Approximate Reasoning with ASPIC+ by Argument Sampling

Matthias THIMM, Tjitze RIENSTRA
University of Koblenz-Landau, Germany

Abstract. We present approximation algorithms for reasoning in structured argumentation approaches such as ASPIC+. While classical approaches consist in constructing all arguments from a knowledge base and determine acceptable arguments from the resulting argument graph, we sample only a small number of arguments and thus only consider a subgraph of the complete argument graph. We develop two approaches for sampling arguments, one which samples arguments independently and one which samples arguments dependently on a query. We present results of an extensive experimental evaluation that showed that runtime can be drastically reduced while the accuracy is only marginally affected.

Keywords. ASPIC+, approximate reasoning, algorithms

1. Introduction

Structured argumentation approaches consider arguments to be collections of formulas and/or rules which entail some conclusion. The most prominent structured approaches are ASPIC+ [11], ABA [14], DeLP [10], and *deductive argumentation* [3]. These approaches consider a knowledge base of formulas and/or rules as a starting point. Queries are answered, e. g. in the case of ASPIC+, by determining all arguments constructible from the knowledge base, identifying attacks between these arguments using e. g. contradictions between conclusions of different arguments, and resolving the conflicts by representing the constructed arguments and attacks as an abstract argumentation framework [7] and relying on reasoning methods for this abstract case. Computationally, reasoning with structured argumentation approaches can be quite demanding as both checking whether a set of formulas and/or rules is an argument can be challenging, and the number of arguments that can be constructed on the basis of a knowledge base may be super-polynomial (and even infinite in some approaches). Some formal analyses on this, in particular regarding the approach of ABA, can be found in [6,8]. Existing solvers for ASPIC+ that implement complete reasoning procedures include TOAST [12] and EPR[1], see also [5].

In this paper, we are interested in approximate methods to reasoning with structured argumentation approaches. This means we are investigating methods that are not necessarily sound and complete in general, but trade correctness for

[1] http://www.wietskevisser.nl/research/epr/

performance. Our approach is motivated by use cases where the number of rules is so high—such as rule mining scenarios as mentioned in [13] which may yield thousands of rules and hundred thousands of arguments— that one has to rely on approximate methods. Of course, one of the aims of ASPIC+ (and any computational model of argumentation) is to provide explainable and correct decision support and we trade this for fast answers. In high-risk domains, correctness is, of course, of higher priority. However, while we give up correctness of reasoning, our approach is still able to provide explanations for reasoning results (i. e. some of the constructed arguments) that can help the decision maker. In our view, this is better than not being able to provide answers at all due to complexity issues of correct reasoning.

For our investigation, we will focus on the propositional logic instantiation under the grounded semantics of the general framework ASPIC+ [11], but our algorithms are general enough to be easily applicable to other instantiations, semantics, and approaches as well. We develop two parametrised algorithms that solve the general problem of checking whether a certain proposition is acceptable wrt. a given knowledge base. We will describe our algorithms in more detail in Section 4. In order to show the practical applicability of our approaches, we perform an experimental evaluation that compares our approaches with baseline methods. This evaluation focuses on the two aspects of *accuracy* and *runtime performance*. Our results show that approximation methods in general and our sampling-based algorithms in particular are viable alternatives for reasoning with complex argumentation scenarios, as runtime can be significantly decreased without losing too much accuracy.

To summarise, the contributions of this paper are as follows:

1. We present two novel approximation algorithms for ASPIC+ that rely on sampling arguments (Section 4).
2. We conduct an extensive experimental evaluation of our approaches that show their viability (Section 5).

Moreover, Section 2 recalls necessary preliminaries while Section 3 describes baseline approaches for reasoning with ASPIC+. Section 6 concludes with a summary.

2. Preliminaries

We recall necessary preliminaries regarding abstract argumentation [7] and ASPIC+ [11].

Definition 1 *An argumentation framework is a pair* $(A, \rightsquigarrow)$ *where* A *is a set whose elements are called* arguments *and* $\rightsquigarrow \subseteq A \times A$ *is a binary relation called* attack. *We also write* $a \rightsquigarrow b$ *instead of* $(a, b) \in \rightsquigarrow$.

We say that a set $E \subseteq A$ is *conflict-free* iff there are no $a, b \in E$ s. t. $a \rightsquigarrow b$. Moreover, a set E *defends* an argument $a \in A$ iff for all $c \in A$ with $c \rightsquigarrow a$ there is $b \in E$ with $b \rightsquigarrow c$. Building on these notions, the basic extension-based semantics [7] can be defined as follows.

