Solver and Benchmark Descriptions of

ICCMA 2025

Sixth International Competition on Computational Models of Argumentation

Iosif Apostolakis, Andrei Popescu, and Johannes P. Wallner (Editors)

1 Preface

The International Competition on Computational Models of Argumentation (ICCMA for short) is a biennual event with the aim of fostering development of systems and tools for reasoning in computational argumentation. The sixth edition was held in the year 2025, with previous competitions organized in 2015, 2017, 2019, 2021, and 2023.

This edition of ICCMA continues the four main tracks of previous editions: the main track, the heuristics track (formerly approximate), the dynamic track, and the ABA track. Each track consists of a number of sub tracks, mirroring diverse reasoning tasks in these tracks. In this edition of ICCMA we included the complete, stable, preferred, semi-stable, and ideal semantics. As reasoning modes, we include credulous and skeptical acceptance, as well as returning an extension.

The main track is concerned with exact solvers for the argumentative reasoning tasks, and sequential and open-source solvers. The heuristics track aims to evaluate heuristical algorithms for the reasoning tasks. The dynamic track features dynamically evolving arguments and attacks between arguments. The ABA track continues the continued interest in structured argumentation formalisms from a computational perspective.

Novelties for this editions are in particular the inclusion of the Fixed-SAT-Solver Comparison, where a SAT solver is fixed by the organizers, and solvers for argumentation problems can use an interface for using the SAT solver. This allows in particular for clearer comparison of algorithmic approaches utilizing SAT solvers, and mitigate potential performance differences due to different choices of SAT solvers.

In this edition we included different witnesses for specific semantics: for credulous acceptance under complete semantics we allow for admissible sets to be used, and for skeptical acceptance under preferred semantics we allow to use admissible sets attacking the queried argument for a counter-witness.

We thank all contributors to ICCMA, in particular the developers of submitted solvers and developers of benchmark suites. Moreover, we are grateful for the computational resources provided by the Austrian Science Computing (ASC).

Contents

2 Solver Descriptions

100BA – SAT-based Algorithms for ABA Reasoning

Andreas Niskanen
University of Helsinki
Finland

Masood Feyzbakhsh Rankooh
University of Helsinki
Finland

Tuomo Lehtonen
Aalto University
Finland

Matti Järvisalo
University of Helsinki
Finland

***Abstract*—100BA is a SAT-based solver for ABA reasoning tasks submitted to ICCMA 2025. 100BA implements three approaches to ABA reasoning: encoding acyclic derivations in ABA with vertex elimination, and two different approaches that incorporate user propagators in a SAT solver, one implementing acyclicity checking and one implementing unfounded set propagation similar to modern ASP solving techniques.**

I. Introduction

In this system description, we give an overview on 100BA, a solver for reasoning on assumption-based argumentation (ABA) [1] with a SAT-based approach. The solver is submitted for the first time to the sixth International Competition on Computational Models of Argumentation (ICCMA 2025). A dedicated ABA track is included in ICCMA 2025, in particular on the flat logic programming fragment of ABA [1], where rules are similar to logic programming rules and assumptions do not occur in rule heads.

II. System Architecture

To capture reasoning in ABA using declarative encodings requires, in particular, modelling *acyclic* derivation of atoms from assumptions from the given rule-based knowledge base. This presents the challenge of modelling acyclicity over potentially significantly long derivation chains. 100BA incorporates various alternative approaches to realizing SAT-based reasoning for ABA. Our SAT-based approaches come in two types in terms of how the acyclicity constraint for ABA is enforced in conjunction with a SAT encoding of the semantics (assuming the constraint): we consider both (i) fully encoding the acyclicity constraint with propositional clauses as part of the SAT encoding, and (i) handling the acyclicity constraint via a user-defined propagator, extending the standard propagation mechanism within a SAT solver based on the recent IPASIR-UP SAT solver interface [2].

As a SAT encoding, we use a recently proposed compact encoding based on vertex elimination [3] based on a heuristically computed tree decomposition, with the benefit of allowing the SAT solver to directly reason and learn on the propositional level also wrt the acyclicity constraint. For the propagator-based approach, we provide both a SAT modulo acyclicity approach [4] and an unfounded set propagation mechanism similar to state-of-the-art ASP solving techniques [5], directly implemented in a modern SAT solver, namely CaDiCaL [6] v2.1.3. Both the encoding and user propagation approaches are also applicable as the abstraction solver within counterexample-guided abstraction refinement approaches for beyond-NP reasoning, and thus we are able to compute skeptical reasoning under preferred semantics.

III. Participation in ICCMA 2025

We call the three versions of our solver 100BA-VE (direct encoding using vertex elimination), 100BA-ACYC (solver based on acyclicity propagator), and 100BA-UFS (solver based on unfounded set propagation). All versions participate in each subtrack of the ABA track of ICCMA 2025, namely in DC-CO, DC-ST, DS-ST, DS-PR, SE-ST and SE-PR.

Acknowledgments

This work has been financially supported by Research Council of Finland (under grants 347588 and 356046), and Helsinki Institute for Information Technology HIIT. The authors thank the Finnish Computing Competence Infrastructure (FCCI) for computational and data storage resources.

References

[1] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni, "An abstract, argumentation-theoretic approach to default reasoning," *Artificial Intelligence*, vol. 93, pp. 63–101, 1997.

[2] K. Fazekas, A. Niemetz, M. Preiner, M. Kirchweger, S. Szeider, and A. Biere, "Satisfiability modulo user propagators," *J. Artif. Intell. Res.*, vol. 81, pp. 989–1017, 2024. [Online]. Available: https://doi.org/10.1613/jair.1.16163

[3] M. F. Rankooh and J. Rintanen, "Propositional encodings of acyclicity and reachability by using vertex elimination," in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 5861–5868.

[4] M. Gebser, T. Janhunen, and J. Rintanen, "SAT modulo graphs: Acyclicity," in *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, ser. Lecture Notes in Computer Science, E. Fermé and J. Leite, Eds., vol. 8761. Springer, 2014, pp. 137–151. [Online]. Available: https://doi.org/10.1007/978-3-319-11558-0_10

[5] M. Gebser, B. Kaufmann, and T. Schaub, "Conflict-driven answer set solving: From theory to practice," *Artificial Intelligence*, vol. 187, pp. 52–89, 2012.

[6] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," in *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Gurfinkel and V. Ganesh, Eds., vol. 14681. Springer, 2024, pp. 133–152.

ACBAR – Atomic-based Argumentation Solver

Tuomo Lehtonen
Aalto University
Finland

Anna Rapberger
Imperial College London
United Kingdom

Markus Ulbricht
Leipzig University
Germany

***Abstract*—ACBAR is a solver for credulous and skeptical acceptance and finding extensions in Assumption-based Argumentation (ABA). ACBAR makes use of a recently introduced procedure to construct an abstract argumentation framework (AF) from ABA frameworks such that, unlike when using a conventional argument construction method, the size of the AF is polynomially bounded. The state-of-the-art SAT-based AF solver μ-TOKSIA is used on the resulting AF to obtain answers to the reasoning tasks. ACBAR participates in all ABA subtracks of ICCMA 2025, namely DC-CO, DC-ST, DS-ST, DS-PR, SE-ST and SE-PR.**

I. INTRODUCTION

In this system description, we give an overview on ACBAR, short for Atomic-based Argumentation solver, which is a solver for reasoning tasks on assumption-based argumentation (ABA) [1]. The underlying workflow of the solver is based on generating a limited number of abstract arguments and using a solver on the resulting argumentation framework (AF) [2]. This solver was submitted to this year's edition of the International Competition on Computational Models of Argumentation (ICCMA), following previous editions [3]–[7]. Following the success in the previous competition, this year a track is dedicated to reasoning on ABA, in particular on the logic-program fragment of ABA [1], where rules are similar to rules in logic programming. ACBAR was first submitted to ICCMA 2023. ACBAR is written in Python 3 and available in open source at https://bitbucket.org/lehtonen/acbar.

II. SYSTEM ARCHITECTURE

ACBAR translates an ABA framework to an AF, drawing from recent research on bounding the size of an AF instantiated from an ABA framework [8]. In order to avoid a potentially exponential number of arguments that a naive construction method might instantiate (an example is given by Strass, Wyner, and Diller [9]), ACBAR first translates the given ABA framework to an *atomic* ABA framework, which is guaranteed to produce an AF whose size is polynomially bounded in terms of the ABA framework [8]. The procedure we employ to translate an ABA framework to an atomic one requires that the input ABA framework is *non-circular*, i.e. it is impossible that an atom occurs twice in any single derivation tree. In case the input ABA framework is circular, ACBAR first removes the circularity by duplicating rules and atoms occurring in each non-trivial strongly connected component of the rule graph. Finally, given an atomic ABA framework (equivalent under the considered reasoning tasks), ACBAR uses the state-of-the-art AF solver μ-TOKSIA [10] to answer the given task. We outline each of these steps in more detail next.

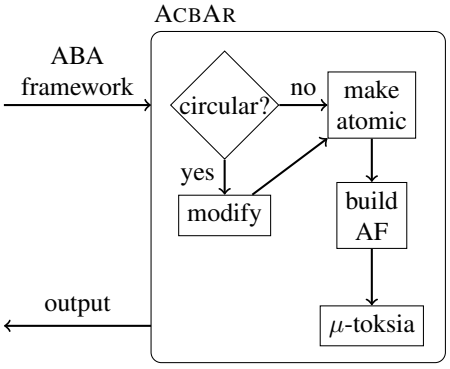


Fig. 1. System architecture of ACBAR

A. Breaking Circularity

In case an input ABA framework is circular, i.e., there are tree derivations with a path from assumptions to the conclusion with the same atom occurring twice, ACBAR first modifies the ABA framework to make it non-circular, but equivalent for reasoning purposes. This is achieved by "copying" rules and atoms to simulate derivations up to a specified height k, where k is a maximum height of a possible derivation. A rule can be used to derive an atom on "level i" by only using atoms on level $j < i$. This procedure is done within each strongly connected component of the dependency graph of the rules of the framework (the dependency graph has an edge from rule heads to each atom in the body of the given rule), with the size of the strongly connected component as the maximum derivation height k. The resulting modified ABA framework is non-circular while preserving all possible derivations.

B. Non-circular ABA to Atomic ABA

Given a non-circular ABA framework, ACBAR translates this framework to a so-called atomic ABA framework, which requires the bodies of each rule to contain only assumptions (i.e., one cannot chain rules for derivations). This is achieved by introducing auxiliary assumptions. Namely, ACBAR introduces two assumptions for each atom (a) occurring in rule bodies: one for a being derivable (a_d) and one for a not being derivable (a_{nd}). Each occurrence of a in a rule body is replaced by a_d, and the contrary of a_d is set as a_{nd} and the contrary

of a_{nd} as a. The translation is achievable in polynomial time and the result is an atomic ABA framework.

C. Atomic ABA to AF

An atomic AF directly gives rise to an AF of polynomial size. Each assumption and each (derivable) rule constitutes an argument. The attack relation between arguments is constructed in the usual way based on which assumptions and atoms occur in the arguments. ACBAR performs this instantiation to an AF.

