652	1159	52	4.54	10	39
`bw3.pfile-4-03.pddl.4.cnf`	2943	5610	212	2.0	2	117
`bw3.pfile-4-08.pddl.3.cnf`	2290	4343	125	4.11	136	85
`bw3.pfile-4-08.pddl.6.cnf`	1882	3521	107	4.02	10	85
`ferry2.pfile-L2-C4-07.pddl.1.cnf`	471	916	69	2.0	2	31
`ferry2.pfile-L3-C1-010.pddl.6.cnf`	348	594	39	4.05	6	21
`bw2.pfile-3-02.pddl.3.cnf`	282	461	53	2.45	10	19
`bw2.pfile-3-02.pddl.4.cnf`	247	391	59	2.0	2	25
`bw2.pfile-3-07.pddl.1.cnf`	265	493	42	2.0	2	31
`bw2.pfile-3-08.pddl.1.cnf`	266	495	42	2.0	2	31
`bw2.pfile-3-08.pddl.4.cnf`	247	391	59	2.0	2	25
`bw2.pfile-3-08.pddl.5.cnf`	305	523	8	34.38	219	7
`ferry2.pfile-L2-C2-04.pddl.1.cnf`	221	402	41	2.0	2	23
`ferry2.pfile-L2-C3-02.pddl.5.cnf`	453	793	4	106.75	421	3
`ferry2.pfile-L3-C3-010.pddl.1.cnf`	302	541	51	2.0	2	31
`ferry2.pfile-L3-C3-07.pddl.1.cnf`	302	541	51	2.0	2	29

Domain Traffic (solved by at most 1 solver)

Graph	Args	Atts	NT SCCs	Avg nt SCC size	Max SCC size	Layers	Solver	Time
`empresa-publica-de-transportes-e-circulao_20141104_0243.gml.20`	930	2577	181	4.31	535	9	-	-
`arlington_va_2016-01.gml.20`	624	847	106	2.55	20	14	-	-
`cascades-east-transit_20151217_0811.gml.80`	207	457	10	20.2	151	4	-	-
`ecobici.stats_20141007_2248.gml.50`	4730	8936	680	5.75	541	16	-	-
`metro-st-louis_20130916_1522.gml.80`	5363	12027	237	22.29	2882	11	-	-
`rtc-ride_20141220_0137.gml.80`	827	1807	51	15.82	365	17	-	-
`spacecoast_fl_2015-12-02.gml.50`	875	1468	167	3.99	40	15	-	-
`villavesas_20150331_1146.gml.50`	505	1011	60	7.25	231	10	-	-
`commuteorg-shuttle_20150308_1938.gml.20`	118	206	14	5.5	27	5	-	-

Graph	**Args**	**Atts**	**NT SCCs**	**Avg nt SCC size**	**Max SCC size**	**Layers**	**Solver**	**Time**
`confederated-tribes-of-the-umatilla-indian-reservation_20130129_1036.gml.80`	66	268	3	22.0	51	3	-	-
`huston_merge_2015-12.gml.80`	6477	16118	258	24.67	4508	13	-	-
`los_angeles_2016-01.gml.80`	8294	19644	333	24.4	5955	10	-	-
`lowell-regional-transit-authority_20121214_0433.gml.80`	128	296	11	11.09	87	4	-	-
`ruter.no_20151217_2005.gml.80`	12455	33767	223	54.81	11459	10	-	-
`transit-services-of-frederick-county_20130128_2046.gml.20`	302	447	40	3.48	12	8	-	-
`translink-archiver_20151219_0124.gml.80`	8746	18471	556	15.3	5726	16	-	-
`tursib_20110626_1306.gml.20`	212	352	36	3.33	18	17	-	-
`view2gt_20150927_1744.gml.50`	312	584	49	5.16	99	7	-	-
`massachusetts_srta_2014-11-13.gml.50`	96	193	9	8.44	38	6	`argmat-mpg`	428.62
`longueuil_qc_2016-01.gml.20`	2241	3537	313	3.47	101	15	`gg-sts`	40.56
`orange-county-transportation-authority_20151202_1534.gml.20`	4065	6543	549	4.09	289	24	`gg-sts`	30.90

Continuation of the previous table

Domain Barabasi-Albert (solved by at most 1 solver)

AAF	**Args**	**Atts**	**NT SCCs**	**Avg nt scc size**	**Max scc size**	**Layers**	**Solver**	**Time**
`BA_80_80_2`	81	145	6	11.83	59	3	-	-
`BA_100_80_4`	101	181	7	12.57	66	3	-	-
`BA_100_90_2`	101	191	5	19.20	74	4	-	-
`BA_120_40_4`	121	169	13	4.77	19	6	-	-
`BA_120_80_3`	121	217	7	14.86	81	4	-	-
`BA_140_70_4`	141	238	10	10.80	65	4	-	-
`BA_160_90_1`	161	305	8	19.13	112	5	-	-
`BA_160_70_2`	161	272	15	8.47	41	4	-	-
`BA_160_80_2`	161	289	11	12.73	97	6	-	-
`BA_180_80_2`	181	325	17	9.53	48	5	-	-
`BA_180_80_3`	181	325	10	15.50	119	4	-	-
`BA_180_90_5`	181	343	4	41.75	155	3	-	-
`BA_200_50_3`	201	301	29	4.48	24	8	-	-
`BA_200_80_1`	201	361	12	14.42	126	5	-	-
`BA_100_70_1`	101	170	9	8.78	46	4	`argmat-mpg`	298.97