Definition 2 *Let* $(A, \rightsquigarrow)$ *be an argumentation framework. A set* $E \subseteq A$ *is said to be* admissible *iff* E *is conflict-free and for all* $a \in E$, E *defends* a*;* complete *iff*

it is admissible and for all $a \in A$ s. t. E defends a, $a \in E$; preferred *iff E is complete and maximal wrt. set inclusion;* grounded *iff E is complete and minimal wrt. set inclusion;* stable *iff E is conflict-free and for all $b \in A \setminus E$ there is $a \in E$ with $a \rightsquigarrow b$.*

Given an argument $a \in A$, a is said to be (credulously) accepted under semantics σ (either *complete*, *grounded*, *stable*, or *preferred*) iff a is a member of at least one σ-extension of $(A, \rightsquigarrow)$.

Structured argumentation approaches can be used to give structure to arguments. In the following, we present a minimal variant of the propositional instantiation of ASPIC+ [11]. Note that ASPIC+ is a general framework that can be instantiated using a variety of different logics and is also able to adhere for the inclusion of orderings between rules, but we only stick to a very simple version in order to show that our algorithms are independent wrt. specific language features.[2]

Let $\mathcal{L}$ be a finite set of propositions and let $\hat{\mathcal{L}}$ be the set of literals of $\mathcal{L}$, i. e., $\hat{\mathcal{L}} = \{a, \neg a \mid a \in \mathcal{L}\}$. For $a \in \mathcal{L}$ define $\overline{a} = \neg a$ and $\overline{\neg a} = a$. ASPIC+ differentiates rules into strict rules (rules that are always supposed to hold) and defeasible rules (rules that "usually" hold).

Definition 3 *A* knowledge base $\mathcal{K}$ *is a pair $\mathcal{K} = (\mathcal{K}_s, \mathcal{K}_d)$ where*

- *$\mathcal{K}_s$ is a set of strict rules $\phi_1, \ldots, \phi_n \to \phi$ with $\phi_1, \ldots, \phi_n, \phi \in \hat{\mathcal{L}}$.*
- *$\mathcal{K}_d$ is a set of defeasible rules $\phi_1, \ldots, \phi_n \Rightarrow \phi$ with $\phi_1, \ldots, \phi_n, \phi \in \hat{\mathcal{L}}$.*

A strict rule $\phi_1, \ldots, \phi_n \to \phi$ with $n = 0$ is written as $\to \phi$ and is also called an *axiom*. A defeasible rule $\phi_1, \ldots, \phi_n \Rightarrow \phi$ with $n = 0$ is written as $\Rightarrow \phi$ and is also called an *assumption*. For practical reasons we often identify $\mathcal{K} = (\mathcal{K}_s, \mathcal{K}_d)$ with $\mathcal{K}_s \cup \mathcal{K}_d$, e. g., expressions such as "$r \in \mathcal{K}$" are to be read as "$r \in \mathcal{K}_s$ or $r \in \mathcal{K}_d$".

Arguments can now be constructed by chaining rules. Following [11], for each argument A we denote by $\mathit{Prem}(A)$ the set of axioms and assumptions used to construct A, $\mathit{Conc}(A)$ is the conclusion of A, $\mathit{Sub}(A)$ is the set of sub-arguments of A, $\mathit{DefRules}(A)$ the set of defeasible rules in A, and $\mathit{TopRule}(A)$ is the last rule used in A.

Definition 4 *The set of arguments $A_{\mathcal{K}}$ generated by a knowledge base $\mathcal{K} = (\mathcal{K}_s, \mathcal{K}_d)$ is inductively defined as follows:*