D. Answering Reasoning Tasks

On the resulting AF we utilize a state-of-the-art SAT-based solver for reasoning tasks on AFs μ-TOKSIA (version ICCMA'21) [10]. The result of a reasoning task on an ABA framework can be read off the result returned by the AF solver, due to correspondences between ABA frameworks and their instantiated AF [11]. ACBAR supports all ABA reasoning tasks included in ICCMA 2025: credulous acceptance under complete and stable semantics (DC-CO, DC-ST), skeptical acceptance under stable and preferred semantics (DS-ST, DS-PR), and finding a stable or a preferred assumption set (SE-ST, SE-PR).

ACKNOWLEDGEMENTS

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) W1255-N23 and P35632, by the Vienna Science and Technology Fund (WWTF) through project ICT19-065, by the German Federal Ministry of Education and Research (BMBF, 01/S18026A-F) by funding the competence center for Big Data and AI "ScaDS.AI" Dresden/Leipzig, and by University of Helsinki Doctoral Programme in Computer Science DoCS. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

REFERENCES

[1] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni, "An abstract, argumentation-theoretic approach to default reasoning," *Artif. Intell.*, vol. 93, pp. 63–101, 1997.

[2] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995.

[3] M. Thimm and S. Villata, "The first international competition on computational models of argumentation: Results and analysis," *Artif. Intell.*, vol. 252, pp. 267–294, 2017.

[4] S. A. Gaggl, T. Linsbichler, M. Maratea, and S. Woltran, "Design and results of the second international competition on computational models of argumentation," *Artif. Intell.*, vol. 279, 2020.

[5] S. Bistarelli, L. Kotthoff, F. Santini, and C. Taticchi, "Summary report for the third international competition on computational models of argumentation," *AI Magazine*, vol. 42, no. 3, pp. 70–73, 2021.

[6] J. Lagniez, E. Lonca, J. Mailly, and J. Rossit, "Introducing the fourth international competition on computational models of argumentation," in *Proc. SAFA*, ser. CEUR Workshop Proceedings, S. A. Gaggl, M. Thimm, and M. Vallati, Eds., vol. 2672. CEUR-WS.org, 2020, pp. 80–85.

[7] M. Järvisalo, T. Lehtonen, and A. Niskanen, "ICCMA 2023: 5th international competition on computational models of argumentation," *Artificial Intelligence*, vol. 342, p. 104311, 2025.

[8] T. Lehtonen, A. Rapberger, M. Ulbricht, and J. P. Wallner, "Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity," in *Proc. KR*. IJCAI, 2023, pp. 440–450.

[9] H. Strass, A. Wyner, and M. Diller, "*EMIL*: Extracting meaning from inconsistent language: Towards argumentation using a controlled natural language interface," *Int. J. Approx. Reason.*, vol. 112, pp. 55–84, 2019.

[10] A. Niskanen and M. Järvisalo, "μ-toksia: An efficient abstract argumentation reasoner," in *Proc. KR*, 2020, pp. 800–804.

[11] K. Čyras, X. Fan, C. Schulz, and F. Toni, "Assumption-based argumentation: Disputes, explanations, preferences," in *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, Eds. College Publications, 2018, ch. 7, pp. 365–408.

AFCA: Searching for Complete Extensions via Enumeration of a Closure System

Sergei Obiedkov
Faculty of Computer Science / cfaed / ScaDS.AI
TU Dresden
Dresden, Germany
Email: sergei.obiedkov@tu-dresden.de
0000-0003-1497-4001

Barış Sertkaya
Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences
Frankfurt am Main, Germany
Email: sertkaya@fb2.fra-uas.de
0000-0002-4196-0150

***Abstract*—AFCA is a solver for abstract argumentation that operates by enumerating closed sets. Unlike most other solvers, it does not rely on SAT or CSP solvers. Instead, AFCA adapts algorithms from formal concept analysis used for enumerating closed sets of a closure system. While the system supports several semantics, this paper focuses on the algorithm used for credulous acceptance under complete semantics. The approach is based on enumerating subsets of arguments that are *semicomplete*—that is, conflict-free sets containing all the arguments they defend. We show that semicomplete sets form a closure system, and we leverage this property to design a targeted enumeration algorithm. In contrast to generic enumeration, our method reduces the search space by dynamically selecting relevant defenders and pruning infeasible branches.**

***Index Terms*—Argumentation frameworks, complete extensions, closure systems.**

I. Introduction

An *argumentation framework* is a directed graph $F = (A, R)$, where A is a finite set of *arguments* and $R \subseteq A \times A$ is the *attack relation* [1]. An edge $(a, b) \in R$ denotes that the argument a *attacks* the argument b.

A set of arguments $S \subseteq A$ *attacks* $b \in A$ if there is $a \in S$ such that $(a, b) \in R$, and b *attacks* S if $(b, a) \in R$ for some $a \in S$. $R(S)$ denotes the arguments attacked by S and $R^{-1}(S)$ denotes those attacking S. If $S = \{a\}$, we write $R(a)$ and $R^{-1}(a)$ instead of $R(\{a\})$ and $R^{-1}(\{a\})$.

We say that $S \subseteq A$ *defends* $a \in A$ if every argument attacking a is attacked by S, i.e., $R^{-1}(a) \subseteq R(S)$. A set S is *self-defending* if it defends all its elements, i.e., $R^{-1}(S) \subseteq R(S)$.

A set $S \subseteq A$ is said to be *conflict-free* if S does not attack any of its elements. Self-defending conflict-free sets are called *admissible*. An admissible set that contains every argument it defends is called a *complete extension* of F.

We present an algorithm for the so-called DC-CO problem: Given an argumentation framework $F = (A, R)$ and an argument $c \in A$, decide if F has a complete extension containing c and, if so, to compute such an extension. Our approach is based on enumerating closure systems.

II. Closure Systems and Complete Extensions

A family $\mathcal{F}$ of subsets of a ground set A is called a *closure system* if $\mathcal{F}$ contains A and is closed under intersection. Elements of a closure system are called *closed sets*. A closure system gives rise to a *closure operator* that maps a subset of A to its (unique) minimal closed superset.

A number of algorithms for enumerating closure systems have been developed, notably, in formal concept analysis [2]–[4]. Some of them have been adapted for enumerating other structures in argumentation frameworks, in particular, stable and preferred extensions [5] and self-defending and admissible sets [6].

To make use of such algorithms for complete extensions, we first relax the requirements associated with them. We call $S \subseteq A$ *semicomplete* if S is conflict-free and contains every argument it defends. The only difference from complete extensions is that semicomplete sets are not required to be self-defending.

Proposition II.1. *Let $F = (A, R)$ be an argumentation framework. The semicomplete subsets of A, together with A, form a closure system.*

Proof. It suffices to show that semicomplete sets are closed under intersection. Let S and T be semicomplete subsets of A. Since S and T are conflict-free, so is $S \cap T$. Let $a \in A$ be defended by $S \cap T$. Then a is defended by both S and T, and, as S and T are semicomplete, both sets contain a. It follows that $S \cap T$ contains every argument it defends and is therefore semicomplete. ■

In a nutshell, our approach (detailed in the next section) is to enumerate semicomplete sets until a complete extension appears in the output.

III. Solving the DC-CO Problem by Enumerating Semicomplete Sets

There is a closure operator that maps a subset of A to its smallest superset in the closure system from Proposition II.1. We assume that `semicomplete`(F, S) is a procedure computing the closure of a conflict-free set S under this operator, except that it returns $\perp$ instead of A when no semicomplete superset of S exists. This can happen, for example, if S defends one of its attackers. The procedure `semicomplete`(F, S) can be implemented in a straightforward way: while there

exists $a \in A \setminus S$ such that $R^{-1}(a) \subseteq R(S)$, add a to S if a does not attack S and return $\bot$ otherwise.

To compute a complete extension containing argument c, we start by checking if c attacks itself. If so, then no complete extension with c exists. Otherwise, we form the minimal semicomplete set S_0 containing c:

$$S_0 := \texttt{semicomplete}(F, \{c\})$$

and then enumerate all its semicomplete supersets until we find one that is also self-defending and thus complete. Enumeration is performed with the `complete` procedure from Fig. 1.

Input: An argumentation framework $F = (A, R)$ and argument sets $S, P \subseteq A$, such that S is semicomplete in F
Output: A complete extension $X \supseteq S$ of F with $X \cap P = \varnothing$ if it exists; $\bot$ otherwise

1: **if** $R^{-1}(S) \subseteq R(S)$ **then** $\{S$ is self-defending$\}$
2: **return** S
3: choose $a \in R^{-1}(S) \setminus R(S)$ $\{$must defend against $a\}$
4: **for all** $d \in R^{-1}(a) \setminus P$ **do** $\{d$ attacks $a\}$
5: $T := \texttt{semicomplete}(F, S \cup \{d\})$
6: **if** $T \neq \bot$ **and** $T \cap P = \varnothing$ **then**
7: $X := \texttt{complete}(F, T, P \cup R(d) \cup R^{-1}(d))$
8: **if** $X \neq \bot$ **then**
9: **return** X
10: $P := P \cup \{d\}$.
11: **return** $\bot$

Fig. 1. Procedure $\texttt{complete}(F, S, P)$

In addition to a set S to be augmented, this procedure takes as input a set P of *prohibited* arguments, which are known to be unsuitable for augmenting S. Initially, P contains arguments that are in conflict with S_0 and self-attacking arguments. Thus, the search for a complete extension of $F = (A, R)$ containing S_0 is initiated as

$$\texttt{complete}(F, S_0, R(S_0) \cup R^{-1}(S_0) \cup \{a \in A \mid aRa\}).$$

In lines 1–2 of the algorithm in Fig. 1, we check if S is self-defending, in which case the algorithm terminates by returning it. Otherwise, we find an attacker a of S that is not attacked by S (line 3) and try to defend S against it by including one of the attackers of a. We consider only those attackers of a that are not in P (line 4).

For each attacker d of a, we tentatively add d to S and extend the result to its minimal semicomplete superset T (line 5). If such T exists and consists entirely of arguments outside P, we attempt to expand it to a complete extension by recursively calling the `complete` procedure. For this call, we extend P with arguments conflicting with d (line 7). If the call succeeds by producing a complete extension, we return it (line 9). If, on the contrary, adding d to S does not allow us to obtain a complete extension, d is added to P (line 10) so as to be able to prune dead-end branches (in line 6) while considering alternative defenders of S.

If adding none of the attackers of a leads to a complete extension, it is not possible to defend S against a. There is then no need to try to defend S against its other attackers, so the algorithm returns $\bot$ (line 11).

There may be different strategies for choosing the attacker of S against which to defend S first (line 3). In our implementation, we choose one that is attacked by the smallest number of arguments outside P; in other words, we minimize the number of arguments to be explored as potential defenders against a (line 4).

The algorithm in Fig. 1 roughly follows the strategy of the `Close by One` algorithm for enumerating closed sets of a closure operator [7]. However, it differs from it in some important aspects (apart from necessary adjustments to the context of abstract argumentation). Thus, `Close by One` employs a fixed order on elements of the ground set and always follows that order when trying to augment the current set. In our algorithm, we dynamically select the argument d to be added to the current set S depending on the arguments against which S must still be defended. `Close by One` attempts adding every possible element, whereas our algorithm considers only the attackers of a single attacker of the current set. This greatly reduces the search space, especially, in relatively sparse frameworks.