- *If $\Rightarrow \phi \in \mathcal{K}$ then $(\Rightarrow \phi)$ is an argument with $\mathit{Prem}(\Rightarrow \phi) = \{\Rightarrow \phi\}$, $\mathit{Conc}(\Rightarrow \phi) = \phi$, $\mathit{Sub}(\Rightarrow \phi) = \{\Rightarrow \phi\}$, $\mathit{DefRules}(\Rightarrow \phi) = \{\Rightarrow \phi\}$, $\mathit{TopRule}(\Rightarrow \phi) = (\Rightarrow \phi)$.*
- *If $\to \phi \in \mathcal{K}$ then $(\to \phi)$ is an argument with $\mathit{Prem}(\to \phi) = \{\to \phi\}$, $\mathit{Conc}(\to \phi) = \phi$, $\mathit{Sub}(\to \phi) = \{\to \phi\}$, $\mathit{DefRules}(\to \phi) = \emptyset$, $\mathit{TopRule}(\to \phi) = (\to \phi)$.*
- *If $\phi_1, \ldots, \phi_n \Rightarrow \psi \in \mathcal{K}$ and $A_1, \ldots, A_n$ are arguments such that $\phi_1 = \mathit{Conc}(A_1), \ldots, \phi_n = \mathit{Conc}(A_n)$, then $A = (A_1, \ldots, A_n \Rightarrow \psi)$ is an argument such that: $\mathit{Prem}(A) = \mathit{Prem}(A_1) \cup \ldots \cup \mathit{Prem}(A_n)$, $\mathit{Conc}(A) = \psi$, $\mathit{Sub}(A) = \mathit{Sub}(A_1) \cup \ldots \cup \mathit{Sub}(A_n) \cup \{A\}$, $\mathit{DefRules}(A) = \mathit{DefRules}(A_1) \cup \ldots \cup \mathit{DefRules}(A_n) \cup \{\phi_1, \ldots, \phi_1 \Rightarrow \psi\}$, $\mathit{TopRule}(A) = \phi_1, \ldots, \phi_n \Rightarrow \psi$.*

[2] Note also that we depart from ASPIC+ terminology at times.

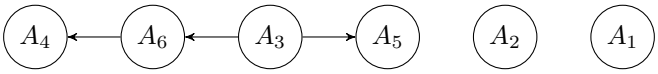

Figure 1. The argumentation framework $AF_{\mathcal{K}}$ from Example 1

- *If* $\phi_1, \ldots, \phi_n \rightarrow \psi \in \mathcal{K}$ *and* $A_1, \ldots, A_n$ *are arguments such that* $\phi_1 = Conc(A_1), \ldots, \phi_n = Conc(A_n)$*, then* $A = (A_1, \ldots, A_n \rightarrow \psi)$ *is an argument such that:* $Prem(A) = Prem(A_1) \cup \ldots \cup Prem(A_n)$*,* $Conc(A) = \psi$*,* $Sub(A) = Sub(A_1) \cup \ldots \cup Sub(A_n) \cup \{A\}$*,* $DefRules(A) = DefRules(A_1) \cup \ldots \cup DefRules(A_n)$*,* $TopRule(A) = \phi_1, \ldots, \phi_n \rightarrow \psi$*.*

An argument A is called *strict* if $DefRules(A) = \emptyset$, otherwise it is called *defeasible*. In our simplified framework, we only consider *rebuts* [11] as the attack relation between arguments.

Definition 5 *Let* A *and* B *be two arguments. We say that* A *attacks* B*, denoted as* $A \rightsquigarrow B$*, if* $Conc(A) = \overline{a}$ *for some* $B' \in Sub(A)$ *of the form* $B'_1, \ldots, B'_n \Rightarrow a$*.*

Using the previous two definitions an abstract argumentation framework can be derived from a knowledge base $\mathcal{K}$ as follows.

Definition 6 *The abstract argumentation framework* $AF_{\mathcal{K}}$ *corresponding to a knowledge base* $\mathcal{K}$ *is an argumentation framework* $AF_{\mathcal{K}} = (A_{\mathcal{K}}, \rightsquigarrow)$ *where* $A_{\mathcal{K}}$ *is the set of arguments generated by* $\mathcal{K}$ *as defined by Definition 4 and* $\rightsquigarrow$ *is the attack relation on* $A_{\mathcal{K}}$ *as defined by Definition 5.*

We conclude this section with a small example illustrating our simplified ASPIC+ framework.

Example 1 *Let* $\mathcal{K} = (\mathcal{K}_s, \mathcal{K}_d)$ *be a knowledge base with*

$$\mathcal{K}_s = \{\rightarrow a, \rightarrow c, \rightarrow \neg d, d \rightarrow \neg b\} \qquad \mathcal{K}_d = \{a \Rightarrow b, c \Rightarrow d\}$$

The corresponding abstract argumentation framework is $AF_{\mathcal{K}} = (A_{\mathcal{K}}, \rightsquigarrow)$ *with* $A_{\mathcal{K}} = \{A_1, \ldots A_6\}$*,*

$$A_1 = (\rightarrow a) \qquad A_2 = (\rightarrow c) \qquad A_3 = (\rightarrow \neg d)$$
$$A_4 = (A_1 \Rightarrow b) \qquad A_5 = (A_2 \Rightarrow d) \qquad A_6 = (A_5 \rightarrow \neg b)$$

and $A_3 \rightsquigarrow A_5$*,* $A_3 \rightsquigarrow A_6$ *and* $A_6 \rightsquigarrow A_4$*.* $AF_{\mathcal{K}}$ *is depicted in Figure 1. Note that* $\{A_1, A_2, A_3, A_4\}$ *is the single grounded, complete, stable, and preferred extension of* $AF_{\mathcal{K}}$*. It follows that* A_1, A_2, A_3, A_4 *are accepted under these semantics.*

3. Reasoning in ASPIC+

As already hinted by Definition 6, reasoning in ASPIC+ is reduced to reasoning in abstract argumentation frameworks. More precisely, given an abstract argumentation semantics σ (either *complete*, *grounded*, *stable*, or *preferred*) as a meta

parameter, we say that a literal $l \in \mathcal{L}$ is *acceptable* wrt. a knowledge base $\mathcal{K}$ if there is an argument A of $\mathcal{K}$ with $Conc(A) = l$ and A is acceptable under σ in $AF_{\mathcal{K}}$.[3] In Example 1, literals $a, c, \neg d, b$ are therefore acceptable for all considered semantics σ.

As the most obvious baseline for the evaluation of our approximate inference algorithms, we consider a direct implementation of Definition 6 that constructs all possible arguments of a knowledge base. More concretely, the NAIVE algorithm first constructs all arguments from axioms and assumptions. Then it iteratively continues by considering each (strict and defeasible) rule and each combination of already constructed arguments for each literal in the body of that rule. It terminates once no further argument can be constructed. However, it is clear that not all arguments from a knowledge base have to be constructed in order to determine the acceptability of a certain literal. Works such as [9,2,4] have shown that many arguments need not to be considered because they are equivalent to other arguments or they are simply irrelevant for a certain query. Therefore, as a second baseline we consider a very simple notion of relevance to prune the set of constructed arguments. More concretely, the *module-based algorithm* (MB) restricts its attention to the subset of rules that is closed under the *syntactic neighbourhood* relationship with the query literal, where a literal or rule is said to be a syntactic neighbour of another rule iff the two share at least one vocabulary element. This approach reduces the set of generated arguments to those that are relevant to the query literal. This assumes that sets of arguments with distinct vocabulary elements (such that no attack is present between these sets) are irrelevant to each other as far as their acceptability is concerned. This property, called *crash-resistence*, is valid under all but the stable semantics [15].

4. Approximate reasoning via sampling

In the following, we present two algorithms that sample arguments in order to obtain a sub-graph of the full corresponding abstract argumentation framework of the knowledge base. We will therefore only modify the part of constructing the argumentation framework, but leave the actual reasoning on the abstract framework untouched. For both algorithms we assume the existence of a function ATTREL(*args*) that, given a set of arguments *args* constructs the attack relation according to Definition 5.

4.1. Independent Sampling

Our first approach consists in simply sampling some number of arguments independently from all constructible arguments. As the total number of constructible arguments is unknown beforehand (and also hard to determine), we cannot tell the algorithm to sample a certain percentage of the constructible arguments. Thus, the algorithm is parametrised by two values *maxArgs* and *maxDuplicates*. The first value *maxArgs* restricts the number of distinct arguments to be sampled. As the same argument may be sampled twice (and certainly will if there are actually less constructible arguments than *maxArgs*) the parameter *maxDuplicates* sets the

[3] Note that other variants of acceptability can be defined.