IV. Implementation

We implemented the presented algorithm within the AFCA toolkit, which also includes algorithms for several other computational tasks in abstract argumentation (e.g., enumeration of stable and preferred extensions). For clarity reasons, we presented a recursive version of the algorithm; however, the AFCA implementation is non-recursive.

AFCA is implemented in the `C` programming language. It does not require any external libraries or solvers. The source code is available on GitHub.[1]

References

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995.

[2] B. Ganter and R. Wille, *Formal Concept Analysis – Mathematical Foundations*. Springer Cham, 2 ed., 2024.

[3] S. O. Kuznetsov and S. Obiedkov, "Comparing performance of algorithms for generating concept lattices," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 14, no. 2-3, pp. 189–216, 2002.

[4] B. Ganter and S. Obiedkov, *Conceptual Exploration*. Springer, 2016.

[5] S. Obiedkov and B. Sertkaya, "Computing stable extensions of argumentation frameworks using formal concept analysis," in *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings* (S. A. Gaggl, M. V. Martinez, and M. Ortiz, eds.), vol. 14281 of *Lecture Notes in Computer Science*, pp. 176–191, Springer, 2023.

[6] M. Elaroussi, L. Nourine, and M. S. Radjef, "Lattice point of view for argumentation framework," *Ann. Math. Artif. Intell.*, vol. 91, no. 5, pp. 691–711, 2023.

[7] S. O. Kuznetsov, "A fast algorithm for computing all intersections of objects from an arbitrary semilattice," *Nauchno-Tekhnicheskaya Informatsiya Seriya 2-Informatsionnye Protsessy i Sistemy*, no. 1, pp. 17–20, 1993.

[1] https://github.com/sertkaya/afca.

ARIPOTER v2 Participation at ICCMA 2025

Paul Cibier
Sorbonne Université
Paris, France
paul.cibier@gmail.com

Jérôme Delobelle
LIPADE
Université Paris Cité
Paris, France
jerome.delobelle@u-paris.fr

Jean-Guy Mailly
IRIT
Université Toulouse Capitole
Toulouse, France
jean-guy.mailly@irit.fr

Julien Rossit
LIPADE
Université Paris Cité
Paris, France
julien.rossit@u-paris.fr

***Abstract*—We present ARIPOTER, an approximate solver for evaluating the acceptability of arguments under various extension-based semantics. Our solver is based on the computation of the grounded extension, associated with various heuristics for evaluating the acceptability of the arguments that cannot be classified as accepted or rejected solely using the grounded extension. It can solve all the problems in the heuristic track.**

***Index Terms*—Abstract Argumentation, Approximate Algorithms, Grounded Semantics, Gradual Semantics**

I. Introduction

We submit ARIPOTER to the International Competition on Computational Models of Argumentation (ICCMA 2025). Our solver was initially submitted to ICCMA 2023 and the corresponding approach was described in [1]. Our solver is able to solve acceptability problems under several Dung semantics [2]. We describe it the updated version compared to the one that participated to ICCMA 2023.

II. System Description

The general approach consists in computing the grounded extension in order to evaluate if the argument belongs to it (in which case it is accepted under all classical semantics) or attacked by it (in which case it cannot be accepted under the classical semantics). For the remaining case, various heuristics can be used. This kind of approach was originally proposed by [3]. In the rest, we write $\text{IN}(\mathcal{F})$ the set of arguments in the grounded extension, and $\text{UNDEC}(\mathcal{F})$ the set of arguments which are neither in the grounded extension nor attacked by it, where $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ is an argumentation framework.

A. Our Heuristics

Our first family of heuristics is based on gradual semantics.

Definition 1. *Given $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ an AF, $a \in \mathcal{A}$ an argument, γ a gradual semantics, and $\tau \in [0, 1]$, the function Acc^{γ} is defined as follows:*

$$Acc^{\gamma}(\mathcal{F}, a, \tau) = \begin{cases} YES & \text{if } a \in \text{IN}(\mathcal{F}) \\ & \text{or } (a \in \text{UNDEC}(\mathcal{F}) \\ & \text{and } \gamma(\mathcal{F}, a) \geq \tau), \\ NO & \text{otherwise.} \end{cases}$$

For the gradual semantics γ, we consider `h-cat` [4], `Mbs` [5] and `Cbs` [5].

The third author is funded by the French National Research Agency (ANR, Agence National de la Recherche) under the grant ANR-22-CPJ1-0061-0.

Then, we also propose a heuristic based on the structure of the graph, more precisely the number of attacks directed towards an argument, or coming from this argument. Formally, for a an argument, we write a^+ the set of arguments attacked by it, and a^- the set of arguments attacking it.

Definition 2. *Given $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ an AF, $a \in \mathcal{A}$ an argument and $k \in \mathbb{R}$, the function $Acc^{Out/In}$ is defined as follows:*

$$Acc^{Out/In}(\mathcal{F}, a, k) = \begin{cases} YES & \text{if } a \in \text{IN}(\mathcal{F}) \\ & \text{or } (a \in \text{UNDEC}(\mathcal{F}) \\ & \text{and } |a^+| \geq k \times |a^-|), \\ NO & \text{otherwise.} \end{cases}$$

We have empirically evaluated the impact of the parameters on the approach accuracy, and for the competition we have chosen the values described in Table I.

TABLE I
Choice of heuristic and parameter for the different problems.

Problem	Heuristic	Parameter
DC-CO	`Cbs`	0.25
DC-ST	`Cbs`	0.5
DC-SST	`Cbs`	0.25
DC-ID	Out/In	8
DS-PR	`Cbs`	0.5
DS-ST	`Cbs`	0.1
DS-SST	Out/In	8

B. Compiling and Running

Since ARIPOTER is implemented in Rust, you need to have it installed on your system (see https://www.rust-lang.org). Then, you can compile the solver by running `cargo build -r` from the source code directory.

III. Conclusion

Contrary to the previous version that participated to ICCMA 2021, our solver is now implemented in Rust. Its source code is available here: https://github.com/Paulo-21/raripoter. Other updates consist of the implemented gradual semantics that are used for defining the heuristics.

Then, ARIPOTER follows the rules of ICCMA 2025, so it can be run with the following command line: `./aripoter -a ARG -f FILE -p TASK`
where `TASK` is one of DC-CO, DC-ST, DS-PR, DS-ST, DC-SST, DC-ID and DS-SST, `FILE` is the path to a file

describing an AF in the CNF format, and `ARG` is the name of an argument in `FILE`.

The choice of the heuristic and its parameter can also be modified on the command line, but their default values correspond to those in Table I.

REFERENCES

[1] J. Delobelle, J.-G. Mailly, and J. Rossit, "Revisiting approximate reasoning based on grounded semantics," in *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 17th European Conference, ECSQARU 2023, Arras, France, September 19-22, 2023, Proceedings*, ser. Lecture Notes in Computer Science, Z. Bouraoui and S. Vesic, Eds., vol. 14294. Springer, 2023, pp. 71–83. [Online]. Available: https://doi.org/10.1007/978-3-031-45608-4_6

[2] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995. [Online]. Available: https://doi.org/10.1016/0004-3702(94)00041-X

[3] M. Thimm, "Harper++: Using grounded semantics for approximate reasoning in abstract argumentation," http://argumentationcompetition.org/2021/downloads/harper++.pdf, 2021.

[4] P. Besnard and A. Hunter, "A logic-based theory of deductive arguments," *Artif. Intell.*, vol. 128, no. 1-2, pp. 203–235, 2001. [Online]. Available: https://doi.org/10.1016/S0004-3702(01)00071-6

[5] L. Amgoud, J. Ben-Naim, D. Doder, and S. Vesic, "Acceptability semantics for weighted argumentation frameworks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 2017, pp. 56–62. [Online]. Available: https://doi.org/10.24963/ijcai.2017/9

ASPFORABA – ASP-based Algorithms for Reasoning in ABA

Tuomo Lehtonen
Aalto University
Finland

Matti Järvisalo
University of Helsinki
Finland

***Abstract*—ASPFORABA is a solver for credulous and skeptical acceptance and finding extensions in Assumption-based Argumentation (ABA). ASPFORABA is based on encoding ABA reasoning tasks as answer set programming (ASP) and using the state-of-the-art ASP solver CLINGO to obtain solutions. ASPFORABA participates in all ABA subtracks of ICCMA 2025, namely DC-CO, DC-ST, DS-ST, DS-PR, SE-ST and SE-PR.**

I. INTRODUCTION

In this system description, we give an overview on ASPFORABA, a solver for reasoning tasks on assumption-based argumentation (ABA) [1]. This solver was submitted to the fifth International Competition on Computational Models of Argumentation (ICCMA), following previous editions [2]–[6]. ASPFORABA was first submitted to ICCMA 2023, the first year an ABA track was included, and is now resubmitted to ICCMA 2025. ASPFORABA is available in open source at https://bitbucket.org/coreo-group/aspforaba/. We give an overview of the system next. Full descriptions of the algorithms are published in [7], [8].

II. SYSTEM ARCHITECTURE

ASPFORABA utilizes answer set programming (ASP) [9], [10] encodings and ASP-based algorithms for ABA problems utilizing the state-of-the-art ASP solver CLINGO [11]. ASPFORABA supports all ABA reasoning tasks included in ICCMA 2023: credulous acceptance under complete and stable semantics (DC-CO, DC-ST), skeptical acceptance under stable and preferred semantics (DS-ST, DS-PR), and finding a stable or a preferred assumption set (SE-ST, SE-PR). ASPFORABA comprises of a Python 3 program that encodes a given ABA framework and a specified semantics as an ASP program and makes calls to the ASP solver CLINGO on this program and reads the solution to the ABA reasoning problem from the output of CLINGO. We use version 5.6.2 of CLINGO. For the problems in (co)NP, namely DC-CO, DC-ST, DS-ST and SE-ST, one call to CLINGO is sufficient. The problems under preferred semantics (DS-PR and SE-PR) have a higher complexity, and thus ASPFORABA makes multiple calls to CLINGO to solve them. We give some further detail on the ASP encodings [7] and iterative algorithms [8] ASPFORABA employs.

III. ENCODING ABA PROBLEMS IN ASP

Our encodings are based on encoding ABA semantics as originally defined, i.e. in terms of sets of assumptions [1] (instead of sets of arguments). The encodings make a non-deterministic guess over the assumptions and include rules for the derivability of atoms from the guessed set of assumptions. The encodings further include a constraint for conflict- freeness: no contrary of the assumptions may be derived from them.

a) Stable semantics: a further constraint is used to enforce that each assumption is either in the guessed set, or a contrary of this assumption is derived from the guessed set. A set of assumptions satisfying these constraints is stable.

b) Complete semantics: instead of the last constraint in the encoding for stable semantics, the encoding infers which atoms are derivable from the set of assumptions that are not attacked by the guess. A constraint is included to enforce that the not attacked set must not attack the guess (i.e. the guess defends itself). It is further enforced that each assumption is either in the guess or attacked by the assumptions not attacked by the guess (i.e. no assumption is defended by the guess but not included in the guess). A set of assumptions satisfying these constraints is complete.

c) Reasoning tasks: For finding a single extension (SE), simply calling an ASP solver given these encodings and an ABA framework suffices. For acceptance problems, a constraint is added to enforce that the queried atom either is derivable (credulous acceptance) or is not derivable (skeptical acceptance) from the guess. In the former case, if an answer set of the given program exists, the query is credulously accepted. In the latter case we are looking for a counterexample: if the program is unsatisfiable, the query is skeptically accepted.