Algorithm 1 The RAND_I algorithm

```
1: function genRand_I(K, maxArgs, maxDuplicates)
2:     args ← ∅
3:     for 0 ≤ i < maxArgs do
4:         arg ← sampleArgument(K)
5:         if arg ∉ args then args ← args ∪ {arg} else duplicates ← duplicates + 1
6:         if duplicates ≥ maxDuplicates then break
7:     end for
8:     return (args, attRel(args))
9: end function
10:
11: function sampleArgument(K)
12:     arg ← fail
13:     while arg = fail do
14:         ψ ← getRandomLiteral(K)
15:         arg ← constructArgument(K, ψ, {ψ})
16:     end while
17:     return arg
18: end function
19:
20: function constructArgument(K, ψ, acc)
21:     candidateRules ← {r ∈ K | Conc(r) = ψ ∧ Premises(r) ∩ acc = ∅}
22:     if candidateRules = ∅ then return fail
23:     r ← randomElementOf(candidateRules)
24:     subArgSet ← ∅
25:     for φ ∈ Premises(r) do
26:         subArg ← constructArgument(K, φ, acc ∪ {φ})
27:         if subArg = fail then return fail
28:         subArgSet ← subArgSet ∪ {subArg}
29:     end for
30:     return newArgument(r, subArgSet)
31: end function
```

upper bound of how many duplicates can be encountered before the algorithm is terminated, even if *maxArgs* many arguments have not been sampled.

The algorithm RAND_I is shown in Listing 1 (we only show the part responsible for constructing the abstract argumentation framework). The function genRand_I takes as input a knowledge base $\mathcal{K}$ and the two parameters *maxArgs* and *maxDuplicates*. The sampling of a single argument is implemented by the sampleArgument function. It first picks a random literal and then attempts to construct an argument for that literal. The construction, implemented by the constructArgument function, first picks a random rule with the required conclusion, but only if that rule does not contain a premise that already occurs in the argument that is being constructed (this is kept track of by the set *acc*). It then recursively construct sub-arguments for that rule until the argument is complete. If the construction of an argument fails (this happens if there is no rule for some literal) then constructArgument returns "fail" and the sampling is restarted.

In general, if genRand_I terminates, it generates an argumentation framework restricted to a random subset of arguments of size at least 1 (in the rare event that sampleArgument returns the same argument every time) and at most *maxArgs* (in the event that the number of duplicate arguments never exceeds

Algorithm 2 The RAND_D algorithm

```
1: function genRand_D(K, φ, p)
2:   args ← argsWithConc(K, φ)
3:   argsDone ← ∅
4:   concsDone ← {φ}
5:   repeat ← true
6:   while repeat = true do
7:     newArgs ← ∅
8:     for A ∈ args \ argsDone do
9:       argsDone ← argsDone ∪ {A}
10:      if flip(p) then
11:        for ψ ∈ attackingConcs(A) \ concsDone do
12:          concsDone ← concsDone ∪ {ψ}
13:          newArgs ← newArgs ∪ argsWithConc(K, ψ)
14:        end for
15:      end if
16:    end for
17:    if newArgs = ∅ then repeat ← false else args ← args ∪ newArgs
18:  end while
19:  return (args, attRel(args))
20: end function
```

maxDuplicates). However, note that this algorithm may never terminate (theoretically) if at line 14 a literal is repeatedly chosen, for which no argument exists. However, this behaviour was not observed in our experiments (see Section 5).

4.2. Directional Sampling

The directional sampling (RAND_D) algorithm is shown in Listing 2. The following functions are assumed to be defined: $\text{argsWithConc}(\mathcal{K}, \phi)$ returns all arguments A constructible from $\mathcal{K}$ with conclusion ϕ; $\text{attackingConcs}(A)$ returns all formulas ϕ such that, if B is an argument with $Conc(B) = \phi$ then B attacks A; and $\text{FLIP}(p)$ returns true (resp. false) with probability p (resp. $1 - p$).

The algorithm takes as input a knowledge base $\mathcal{K}$, formula ϕ, and probability p and is based on initially constructing all arguments for ϕ, then the attackers of this set, the attackers of those attackers, and so on. However, for each argument A that is constructed, we first call $\text{FLIP}(p)$, and we generate the attackers of A *only* if the result is true (line 10). To avoid generating an argument more than once, we use the set *argsDone* to keep track of arguments whose attackers are generated, and *concsDone* for conclusions for which arguments are generated.

The result of running RAND_D for a query formula ϕ is an argumentation framework which contains all arguments for ϕ as well as a random selection of the (indirect) attackers of those arguments. However, an argument A is included only if all arguments on the directed path from A to some argument for ϕ are also included. This restricts the set of generated arguments to those that are relevant to the query formula, under the assumption that the semantics in use satisfies the *directionality* property [15]. Directionality is satisfied under all semantics we consider except stable semantics.

In general, genRand_D generates an argumentation framework consisting only of the arguments for the given conclusion, plus a subset of its attackers and their attackers etc.