IV. ASP-BASED ALGORITHMS FOR PREFERRED SEMANTICS

For DS-PR and SE-PR, ASPFORABA implements counterexample-guided abstraction refinement [12] algorithms using incremental solving in CLINGO. We use the encoding for complete semantics as a base abstraction. A preferred assumption set can be found by maximizing a complete assumption set (with iterative ASP calls, enforcing a superset of the assumptions to be included). For skeptical acceptance, we add the requirement that the queried atom is derivable from the assumption set and check whether there is a superset of the maximized set that does not derive the query but is complete. If so, this is a counterexample to the query being skeptically accepted. Otherwise the search is continued after constraining

the search space to not consider any subsets of the maximized complete assumption set. We implement these iterative calls to CLINGO using its Python interface, speeding up solving by taking advantage of incremental computation within CLINGO.

ACKNOWLEDGEMENTS

This work has been financially supported in part by Academy of Finland (grants 322869 and 356046), University of Helsinki Doctoral Programme in Computer Science DoCS, and by the Austrian Science Fund (FWF) P35632. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

REFERENCES

[1] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni, "An abstract, argumentation-theoretic approach to default reasoning," *Artificial Intelligence*, vol. 93, pp. 63–101, 1997.

[2] M. Thimm and S. Villata, "The first international competition on computational models of argumentation: Results and analysis," *Artificial Intelligence*, vol. 252, pp. 267–294, 2017.

[3] S. A. Gaggl, T. Linsbichler, M. Maratea, and S. Woltran, "Design and results of the second international competition on computational models of argumentation," *Artificial Intelligence*, vol. 279, 2020.

[4] S. Bistarelli, L. Kotthoff, F. Santini, and C. Taticchi, "Summary report for the third international competition on computational models of argumentation," *AI Magazine*, vol. 42, no. 3, pp. 70–73, 2021.

[5] J. Lagniez, E. Lonca, J. Mailly, and J. Rossit, "Introducing the fourth international competition on computational models of argumentation," in *Proc. SAFA*, ser. CEUR Workshop Proceedings, S. A. Gaggl, M. Thimm, and M. Vallati, Eds., vol. 2672. CEUR-WS.org, 2020, pp. 80–85.

[6] M. Järvisalo, T. Lehtonen, and A. Niskanen, "ICCMA 2023: 5th international competition on computational models of argumentation," *Artificial Intelligence*, vol. 342, p. 104311, 2025.

[7] T. Lehtonen, J. P. Wallner, and M. Järvisalo, "Declarative algorithms and complexity results for assumption-based argumentation," *Journal of Artificial Intelligence Research*, vol. 71, pp. 265–318, 2021.

[8] ——, "Harnessing incremental answer set solving for reasoning in assumption-based argumentation," *Theory Pract. Log. Program.*, vol. 21, no. 6, pp. 717–734, 2021.

[9] I. Niemelä, "Logic programs with stable model semantics as a constraint programming paradigm," *Annals of Mathematics and Artificial Intelligence*, vol. 25, no. 3-4, pp. 241–273, 1999.

[10] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proc. ICLP/SLP*. MIT Press, 1988, pp. 1070–1080.

[11] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko, "Theory solving made easy with Clingo 5," in *Technical Communications of ICLP*, ser. OASICS. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:15.

[12] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *J. ACM*, vol. 50, no. 5, pp. 752–794, 2003.

Enhanced SAT-based Argumentation Solvers Using in Label Priority Strategy and Large Language Model Tuning

Qi Liu
Hubei University of Technology
Wuhan, China
102401137@hbut.edu.cn

Guangcheng Dong
Hubei University of Technology
Wuhan, China
a257158281@126.com

Aoxu Ji
Hubei University of Technology
Wuhan, China
jiaoxu@hbut.edu.cn

Hang Ding
Hubei University of Technology
Wuhan, China
dinghang@hbut.edu.cn

Mao Luo
Hubei University of Technology
Wuhan, China
luomao@hbut.edu.cn

Ningning He
Hubei University of Technology
Wuhan, China
hnn102211103@163.com

Caiquan Xiong
Hubei University of Technology
Wuhan, China
xiongcq@hbut.edu.cn

Xinyun Wu
Hubei University of Technology
Wuhan, China
xinyun@hbut.edu.cn

Yuanzhi Ke
Hubei University of Technology
Wuhan, China
keyuanzhi@hbut.edu.cn

***Abstract*—We present four solvers—ASSAT_inPriority, ASSAT_Amsat, ASSAT_Cadical and ASSAT_LLM to ICCMA 2025, all of which are improved versions of *ArgSemSAT*. Some solvers replace the original *Minisat* solver used in *ArgSemSAT*, and all solvers incorporate optimizations to the rephasing strategies of the SAT solver. All four solvers participate in the following subtracks of the Main and No-Limits track: SE-{PR, ST, SST}.**

***Index Terms*—Abstract Argumentation, Solver**

I. Introduction

ArgSemSAT is a well-established solver for abstract argumentation frameworks [1]. In particular, *ArgSemSAT* participated in the ICCMA 2017 and was the winner of the preferred semantics track, demonstrating its effectiveness and efficiency in computing argumentation semantics.

ArgSemSAT employs a encoding approach grounded in the three-valued labelling-based semantics [2]. In this method, each argument in the argumentation framework is assigned exactly one of three mutually exclusive labels:

- `in` (I_i): the argument is justified.
- `out` (O_i): the argument is overruled.
- `undec` (U_i): the argument is neither justified nor overruled due to uncertainty.

In our work, we observe that arguments assigned the `in` label typically lead to stronger propagation effects during the SAT solving process, compared to those assigned `out` or `undec` labels. Based on this observation, we adopt the rephasing strategy that prioritizes assigning `in`-label decision variables to true during the SAT solving process.

II. Architecture

A. ASSAT_inPriority

In our previous work [3], our team proposed an in-label prioritizing variable branching strategy that significantly improved the SAT solver from two aspects: the initialization of the polarity values of decision variables and the evaluation criteria for variable's activity, enabling efficient solving of a preferred extension of argumentation frameworks.We further found that the `in`-variable-priority strategy can also be effectively applied in the search process of SAT solvers. In particular, it can be integrated into the rephasing strategy used after each restart, further enhancing the solver's performance and robustness.

Building upon these promising results, We retained the original *Minisat* solver used in *ArgSemSAT*, and based on our observation of the advantages associated with variables labeled as `in`, we modified the original variable polarity initialization strategy, as well as the rephasing strategy. Specifically, during the initial polarity assignment, variables corresponding to `in`-labeled arguments are initialized with negative polarity, whereas variables associated with other labels are initialized with positive polarity. Separately, in the phase saving process of *Minisat*, if a variable corresponds to an `in`-labeled argument, we assign its polarity with a 35% probability of being negative, a 15% probability of being positive, and a 50% probability of preserving its previous polarity. For variables associated with other labels, we assign 15% probability to both negative and positive polarities, while maintaining the previous polarity with a 70% probability. This probabilistic polarity strategy introduces a degree of diversification, and our

experimental results demonstrate that it significantly improves solving performance.

B. ASSAT_AmSAT

In this solver, we replace the original *Minisat* SAT solver of *ArgSemSAT* with the *AmSAT* SAT solver [4], which improves the rephasing strategy and significantly enhances solving efficiency. The *Relaxed CDCL Approach* [5] employed by AmSAT aims at preserving "promising" partial assignments. A partial assignment is considered "promising" if it satisfies two conditions: it does not lead to a conflict and it has a relatively long assignment trail.

Building on this, we further modified *AmSAT*'s rephasing strategy. Specifically, *AmSAT*'s original rephasing strategy randomly selects between two branches, `info_based` and `rand_based`, after each restart. We extended this approach to randomly select among three branches: `info_based`, `rand_based`, and `in_priority`. The implementation of `in_priority` is exactly the same as the rephasing strategy used in *ASSAT_inPriority*.

C. ASSAT_Cadical

In this solver, we replace the original the *Minisat* SAT solver of *ArgSemSAT* with the *Cadical* SAT solver. Motivated by our empirical observations regarding the propagation strength of `in`-labeled arguments, we configure the initial phase assignments such that variables corresponding to `in` labels are initialized to `true`, whereas all other variables are initialized to `false`. This heuristic notably improves the solver's performance by guiding the search toward promising regions of the solution space.

D. ASSAT_LLM

In this solver, we integrate *ArgSemSAT* with large language models to enhance the branching decision process by associating the polarity selection of branching variables with the historical frequency of conflict triggers. Specifically, the solver prefers to explore the polarity direction that has led to fewer conflicts. Leveraging large language models, we generate diverse heuristic strategies and establish an automated framework for code generation and strategy evaluation. Through an evolutionary iteration strategy, we enable a directed exploration of the strategy space, ultimately identifying the optimal conflict-driven branching heuristic.

III. Summary

We presented four SAT-based solvers designed for various reasoning tasks in abstract argumentation frameworks. These solvers integrate techniques including customized rephasing strategies based on `in`-labelled argument behaviors, relaxed backtracking for preserving promising partial assignments, and the use of large language models to automatically generate and refine branching heuristics via evolutionary search. Each solver targets a different aspect of search efficiency and solution quality. The source code for all solvers is available at The source code for all solvers is available at https://github.com/36298liu/Solvers-for-ICCMA-2025.

References

[1] F. Cerutti, M. Vallati, and M. Giacomin, "ArgSemSAT: Solving Argumentation Problems Using SAT," in *Proc. 5th Int. Conf. on Computational Models of Argument (COMMA 2014)*, Frontiers in Artificial Intelligence and Applications, vol. 266, IOS Press, 2014, pp. 455–456.

[2] S. Modgil and M. Caminada, "Proof theories and algorithms for abstract argumentation frameworks," in *Argumentation in Artificial Intelligence*, I. Rahwan and G. R. Simari, Eds. Boston, MA: Springer, 2009, pp. 105–129.

[3] M. Luo, J. Xiong, N. He, C. Xiong, X. Wu, and J. Wu, "An in-label prioritizing variable branching strategy of SAT solvers for a preferred extension of argumentation frameworks," in *PRICAI 2024: Trends in Artificial Intelligence*, vol. 5, Springer, 2025, pp. 216–231.

[4] S. Li, C.-M. Li, M. Luo, J. Coll, S. Cherif, D. Habet, and F. Manyà, "ESA Solvers, Kissat MAB Binary and AMSAT in SAT competition 2023," in *Proc. of SAT Competition 2023 – Solver and Benchmark Descriptions*, 2023, pp. 32–33.

[5] S. Cai and X. Zhang, "Four Relaxed CDCL Solvers," in *Proceedings of SAT Race 2019: Solver and Benchmark Descriptions*, University of Helsinki, 2019, pp. 35–38.

FARGO-LIMITED V2.2.2

Matthias Thimm
Artificial Intelligence Group
University of Hagen
Germany
matthias.thimm@fernuni-hagen.de

***Abstract*—We present FARGO-LIMITED V2.2.2, a solver for heuristic reasoning for various tasks in abstract argumentation. The solver relies on a DPLL-approach to exhaustive search for extensions, but is constrained in the search space by a bounded depth or a bounded number of sub-queries.**

I. INTRODUCTION

An *abstract argumentation framework* AF is a tuple $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ where A is a (finite) set of arguments and R is a relation $\mathsf{R} \subseteq \mathsf{A} \times \mathsf{A}$ [3]. For two arguments $a, b \in \mathsf{A}$ the relation $a\mathsf{R}b$ means that argument a attacks argument b. For a set $S \subseteq \mathsf{A}$ we define

$$S^+ = \{a \in \mathsf{A} \mid \exists b \in S, b\mathsf{R}a\}$$
$$S^- = \{a \in \mathsf{A} \mid \exists b \in S, a\mathsf{R}b\}$$

We say that a set $S \subseteq \mathsf{A}$ is *conflict-free* if for all $a, b \in S$ it is not the case that $a\mathsf{R}b$. A set S *defends* an argument $b \in \mathsf{A}$ if for all a with $a\mathsf{R}b$ there is $c \in S$ with $c\mathsf{R}a$. A conflict-free set S is called *admissible* if S defends all $a \in S$.

Different semantics [1] can be phrased by imposing constraints on admissible sets. In particular, a set E

- is a *complete* (CO) extension iff it is admissible and for all $a \in \mathsf{A}$, if E defends a then $a \in E$,
- is a *grounded* (GR) extension iff it is complete and minimal,
- is a *stable* (ST) extension iff it is conflict-free and $E \cup E^+ = \mathsf{A}$,
- is a *preferred* (PR) extension iff it is admissible and maximal.
- is a *semi-stable* (SST) extension iff it is complete and $E \cup E^+$ is maximal.
- is a *stage* (STG) extension iff it is conflict-free and $E \cup E^+$ is maximal.
- is an *ideal* (ID) extension iff $E \subseteq E'$ for each preferred extension E' and E is maximal.

All statements on minimality/maximality are meant to be with respect to set inclusion.

Given an abstract argumentation framework $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ and a semantics $\sigma \in \{CO, GR, ST, PR, SST, STG, ID\}$ we are interested in the following computational problems [4], [5]:

- DC-σ : For a given argument a, decide whether a is in at least one σ-extension of AF.
- DS-σ : For a given argument a, decide whether a is in all σ-extensions of AF.

Note that DC-σ and DS-σ are equivalent for $\sigma \in \{GR, ID\}$ as those extensions are uniquely defined [1]. For these, we will only consider DS-σ.

The FARGO-LIMITED V2.2.2 solver supports solving the above-mentioned computational problems wrt. to all $\sigma \in \{CO, GR, ST, PR, SST, STG, ID\}$. In the remainder of this system description, we give a brief overview on the architecture of FARGO-LIMITED V2.2.2 (Section II), highlight the changes made since the ICCMA'23 version (Section III), and conclude in Section IV.

II. ARCHITECTURE

The core of the solver lies in an algorithm for heurstically determining whether an argument is contained in an admissible set.[1] For $\sigma \in \{CO, ST, PR, SST, STG, ID\}$ we estimate the answer to a DC-σ query by a positive answer to such a test. For DS-σ, we additionally check whether any attacker of the query argument is (likely) in an admissible set. If the query argument is (likely) contained in an admissible set and no attacker of the query argument is (likely) contained in an admissible set, the answer to DS-σ is positive.

The general algorithm for checking whether a given argument a is contained in an admissible set is given in Algorithm 1. This algorithm is a variant of the standard DPLL-search algorithm [2], where the search direction is influenced by the attack directions. Moreover, the search is bounded by a given maximum depth $n \in \mathbb{N} \cup \{\infty\}$. More precisely, Algorithm 1 is initially called via $\mathsf{admSuperSet}(\mathsf{AF}, \{a\}, n)$. If $S = \{a\}$ is already admissible, we terminate with a positive answer in line 2. As long as the maximum search depth is not reached (lines 3–4), we iterate over all arguments b that attack the current set S and are not defended against (line 6). If there is no possible defender c that can be added to S without violating conflict-freeness, we terminate with a negative answer (lines 6–7). Otherwise, we recursively call the algorithm again with the defender c added to S and the adapted maximum search depth (lines 9–10). Note that the algorithm is complete if the maximum search depth is unbounded, i. e., iff $n = \infty$. If the search depth n is finite, it may happen that the answer is FALSE although a is contained in an admissible set (which could not be found due to the limited search depth). However, if the algorithm's answer is TRUE, this is always

[1] Exceptions are problems DC-GR, DS-GR, DS-CO, which are directly solved by an algorithm running in polynomial time.

the correct answer, since an admissible set has been found. In addition to this algorithm, FARGO-LIMITED V2.2.2 also implements a variant of this algorithm, where the *number* of calls to admSuperSet is limited (instead of the search depth). Experiments have shown that some problems are better handled by this version of the algorithm.

Algorithm 1 (Heuristically) verifying whether a given subset can be extended to an admissible set

Input: $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, $S \subseteq \mathsf{A}$, $n \in \mathbb{N} \cup \{\infty\}$
Output: TRUE if there is admissible S' with $S \subseteq S'$.

admSuperSet(AF,S,n)
1: **if** S is admissible **then**
2: **return** TRUE
3: **if** $n \leq 0$ **then**
4: **return** FALSE
5: **for** $b \in S^- \setminus S^+$ **do**
6: **if** $b^- \setminus (S^- \cup S^+) = \emptyset$ **then**
7: **return** FALSE
8: **for** $c \in b^- \setminus (S^- \cup S^+)$ **do**
9: **if** admSuperSet$(\mathsf{AF}, S \cup \{c\}, n-1)$ **then**
10: **return** TRUE
11: **return** FALSE

FARGO-LIMITED V2.2.2 is written in C++ and relies on no specific libraries other than the C++ standard libraries.

III. CHANGES TO FARGO-LIMITED V1.1.1 (ICCMA'23 VERSION)

The most significant change between FARGO-LIMITED V1.1.1 and FARGO-LIMITED V2.2.2 is the addition of the alternative algorithm for admSuperSet described above. Problems DS-ST and DS-SST are solved using the original depth-bounded variant (with maximum search depth set to 1) while all other problems are solved using the new iteration-bounded variant (with varying numbers of the maximum number of iterations, proportional to the number of arguments in the given argumentation framework). Moreover, FARGO-LIMITED V2.2.2 also first checks whether the query argument is contained or attacked by the grounded extension. In the first case, the solver directly answers positively to the query (for alle problems), in the latter case, the solvers answers negatively (for all problems).

IV. SUMMARY

We presented FARGO-LIMITED V2.2.2, a heuristic solver for various problems in abstract argumentation. The solver relies on a variant of the DPLL-algorithm for searching for admissible sets and includes a maximum search depth. The source code of FARGO-LIMITED V2.2.2 is available at https://github.com/aig-hagen/taas-fargo.

REFERENCES

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

[2] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[3] 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.

[4] 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, February 2018.

[5] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, August 2017.

Fargo-Timelimited-FLiGS System Description for ICCMA 2025

Paul Cibier
Sorbonne Université
Paris, France
paul.cibier@gmail.com

Jean-Guy Mailly
IRIT - Université Toulouse Capitole
Toulouse, France
jean-guy.mailly@irit.fr

***Abstract*—We present Fargo-Timelimited-FLiGS, an approximate solver for evaluating the acceptability of arguments under various extension-based semantics. Our solver adapts the state-of-the-art approach Fargo-limited, combined with a lightweight neural network based on several consecutive linear layer. It can solve the problems DC-CO, DC-ST, DS-PR, DS-ST DC-SST, DC-ID and DS-SST of the heuristic track.**

***Index Terms*—Abstract Argumentation, Approximate Algorithms, DPLL, Neural Networks**

I. Introduction

We submit Fargo-Timelimited-FLiGS to the International Competition on Computational Models of Argumentation (ICCMA 2025). Fargo-Timelimited-FLiGS is based on Fargo-limited [1], a solver that participated to previous editions of ICCMA. Our solver is able to solve acceptability problems under several Dung semantics [2].

II. System Description

A. Overview

Our solver is based on Fargo-limited [1], which applies a DPLL-style algorithm to search for admissible sets, with a bounded depth in order to avoid excessive runtime. In cases where the search stops without a solution, the algorithm returns -1. We have modified Fargo-limited in two directions:

- the limited search is not limited anymore to a number of recursive calls to the algorithm (*i.e.* to the depth of the search tree), but to a limited runtime (fixed in our case to 54 seconds), which means that the exact depth of the search tree depends also on the computational resources (*e.g.* it will be deeper when run on a more powerful CPU);
- When the DPLL-like algorithm returns -1 (*i.e.* when the search fails to find a solution because it has reached its limit), our version relies on another tool, FastLiGS, a solver based on lightweight neural network to evaluate the acceptability of arguments. In the original version of Fargo-limited, when the DPLL search algorithm returns -1, then the solver always returns YES for the DC track and NO for the DS track.
- We use the `flto` compiler option which allows the compiler to realize optimizations on the whole program instead of optimizing each file independently. We noticed a $\times 2$ runtime improvement thanks to this option.

We do not describe FastLiGS in details here since it has also been submitted on its own to ICCMA 2025, and it is thus properly described in the corresponding system description.

B. Compilation

To compile Fargo-Timelimited-FLiGS, you need to have installed Rust to compile FastLiGS, and standard C tools with the boost library installed, (sudo apt-get install libboost-all-dev). Then you need to run the script `./comile_all.sh`, that will handle the compilation of the Rust program FastLiGS, and the C program Fargo-Timelimited-FLiGS.

C. Usage

./fargo-tlimited-fligs -f FILE_PATH -a ARG_NAME -p TASK_NAME -tlimit nb sec DPLL.
The `-tlimit` parameter is the number of seconds allowed for the DPLL like algorithm, *i.e.* 54 seconds by default.

III. Conclusion

The source code of our solver is available here: https://github.com/Paulo-21/fargo-timelimited.

The second author is funded by the French National Research Agency (ANR, Agence National de la Recherche) under the grants ANR-22-CE23-0005 and ANR-22-CPJ1-0061-0.

References

[1] M. Thimm, "Fargo-limited v1.1.1," in *Fifth International Competition on Computational Models of Argumentation (ICCMA'23)*, September 2023.
[2] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995. [Online]. Available: https://doi.org/10.1016/0004-3702(94)00041-X

FastAFGCN: A Memory-Lean GNN-based Approximate Solver using ONNX Runtime

Lars Malmqvist
The Tech Collective
Denmark
lama@thetechcollective.eu

***Abstract*—FastAFGCN is an approximate abstract argumentation solver designed for memory efficiency. It computes the credulous or skeptical acceptance of arguments using Graph Neural Network (GNN) models deployed via the ONNX runtime. The solver first computes the grounded extension using a memory-lean algorithm without dense matrices. If the argument's status is not determined by the grounded extension, it leverages pre-trained and quantized GNN models for inference, applying task-specific thresholds. This approach builds upon previous work on GNN-based approximation but prioritizes runtime efficiency and reduced memory footprint by using ONNX and simplifying runtime feature computation.**

***Index Terms*—abstract argumentation, GNN, ONNX, approximation, memory efficiency, ICCMA**

I. Introduction

Abstract Argumentation Frameworks (AFs) provide a powerful formalism for non-monotonic reasoning by modeling arguments and their attack relations as directed graphs [1]. Determining the acceptability of arguments under various semantics often involves computationally hard problems, many being NP-hard or beyond [1], [2]. This complexity motivates the development of approximate solvers, particularly for large-scale frameworks.