5. Experimental Evaluation

In order to test the feasibility of our algorithms we performed an experimental evaluation. The goal of this evaluation is to show that our approaches significantly reduce the runtime of evaluating queries while still being accurate in their answers.

All algorithms presented in this paper have been implemented using Java in the TweetyProject[4], including the complete algorithms described in Section 3. The source code for all implementations can be found here[5].

5.1. Benchmark data

Due to the lack of existing real-world ASPIC+ knowledge bases, we rely on randomly generated knowledge bases and knowledge bases extracted from sources with originally different purposes. More precisely, in our evaluation we used the following three sets of benchmarks:

Random knowledge bases (Random) We implemented a simple algorithm[6] that randomly generates (propositional logic) ASPIC+ knowledge bases as follows. The algorithm takes as input the number of propositions (n), the number of formulas (m), the maximum number of literals in bodies of rules (l) and the percentage of strict rules (s), and generates m rules, each with at most l body literals (uniformly distributed, zero body literals are also possible, giving rise to axioms and assumptions) and uniformly distributed head literal. For each knowledge base generated in that fashion, we selected a single literal at random to be the fixed query. Using this generator, we created a set of 5400 random instances with $n \in \{5, 10, 15\}$, $m \in \{10, 20, 30\}$, $l \in \{2, 3\}$, $s \in \{0.2, 0.4, 0.6\}$.

Knowledge bases learnt from data (Animals) The dataset "Animals with attributes"[7] describes 50 animals, e. g. ox, mouse, dolphin, using 85 binary attributes such as "swims", "black", and "arctic". Following the approach outlined in [13], we use the Apriori algorithm [1] to mine association rules from this dataset for a given minimal confidence value c and minimal support value s. Association rules with confidence value 1 are interpreted as strict rules, all other rules are interpreted as defeasible rules. Finally, we selected one animal at random and added all but one of its attributes as an axiom, leaving the remaining attribute as the fixed query for the generated knowledge base. We set $c \in \{0.6, 0.65, 0.70, 0.75, 0.8, 0.85, 0.90, 0.95\}$, $s \in \{0.6, 0.65, 0.70, 0.75, 0.8, 0.85, 0.90, 0.95\}$, and allowed maximal 4 literals per rule. The final dataset contained 1920 instances.

Knowledge bases extracted from MaxSAT instances (MaxSAT) For this dataset, we used the set of incomplete weighted SAT instances for the MaxSAT Evaluation 2017[8], which are propositional clauses divided into a set of soft clauses and a set of hard clauses. First, we removed the 7 largest instances from this set, as they brought about severe memory issues for all our solvers.

[4] `http://tweetyproject.org`
[5] `http://tweetyproject.org/r/?r=aspic_reasoner`
[6] `http://tweetyproject.org/r/?r=aspic_random`
[7] `http://attributes.kyb.tuebingen.mpg.de`
[8] `http://mse17.cs.helsinki.fi/benchmarks.html`

For each remaining instance, hard clauses with only one literal are interpreted as axioms. For larger hard clauses, one literal is uniformly selected to be the head of a strict rule while the complements of the remaining literals are put into the body of the rule. Soft clauses are similarly transformed to assumptions and defeasible rules. Note that this transformation does not preserve the semantics of the original instances, but still yields knowledge bases with a structure similar in spirit to the original instances. For each original instance, we generated 10 different variants using this scheme and for each of these variants, we selected one literal at random to be the fixed query. The final dataset contained 1490 instances.

All datasets used in our experiments, i. e., knowledge bases with associated queries, are available online[9].

5.2. Experiment details

Each instance of the three datasets was used to query the following five solvers:

Naive The naive complete algorithm described in Section 3.

MB The module-based complete algorithm described in Section 3.

DIR The complete version of the algorithm RAND_D where the sampling probability is set to 1, see Section 4.

ISAMP The implementation of algorithm RAND_I; as this algorithm depends on two parameters (maximum number of sampled arguments and maximum number of duplicates), we performed a small pre-test in order to find a suitable parameter combination; the results presented in the next section refer to setting the first parameter to 500 and the second one to 50.

DSAMP The implementation of algorithm RAND_D; as this algorithm depends on a parameter (sampling probability), we performed a small pre-test in order to find a suitable parameter; the results presented in the next section refer to setting this parameter to 0.9.

For determining acceptable arguments of the individual corresponding argumentation frameworks we used grounded semantics for all solvers (for reasons of simplicity of computation).