FastAFGCN is an approximate solver that leverages the predictive power of Graph Neural Networks (GNNs), building on previous work like AFGCN [3], [4] and recent findings on GNN-based approximation [5]. However, FastAFGCN introduces significant architectural changes focused on runtime efficiency and reduced memory consumption. It utilizes pre-trained GNN models that have been converted to the Open Neural Network Exchange (ONNX) format and run using the efficient ONNX Runtime.

Key features include:

- A memory-efficient implementation for calculating the grounded extension.
- Deployment of pre-trained, quantized GNN models via ONNX Runtime.
- Simplified runtime feature generation, reducing computational overhead and dependencies.
- Use of task-specific decision thresholds adjustable via a configuration file.

This approach aims to provide fast approximations while minimizing the memory resources required, making it suitable for environments with tighter constraints.

II. Approximation Approach using ONNX

Graph Neural Networks, particularly Graph Convolutional Networks (GCNs) [6], [7], have shown promise for approximating argumentation semantics [3], [5]. FastAFGCN adopts this GNN-based approach but modifies the runtime deployment strategy for efficiency.

Instead of loading a full deep learning framework (like PyTorch) at runtime, FastAFGCN uses pre-trained GNN models that have been exported to the ONNX format. These models are based on GCN architectures similar to previous work [3], [4], incorporating techniques like residual connections and dropout [8], trained using standard methods like Adam optimization [9] with Focal Binary Cross-Entropy loss. The key difference lies in the deployment. Using ONNX Runtime allows for optimized inference across different hardware platforms with a significantly smaller footprint compared to the original training framework. The models used are also quantized (indicated by '_int8.onnx' filenames), further reducing size and speeding up inference.

The runtime process is as follows:

1) The input AF is parsed, storing edges efficiently.
2) The grounded extension is computed using an algorithm optimized to avoid dense adjacency matrices, operating in space proportional to the number of nodes and edges.
3) If the target argument's status is determined by the grounded extension (i.e., it is IN), the solver outputs "YES" immediately.
4) Otherwise, task-specific thresholds are loaded from a 'thresholds.json' file (defaulting to 0.5 if not specified).
5) If inference is needed, simple, randomly initialized node features are generated on-the-fly using PyTorch .
6) The appropriate pre-trained ONNX model for the specified task (e.g., 'DS-PR_int8.onnx') is loaded via ONNX Runtime.
7) Inference is performed using the graph's edge index and the generated random features as input. This yields logits for all arguments.
8) The pre-defined threshold is applied to the logit corresponding to the target argument to produce a "YES" or "NO" decision.

A. Input Features

Unlike the previous AFGCN v2 solver, which utilized complex, pre-computed graph-based features (like centrality

measures, PageRank, coloring), FastAFGCN employs a simpler approach at runtime. The ONNX model is fed only the graph structure (edge index) and basic node features that are randomly initialized at runtime. This design choice significantly reduces the computational cost and library dependencies during the inference phase, contributing to the solver's speed and memory efficiency, assuming the GNN model was trained to work effectively with such minimal features or implicitly learns relevant structural properties. The grounded extension information is used before invoking the GNN, acting as a fast path rather than an input feature to the network itself.

III. Implementation

A. Design of the Solver

The solver consists of a Python script (‘solver.py‘) and a Bash wrapper (‘solver.sh‘). The Python script implements the core logic using ‘numpy‘ for the efficient grounded computation and basic operations, ‘onnxruntime‘ for loading and running the inference models, and ‘torch‘ minimally for generating the random input features required by the ONNX model. The emphasis is on minimizing memory usage, notably by avoiding dense matrix representations where possible and delaying tensor creation until immediately before the ONNX call.

The underlying GNN models are trained offline, using frameworks like PyTorch and graph libraries, on datasets derived from sources like the ICCMA competitions [4], similar to previous versions. These trained models are then converted to quantized ONNX format (‘_int8.onnx‘) for deployment.

The Bash wrapper (‘wrapper.sh‘) provides a command-line interface conforming to ICCMA standards, parsing arguments and invoking the Python script (‘solver.py‘) with the necessary parameters (‘–filepath‘, ‘–task‘, ‘–argument‘).

The ONNX inference step inherently calculates acceptability probabilities for all arguments in parallel, leveraging the efficiency of the GNN and ONNX Runtime. However, the solver only outputs the result for the single argument specified in the query, as per standard decision task requirements. A safety net is included to output ”NO” in case of timeouts or interruptions.

B. Competition Specific Information

FastAFGCN is submitted for the approximate track. It does not participate in exact or other tracks.

The solver implements functionality for the following semantics and problem types:

- Credulous Complete (DC-CO)
- Skeptical Preferred (DS-PR)
- Credulous Stable (DC-ST)
- Skeptical Stable (DS-ST)
- Credulous Semi-stable (DC-SST)
- Skeptical Semi-stable (DS-SST)
- Credulous Ideal (DC-ID)

Support for other tasks would depend on the availability of corresponding trained ‘.onnx‘ model files.

The solver is called via the wrapper script:

```
./solver.sh -p <task> -f <file> -a
<argument_id>
```

Example: `./solver.sh -p DS-PR -f testaf1.af -a 4`

References

[1] 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*, vol. 220, pp. 28–63, 2015.

[2] S. Woltran, “Abstract argumentation – all problems solved ?” *ECAI 2014*, pp. 1–93, 2014.

[3] L. Malmqvist, “Afgcn: An approximate abstract argumentation solver,” *ICCMA 2021*, 2021. [Online]. Available: http://argumentationcompetition.org/2021/downloads/afgcn.pdf

[4] ——, “Approximate solutions to abstract argumentation problems using graph neural networks,” 2022. [Online]. Available: https://etheses.whiterose.ac.uk/32152/6/thesis.pdf

[5] L. Malmqvist, T. Yuan, and P. Nightingale, “Approximating problems in abstract argumentation with graph convolutional networks,” *Artificial Intelligence*, vol. 336, p. 104209, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370224001450

[6] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 9 2019. [Online]. Available: http://arxiv.org/abs/1609.02907

[7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[9] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization.” *ICLR 2015*, pp. 1–15, 2015. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

FastLiGS System Description for ICCMA 2025

Paul Cibier
Sorbonne Université
Paris, France
paul.cibier@gmail.com

Jean-Guy Mailly
IRIT - Université Toulouse Capitole
Toulouse, France
jean-guy.mailly@irit.fr

***Abstract*—We present FastLiGS, an approximate solver for evaluating the acceptability of arguments under various extension-based semantics. Our solver is based on several consecutive linear layer, and can solve the problems DC-CO, DC-ST, DS-PR, DS-ST, DC-SST, DC-ID and DS-SST of the heuristic track.**

***Index Terms*—Abstract Argumentation, Approximate Algorithms, Neural Networks**

I. Introduction

We submit FastLiGS to the International Competition on Computational Models of Argumentation (ICCMA 2025). Our solver is able to solve acceptability problems under several Dung semantics [1].

II. System Description

The general process is as follows: we have trained a neuronal network with some linear layer to solve credulous and skeptical acceptability problems under classical Dung semantics. To do so, we have made the choice to implement the inference solver using Rust only instead of Python so the solver can run with a minimum dependency like him-self and the time to process an instance is also reduced.

Notice that some preprocessing is done. First of all, for all cases (except DS-ST), the solver directly answers NO for self-attacking arguments. Then, following several other approaches in the literature [2], [3], the solver first verifies whether the query argument belongs to the grounded extension or is attacked by it, which prevents running the neural network when the grounded semantics is sufficient for solving the problem.

A. Node Embedding

For each argument in the argumentation framework, we compute features corresponding to the score of the argument w.r.t. several gradual semantics from the literature (h-categorizer [4], no self-attacker [5], Max-based [6], Card-based [6], Euclidian-based [7]) and the grounded semantics (1 if the argument is in the grounded extension, 0 if it is attacked by it, 0.5 otherwise). Other features correspond to the in-degree centrality (number of incoming edges divided by the total number of nodes), the out-degree centrality (number of outgoing edges divided by the total number of nodes), and a value indicating whether the argument is self-attacking (0 if it is self-attacking, 0.5 otherwise).

The second author is funded by the French National Research Agency (ANR, Agence National de la Recherche) under the grants ANR-22-CE23-0005 and ANR-22-CPJ1-0061-0.

B. Network Architecture

We propose a personal structure of neural network composed of four linear layers and a dropout layer:

- First layer: 9 input features to 9^2 output features,
- Second layer: 9^2 input features to 9^2 output features,
- Dropout layer: to help preventing overfitting, with a probability of 20%,
- Third layer: 9^2 input features to 9 outputs features,
- Fourth layer: 9 input features to 1 output with the sigmoid activation function.

The leaky relu activation function was used between all the layers. The output of the sigmoid function represents the probability of acceptability of the argument. The final answer of the solver is then YES if and only if this probability is strictly higher than 0.5.

C. Training and Running the Solver

a) Training: The training was done using PyTorch with Python. 2500 epoch were needed to train the neural network with a dataset composed of a fusion between the benchmarks from the ICCMA 2017 and ICCMA 2023 and soem instances generated with crusti 2gio tools with a maximum of 200 000 arguments per instances (this limit comes from a memory consideration).

After the training, the architecture of the neural network and the associate weight were saved to the ONNX format. When compiling the Rust solver the architecture and the weight are extracted for the ONNX file and integrated into the rust solver.

b) Running: If the binary is build with cargo in release mode, the binary is generally in the target/release directory. Running FastLiGS follows the requirements of ICCMA 2025, *i.e.*:

```
./fligs -p TASK -f FILE -a argument
```

where `TASK` is one of DC-CO, DC-ST, DS-PR, DS-ST, DC-SST, DC-ID and DS-SST, `FILE` is the path to a file describing an AF in the CNF format, and `argument` is the name of an argument in `FILE`.

One can obtain information on the solver by typing:

```
./fligs -h.
```

III. Conclusion

The source code of our solver is available here: https://github.com/Paulo-21/FastLiGS.

REFERENCES

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995. [Online]. Available: https://doi.org/10.1016/0004-3702(94)00041-X

[2] M. Thimm, "Harper++ v1.1.1," in *Fifth International Competition on Computational Models of Argumentation (ICCMA'23)*, September 2023.