For each instance/solver combination we set a timeout of 5 minutes and recorded the runtime. In order to measure the accuracy of the approximate approaches ISAMP and DSAMP, we considered the subset of instances that could be solved by at least one of the complete approaches Naive, MB, and DIR. For each of those instances we checked whether the approximate approaches returned the same answer and we took the ratio of the number of correct answers to the size of this subset as a measure of accuracy.

We did not include TOAST [12], an already existing (complete) solver for ASPIC+, in our evaluation. As TOAST is only available through a web form and a web service (and no source code is available), runtime performance cannot be compared in a fair manner. Furthermore, we also did not include the complete solver EPR[10] as its approach to construct arguments is equivalent to DIR.

[9] `http://mthimm.de/misc/ds_aspic2020.zip`

[10] `http://www.wietskevisser.nl/research/epr/`

	Random			Animals			MaxSAT		
	S	RT	Acc	S	RT	Acc	S	RT	Acc
Naive	99.9%	617.5	**100.0%**	98.1%	3355.1	**100.0%**	**99.9%**	505.0	**100.0%**
MB	99.8%	571.5	**100.0%**	98.9%	1881.1	**100.0%**	99.6%	316.8	**100.0%**
DIR	99.9%	476.4	**100.0%**	99.3%	2514.4	**100.0%**	99.6%	137.4	**100.0%**
ISAMP	**100.0%**	**336.8**	99.1%	**99.5%**	**455.9**	99.0%	**99.9%**	311.1	99.9%
DSAMP	99.9%	482.6	99.1%	99.4%	1625.1	99.4%	**99.9%**	**122.6**	**100.0%**

Table 1. Results of the experimental evaluation on the three datasets *Random* (5400 instances), *Animals* (1920 instances), and *MaxSAT* (1490 instances); column "S" indicates percentage of solved instances within the time limit of 5 minutes, column "RT" gives the average runtime in milliseconds, and "Acc" gives the percentage of correctly solved instances; the best results per column are highlighted in bold face.

	Random	Animals	MaxSAT
Avg. percentage strict rules	24.4%	14.9%	88.1%
Avg. percentage def. rules	36.9%	52.9%	10.2%
Avg. percentage axioms	15.8%	29.1%	0.2%
Avg. percentage assumptions	22.9%	3.1%	1.5%

Table 2. Statistics of the three datasets *Random*, *Animals*, and *MaxSAT*; cells indicate the average percentage of strict rules, defeasible rules, axioms, and assumptions within each dataset.

All experiments were performed on a single virtual machine running Ubuntu with eight Intel Xeon 2GHz CPUs and 16GB RAM.

5.3. Results

Table 1 summarises the results of our experiments. For each of the three datasets *Random*, *Animals*, and *MaxSAT* and each solver mentioned above, we report on the percentage of solved instances within the time limit of 5 minutes (column "S"), the average runtime of solved instances in milliseconds (column "RT"), and the accuracy as a percentage of the number of correctly solved instances wrt. all solved instances (column "Acc"). In each column we emphasised the best solver with bold face. As a first remark, note that the complete solvers Naive, MB, and DIR have, of course, perfect accuracy. Second, all solvers are quite similar in terms of the percentage of solved instances within the time limit (98%–100%). So let us look a bit closer at the important measures of runtime and accuracy of our sampling-based approaches. In terms of runtime, our new solvers significantly outperfom the baseline approaches in all cases. For the *Random* dataset, the solver ISAMP has an average runtime of 336.8 milliseconds, compared to 476.4 milliseconds of the best performing complete solver (RT). The difference is most significant for the *Animals* dataset where ISAMP has an average 455.9 milliseconds compared to 1881.1 milliseconds (MB), therefore outperforming the complete approaches by a factor over 3. In the *MaxSAT* dataset, the solver DSAMP outperforms the best direct one (DIR), though only slightly. However, recall that

we used grounded semantics to determine acceptable arguments in the corresponding abstract argumentation frameworks, which is a problem of polynomial complexity in the size of the argumentation framework. If we had used a more complex semantics, differences in runtime would have been even more significant. Evaluating this aspect is envisioned for future work.

Interestingly, the gain in performance comes with almost no cost wrt. accuracy. In all datasets, all approximate approaches show an accuracy of at least 99%. This shows that our approaches are indeed competitive for the task at hand.