[3] P. Cibier and J.-G. Mailly, "Graph convolutional networks and graph attention networks for approximating arguments acceptability," in *Computational Models of Argument - Proceedings of COMMA 2024, Hagen, Germany, September 18-20, 2024*, ser. Frontiers in Artificial Intelligence and Applications, C. Reed, M. Thimm, and T. Rienstra, Eds., vol. 388. IOS Press, 2024, pp. 25–36. [Online]. Available: https://doi.org/10.3233/FAIA240307

[4] P. Besnard and A. Hunter, "A logic-based theory of deductive arguments," *Artif. Intell.*, vol. 128, no. 1-2, pp. 203–235, 2001. [Online]. Available: https://doi.org/10.1016/S0004-3702(01)00071-6

[5] V. Beuselinck, J. Delobelle, and S. Vesic, "A principle-based account of self-attacking arguments in gradual semantics," *J. Log. Comput.*, vol. 33, no. 2, pp. 230–256, 2023. [Online]. Available: https://doi.org/10.1093/logcom/exac093

[6] L. Amgoud, J. Ben-Naim, D. Doder, and S. Vesic, "Acceptability semantics for weighted argumentation frameworks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 2017, pp. 56–62. [Online]. Available: https://doi.org/10.24963/ijcai.2017/9

[7] A. Libman, N. Oren, and B. Yun, "Abstract weighted based gradual semantics in argumentation theory," *CoRR*, vol. abs/2401.11472, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.11472

FUDGE V3.3.4

Matthias Thimm
Artificial Intelligence Group
University of Hagen
Germany
matthias.thimm@fernuni-hagen.de

Federico Cerutti
Department of Information Engineering
University of Brescia
Italy
federico.cerutti@unibs.it

Mauro Vallati
School of Computing and Engineering
University of Huddersfield
United Kingdom
m.vallati@hud.ac.u

***Abstract*—We present FUDGE V3.3.4, an abstract argumentation solver that tightly integrates satisfiability solving technology to solve a series of abstract argumentation problems. While most of the encodings used by FUDGE V3.3.4 derive from standard translation approaches, FUDGE V3.3.4 makes use of sophisticated encodings to solve the skeptical reasoning problem wrt. preferred semantics and problems wrt. ideal semantics.**

I. INTRODUCTION

An *abstract argumentation framework* AF is a tuple $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ where A is a (finite) set of arguments and R is a relation $\mathsf{R} \subseteq \mathsf{A} \times \mathsf{A}$ [6]. For two arguments $a, b \in \mathsf{A}$ the relation $a\mathsf{R}b$ means that argument a attacks argument b. For a set $S \subseteq \mathsf{A}$ we define

$$S^+ = \{a \in \mathsf{A} \mid \exists b \in S, b\mathsf{R}a\}$$
$$S^- = \{a \in \mathsf{A} \mid \exists b \in S, a\mathsf{R}b\}$$

We say that a set $S \subseteq \mathsf{A}$ is *conflict-free* if for all $a, b \in S$ it is not the case that $a\mathsf{R}b$. A set S *defends* an argument $b \in \mathsf{A}$ if for all a with $a\mathsf{R}b$ there is $c \in S$ with $c\mathsf{R}a$. A conflict-free set S is called *admissible* if S defends all $a \in S$.

Different semantics [1] can be phrased by imposing constraints on admissible sets. In particular, set E

- is a *complete* (CO) extension iff it is admissible and for all $a \in \mathsf{A}$, if E defends a then $a \in E$,
- is a *grounded* (GR) extension iff it is complete and minimal,
- is a *stable* (ST) extension iff it is conflict-free and $E \cup E^+ = \mathsf{A}$,
- is a *preferred* (PR) extension iff it is admissible and maximal,
- is an *ideal* (ID) extension iff $E \subseteq E'$ for each preferred extension E', E is admissible, and E is maximal,
- is a *semi-stable* (SST) extension E is admissible and $E \cup E^+$ is maximal, and
- is a *stage* (SST) extension E is conflict-free and $E \cup E^+$ is maximal.

All statements on minimality/maximality are meant to be with respect to set inclusion.

Given an abstract argumentation framework $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ and a semantics $\sigma \in \{CO, GR, ST, PR, ID, SST, STG\}$ we are interested in the following computational problems:

SE-σ: Given AF, compute some σ-extension.

DC-σ: Given AF and an argument a, decide whether a is in at least one σ-extension of AF.

DS-σ: Given AF and an argument a, decide whether a is in all σ-extensions of AF.

Note that DC-σ and DS-σ are equivalent for $\sigma \in \{GR, ID\}$ as those extensions are uniquely defined [1]. For these, we will only consider DS-σ.

The FUDGE V3.3.4 solver supports solving the above-mentioned computational problems wrt. all $\sigma \in \{CO, GR, ST, PR, ID, SST, STG\}$. In the remainder of this system description, we give a brief overview on the architecture of FUDGE V3.3.4 (Section II), highlight the changes made since the ICCMA'23 version (Section III), and conclude in Section IV.

II. ARCHITECTURE

FUDGE V3.3.4 follows the standard reduction-based approach to solve the above-mentioned reasoning problems [2], [4] with the target formalism being the satisfiability problem SAT [3]. For example, given the problem SE-ST and an input argumentation framework $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$, first, for each argument $a \in \mathsf{A}$, we create propositional variables in_a and out_a, with the meaning that in_a (resp. out_a) is true in a satisfying assignment iff the argument a is in (resp. attacked by) the stable extension to be found. Then

$$\Phi_1(\mathsf{AF}) = \bigwedge_{a \in \mathsf{A}} (\neg \text{in}_a \vee \neg \text{out}_a)$$

and

$$\Phi_2(\mathsf{AF}) = \bigwedge_{a \in \mathsf{A}} (\text{out}_a \Leftrightarrow \bigvee_{(b,a) \in \mathsf{R}} \text{in}_b)$$

model the basic intuition behind these variables and, in particular, ensure conflict-freeness of the modelled extension. The constraint that all arguments not included in the extension must be attacked can be modelled by

$$\Phi_3(\mathsf{AF}) = \bigwedge_{a \in \mathsf{A}} (\text{in}_a \vee \text{out}_a)$$

Then the formula $\Phi_1(\mathsf{AF}) \wedge \Phi_2(\mathsf{AF}) \wedge \Phi_3(\mathsf{AF})$ is satisfiable iff AF has a stable extension and a stable extension can be easily extracted from a satisfying assignment of $\Phi_1(\mathsf{AF}) \wedge \Phi_2(\mathsf{AF}) \wedge \Phi_3(\mathsf{AF})$. All reasoning problems on the first level

of the polynomial hierarchy [7] can be solved in a similar manner.

Particularly challenging problems are those wrt. preferred semantics as, in particular, $DS\text{-}PR$ is Π_2^P-complete [7]. To solve that problem, we use the approach of [8]. This approach relies on the following observation[1]:

Theorem 1 ([8]). *$a \in \mathsf{A}$ is skeptically accepted wrt. preferred semantics iff*

1) *there is an admissible set S with $a \in S$ and*
2) *for every admissible set S with $a \in S$ and every admissible set S' with $S' \mathsf{R} S$, there is an admissible set S'' with $S' \cup \{a\} \subseteq S''$.*

The above theorem states that we can decide skeptical acceptance wrt. preferred semantics by considering only those admissible sets that attack an admissible set containing the argument in question. As an admissibility check can be solved by a satisfiability check, similarly as above, the above insight leads to an algorithm that can solve $DS\text{-}PR$ without actually computing preferred extensions.

Coming to ideal semantics, it is worth recalling [5, Theorem 3.3].

Theorem 2 ([5, Theorem 3.3]). *An admissible set of arguments S is ideal iff for each argument a attacking S there exists no admissible set of arguments containing a.*

The interesting aspect of [5, Theorem 3.3] is that ideal semantics, although defined based on skeptical acceptance wrt. preferred semantics, does not rely on that notion. Moreover, in [8] we also prove that starting from the set of arguments which are not attacked by an admissible set, its largest admissible set is the ideal extension. We can therefore tweak the machinery we created for computing skeptical acceptance wrt. preferred semantics to compute the ideal extension too. The complete algorithm is presented in [8].

FUDGE V3.3.4 is written in C++ and uses the IPASIR[2] interface to connect to a satisfiability solver. It comes with CaDiCaL 2.1.0[3] pre-configured. FUDGE V3.3.4 improves over its previous version v2.4 submitted at ICCMA'21 by using a different method for calling the SAT solver, as well as streamlined encodings, and support for the semi-stable and stage semantics.

III. CHANGES TO FUDGE V3.2.8 (ICCMA'23 VERSION)

The most significant change is the adoption of the IPASIR interface to connect to any satisfiability solver. Further changes pertain to bug fixes and code streamlining.

IV. SUMMARY

We presented FUDGE V3.3.4 a reduction-based solver for various problems in abstract argumentation. FUDGE V3.3.4 leverages on a mix of standard and novel SAT encodings to solve reasoning problems, with the aim of avoiding the costly maximisation step that is characteristic of some of the abstract argumentation problems. The source code of FUDGE V3.3.4 is available at https://github.com/aig-hagen/taas-fudge.

[1] Define $S' \mathsf{R} S$ iff there is $a \in S'$ and $b \in S$ with $(a, b) \in \mathsf{R}$.

[2] https://github.com/biotomas/ipasir

[3] http://fmv.jku.at/cadical/

REFERENCES

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

[2] Philippe Besnard, Sylvie Doutre, and Andreas Herzig. Encoding argument graphs in logic. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems - IPMU 2014*, 2014.

[3] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[4] 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 15. College Publications, February 2018.

[5] P. M. Dung, P. Mancarella, and F. Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(10):642–674, 2007.

[6] 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.

[7] 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, February 2018.

[8] Matthias Thimm, Federico Cerutti, and Mauro Vallati. Skeptical reasoning with preferred semantics in abstract argumentation without computing preferred extensions. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, 2021.

HARPER++ V1.1.2

Matthias Thimm
Artificial Intelligence Group
University of Hagen
Germany
matthias.thimm@fernuni-hagen.de

***Abstract*—We present HARPER++ V1.1.2, a solver for heuristic reasoning for various tasks in abstract argumentation. The solver operates by determining the grounded extension of an input argumentation framework and answering queries solely based on information extracted from that extension.**

I. Introduction

An *abstract argumentation framework* AF is a tuple $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ where A is a (finite) set of arguments and R is a relation $\mathsf{R} \subseteq \mathsf{A} \times \mathsf{A}$ [3]. For two arguments $a, b \in \mathsf{A}$ the relation $a\mathsf{R}b$ means that argument a attacks argument b. For a set $S \subseteq \mathsf{A}$ we define

$$S^+ = \{a \in \mathsf{A} \mid \exists b \in S, b\mathsf{R}a\}$$
$$S^- = \{a \in \mathsf{A} \mid \exists b \in S, a\mathsf{R}b\}$$

We say that a set $S \subseteq \mathsf{A}$ is *conflict-free* if for all $a, b \in S$ it is not the case that $a\mathsf{R}b$. A set S *defends* an argument $b \in \mathsf{A}$ if for all a with $a\mathsf{R}b$ there is $c \in S$ with $c\mathsf{R}a$. A conflict-free set S is called *admissible* if S defends all $a \in S$.

Different semantics [1] can be phrased by imposing constraints on admissible sets. In particular, a set E

- is a *complete* (CO) extension iff it is admissible and for all $a \in \mathsf{A}$, if E defends a then $a \in E$,
- is a *grounded* (GR) extension iff it is complete and minimal,
- is a *stable* (ST) extension iff it is conflict-free and $E \cup E^+ = \mathsf{A}$,
- is a *preferred* (PR) extension iff it is admissible and maximal.
- is a *semi-stable* (SST) extension iff it is complete and $E \cup E^+$ is maximal.
- is a *stage* (STG) extension iff it is conflict-free and $E \cup E^+$ is maximal.
- is an *ideal* (ID) extension iff $E \subseteq E'$ for each preferred extension E' and E is maximal.

All statements on minimality/maximality are meant to be with respect to set inclusion.