In order to explain the difference between the performances of solver ISAMP and DSAMP wrt. to the datasets *Random* and *Animals* on the one hand (where ISAMP performs better than DSAMP) and *MaxSAT* on the other (where DSAMP outperforms ISAMP and the differences between the complete approaches and the approximate approaches differ only slightly), we took a closer look at the structure of the knowledge bases in the different datasets. For each instance we computed the distribution of strict rules, defeasible rules, axioms, and assumptions and determined the average of those values for each dataset. Table 2 reports those numbers. Comparing these statistics, we can see that instances in the *MaxSAT* dataset have a significantly larger proportion of strict rules (and therefore fewer defeasible rules). Consequently, many arguments built from this dataset will have few or no attackers as arguments can only be attacked on defeasible rules in our simplified framework, cf. Definition 5. By sampling dependently on the existence of counterarguments, DSAMP likely includes most arguments relevant for making correct decisions (as also indicated by the accuracy of 100% on this dataset). This also explains the similarity of the performances of DSAMP and DIR, the latter being a version of DSAMP with sampling probability 1. On the other hand, ISAMP does not care about the connectedness of arguments and samples arguments independently. This means that many arguments irrelevant to the query are constructed, this resulting in a larger (relative) runtime.

To summarise the results, by sampling arguments we get significant improvements of the average runtime with only minimal loss of accuracy. Moreover, the performance gain is larger when an instance contains many defeasible rules and fewer strict rules.

6. Summary and Conclusion

In this paper we presented the first approximation algorithms for structured argumentation approaches such as ASPIC+. Instead of constructing all possible arguments from a knowledge base, our algorithms only sample a limited number of arguments. We developed two variants of this general idea. The first variant (algorithm RAND_I with implementation ISAMP) samples arguments *independently* from the knowledge base while the second variant (algorithm RAND_D with implementation DSAMP) samples arguments dependently on arguments constructed so far. Our experimental evaluation showed that approximation by sampling significantly improves runtime while accuracy is only marginally affected.

While the present paper used ASPIC+ as the underlying argumentation formalism, it should be noted that our algorithms are general enough to be applied to other structured argumentation approaches such as ABA, DeLP, or deduc-

tive argumentation. Investigating these application formalisms is part of ongoing work.

Acknowledgements The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (grant KE 1686/3-1).

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

[2] Leila Amgoud, Philippe Besnard, and Srdjan Vesic. Equivalence in logic-based argumentation. *Journal of Applied Non-Classical Logics*, 24(3):181–208, 2014.

[3] Philippe Besnard and Anthony Hunter. Constructing argument graphs with deductive arguments: a tutorial. *Argument & Computation*, 5(1):5–30, 2014.

[4] AnneMarie Borg and Christian Straßer. Relevance in structured argumentation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, 2018.

[5] Federico Cerutti, Sarah A. Gaggl, Matthias Thimm, and Johannes P. Wallner. Foundations of implementations for formal argumentation. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14. College Publications, 2018.

[6] Yannis Dimopoulos, Bernhard Nebel, and Francesca Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence*, 141(1):57 – 78, 2002.

[7] Phan Minh Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2):321–358, 1995.

[8] Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14. College Publications, 2018.

[9] Vasiliki Efstathiou and Anthony Hunter. Algorithms for generating arguments and counterarguments in propositional logic. *International Journal of Approximate Reasoning*, 52:672–704, 2011.

[10] Alejandro Javier García and Guillermo Ricardo Simari. Defeasible logic programming: Delp-servers, contextual queries, and explanations for answers. *Argument & Computation*, 5(1):63–88, 2014.

[11] Sanjay Modgil and Henry Prakken. The ASPIC+ framework for structured argumentation: A tutorial. *Argument & Computation*, 5:31–62, 2014.

[12] Mark Snaith and Chris Reed. TOAST: online aspic+ implementation. In *Proceedings of the Fourth International Conference on Computational Models of Argument (COMMA 2012)*, pages 509–510. IOS Press, 2012.

[13] Matthias Thimm and Kristian Kersting. Towards argumentation-based classification. In *Logical Foundations of Uncertainty and Machine Learning, Workshop at IJCAI'17*, August 2017.

[14] Francesca Toni. A tutorial on assumption-based argumentation. *Argument & Computation*, 5(1):89–117, 2014.

[15] Leendert van der Torre and Srdjan Vesic. The principle-based approach to abstract argumentation semantics. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 16. College Publications, 2018.