Given an abstract argumentation framework $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ and a semantics $\sigma \in \{CO, GR, ST, PR, SST, STG, ID\}$ we are interested in the following computational problems [4], [5]:

- DC-σ : For a given argument a, decide whether a is in at least one σ-extension of AF.
- DS-σ : For a given argument a, decide whether a is in all σ-extensions of AF.

Note that DC-σ and DS-σ are equivalent for $\sigma \in \{GR, ID\}$ as those extensions are uniquely defined [1]. For these, we will only consider DS-σ.

The HARPER++ V1.1.2 solver supports solving the above-mentioned computational problems wrt. to all $\sigma \in \{CO, GR, ST, PR, SST, STG, ID\}$. In the remainder of this system description, we give a brief overview on the architecture of HARPER++ V1.1.2 (Section II), highlight the changes made since the ICCMA'23 version (Section III), and conclude in Section IV.

II. Architecture

The HARPER++ V1.1.2 solver is based on the insight that grounded semantics almost perfectly captures other semantics in many practical instances of argumentation frameworks [2]. In particular, arguments contained in the grounded extension are always contained in every σ-extension as long as σ is based on complete semantics (which is true for all semantics considered except stage semantics). So a positive answer to DS-GR implies a positive answer to DS-σ and DC-σ for these other semantics σ. On the other hand, if an argument is attacked by an argument contained in the grounded extension then the answer to DS-σ and DC-σ is negative for these semantics due to these semantics being based on conflict-freeness and the aforementioned observation. In [2] it has been observed that on many practical instances of argumentation frameworks—i. e., those used as benchmarks in previous competitions—skeptical reasoning with any semantics often coincides with reasoning with grounded semantics in general. For example, the Jaccard distance[1] between the grounded extension and the set of arguments contained in each (some) preferred extensions averaged over a set of 426 argumentation frameworks compiled from assumption-based argumentation frameworks submitted to ICCMA 2017[2], has been observed to be 0.03 (0.06) [2]. The advantage of only relying on grounded semantics to heuristic reasoning with other semantics, is of course tractability: computing the grounded extension can be done in polynomial time [4].

For any $\sigma \in \{CO, ST, PR, SST, STG, ID\}$, the computational problem DS-σ is solved by the HARPER++ V1.1.2 solver via

[1]The Jaccard distance J between two sets X_1, X_2 is defined via $J(X_1, X_2) = 1 - \frac{|X_1 \cap X_2|}{|X_1 \cup X_2|}$ and it is zero iff the two sets are the same.

[2]http://argumentationcompetition.org/2017/ABA2AF.pdf

1) If the query argument is in the grounded extension, the answer is YES,
2) otherwise the answer is NO.

The problem DC-σ is addressed slightly different: arguments neither in the grounded extension nor attacked by it are accepted, as it is likely that those arguments appear in at least one σ-extension. So, for any $\sigma \in \{CO, ST, PR, SST, STG, ID\}$, the computational problem DC-σ is solved by the HARPER++ V1.1.2 solver via

1) If the query argument is attacked by an argument in the grounded extension, the answer is NO,
2) otherwise the answer is YES.

III. CHANGES TO HARPER++ V1.1.1 (ICCMA'23 VERSION)

No changes have been made between HARPER++ V1.1.1 and HARPER++ V1.1.2 other than incrementing the version number to indicate the ICCMA'25 version.

IV. SUMMARY

We presented HARPER++ V1.1.2, a heuristic solver for various problems in abstract argumentation. HARPER++ V1.1.2 only makes use of the information provided by the grounded extension to answer queries with respect to other semantics. The source code of HARPER++ V1.1.2 is available at https://github.com/aig-hagen/taas-harperpp.

REFERENCES

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

[2] Federico Cerutti, Matthias Thimm, and Mauro Vallati. An experimental analysis on the similarity of argumentation semantics. *Argument & Computation*, 11(3):269–304, October 2020.

[3] 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.

[4] 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, February 2018.

[5] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, August 2017.

Heuback: A Task-Driven Approximate Solver for Admissibility-Related Semantics

Lars Malmqvist
The Tech Collective
Denmark
lama@thetechcollective.eu

***Abstract*—Heuback is an approximate solver for abstract argumentation frameworks focused on admissibility-related decision problems. It employs a task-driven approach, selecting its strategy based on the specific ICCMA task provided. The solver always begins by computing the exact grounded extension. If the query argument's status is determined, it terminates. Otherwise, depending on the task, it either relies solely on the grounded result (for tasks like DC-ID and DS-PR, providing an exact result for grounded semantics but potentially an approximation for others), employs a specific SCC reinstatement heuristic (for DS-ST), or utilizes a time-limited backtracking search algorithm (for tasks like DC-CO, DC-ST). The backtracking search attempts to find an admissible set containing the query; however, due to the time limit, its "OUT" responses are approximate. The implementation leverages Python with optional NumPy and Numba acceleration.**

***Index Terms*—abstract argumentation, admissibility, approximation, backtracking, grounded extension, heuristics, time-limited search, task-based solver, ICCMA**

I. Introduction

Abstract Argumentation Frameworks (AFs), introduced by Dung [1], provide a powerful formalism for non-monotonic reasoning [2]. Determining the acceptability of arguments under various semantics often involves computationally hard problems [2], motivating the development of approximate solvers for large or complex instances.

Heuback is an approximate solver designed to tackle decision problems for several standard argumentation semantics by leveraging admissibility checks. It implements a task-driven workflow: it first computes the exact grounded extension. If this resolves the query, the exact result is returned. If not, Heuback selects a subsequent strategy based on the task ('-p' parameter). These strategies include using only the grounded result (exact for grounded semantics, approximate for others like DS-PR), applying a specific SCC reinstatement heuristic (for DS-ST), or performing a backtracking search for credulous admissibility (for tasks like DC-CO, DC-ST). Crucially, the backtracking search is time-limited; if it fails to find an admissible set within the given time, it returns "OUT", which constitutes an approximation.

Key features include:

- An exact, efficient implementation for calculating the grounded extension.
- Task-based dispatching to different algorithms (grounded-only, heuristic, backtracking).
- A time-limited backtracking search algorithm for approximating credulous admissibility, enhanced with preprocessing (SCC reduction, k-hop cone).
- A specific SCC reinstatement heuristic used as an approximation for the DS-ST task.
- Acceleration using NumPy and Numba for performance-critical sections.
- Adherence to the ICCMA command-line interface via a wrapper script.

This approach aims to provide fast, often exact (if grounded suffices), but ultimately approximate decisions for complex tasks by employing time limits and heuristics.

II. Solver Architecture and Algorithms

Heuback operates through a defined sequence of steps, adapting its strategy based on the input task and intermediate results.

The runtime process is as follows:

1) The input AF is parsed from the specified file (apx format), storing the graph structure using adjacency lists.
2) The grounded extension is computed exactly using an optimized algorithm (leveraging Numba/CSR format if available, otherwise pure Python).
3) If the query argument is found within the computed grounded extension, the solver outputs "IN" (exact result) and terminates.
4) If the query argument is not in the grounded extension, the solver consults the specified task ('-p' parameter):
   - **For tasks like DS-PR, DS-SST, DC-ID:** The solver concludes "OUT". This relies solely on the exact grounded computation, which serves as an approximation for the requested semantics (e.g., skeptical preferred is not always equivalent to grounded).
   - **For task DS-ST:** The solver invokes the SCC Reinstatement Heuristic. This heuristic checks if all external attackers of the query argument's SCC are outside the grounded extension. This provides an approximate answer for DS-ST, outputting "IN" or "OUT" based on the heuristic check.
   - **For tasks like DC-CO, DC-ST, DC-SST:** The solver initiates a time-limited backtracking search to find an admissible set containing the query argument. This involves:

- Preprocessing: Optionally reducing the graph (SCC reduction) and extracting a k-hop cone.
- Bitset Representation: Using efficient bitsets for the subgraph.
- Time-Limited Backtracking Search: Performing a recursive DFS attempt to build an admissible set. If a set is found within the time limit (specified internally or defaulted), it outputs "IN". If the search space is exhausted without finding a set, or if the time limit is reached, it outputs "OUT". **This "OUT" result is approximate**, as an admissible set might exist but was not found in time.

A. Performance Optimizations

To enhance performance, Heuback utilizes these dependencies:

- **NumPy:** Enables the use of efficient array operations, particularly for creating CSR representations and managing bitsets during backtracking.
- **Numba:** Provides Just-In-Time (JIT) compilation for performance-critical functions like grounded extension calculation, conflict checking, and admissibility defect calculation.

The solver remains fully functional without these libraries using pure Python fallbacks, albeit with significantly reduced speed, making timeouts in the backtracking search more likely. The competition version has a hard dependency on these libraries.

III. Implementation

A. Design of the Solver

The solver comprises a Python script (‘heuback.py‘) and a Bash wrapper (‘solver.sh‘). The Python script implements the core logic using standard libraries, optionally NumPy/Numba. It uses adjacency lists, CSR, and bitsets. The emphasis is on providing fast decisions, accepting approximation (via time limits or heuristics) for computationally hard tasks.

The time-limited backtracking algorithm uses recursion with pruning and heuristics. The SCC reinstatement heuristic provides a fast check for DS-ST. The overall solver provides approximate answers for several tasks, with the degree of approximation depending on the task, the instance complexity, and the time limit imposed on the backtracking search.

The Bash wrapper (‘solver.sh‘) parses standard ICCMA command-line arguments and invokes the Python script (‘heuback.py‘) appropriately.

B. Competition Specific Information

Heuback is submitted as an approximate solver.

The solver implements functionality for the following tasks specified via the ‘-p‘ flag:

- DS-PR (approximate, uses grounded)
- DC-CO (approximate, uses time-limited backtracking)
- DS-ST (approximate, uses SCC heuristic)
- DS-SST (approximate, uses grounded)
- DC-ST (approximate, uses time-limited backtracking)
- DC-SST (approximate, uses time-limited backtracking)
- DC-ID (approximate, uses grounded)

The results for DC tasks relying on backtracking are approximate due to the time limit. DS-ST uses a specific heuristic. Other DS tasks and DC-ID rely on the exact grounded computation, which serves as an approximation for those semantics.

The solver is called via the wrapper script:

```
./solver.sh -p <task> -f <file> -a <argument_id>
```

Example: `./solver.sh -p DC-CO -f testAF.apx -a 5`

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*, vol. 77, no. 2, pp. 321–357, 1995.

[2] G. Charwat, W. Dvořák, S. Gaggl, J. P. Wallner, and S. Woltran, "Computational Aspects of Abstract Argumentation," in *Handbook of Formal Argumentation*, vol. 1, P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, Eds. College Publications, 2018, pp. 367–448.

MS-DIS: Multi-Shot ASP-Driven ABA Disputes*

Martin Diller
Logic Programming and Argumentation Group
TU Dresden, Germany
martin.diller@tu-dresden.de

Piotr Gorczyca
Computational Logic Group
TU Dresden, Germany
piotr.gorczyca@tu-dresden.de

***Abstract*—We describe MS-DIS, a system implementing dispute derivations for assumption-based argumentation (ABA) via multi-shot answer set programming (ASP). It participates in the ICCMA 2025 ABA tracks for credulous acceptance under the complete and stable semantics. While primarily designed to support dialectical explication rather than performance, MS-DIS also showcases the use of the process of argumentation itself as the core reasoning mechanism.**

I. Introduction

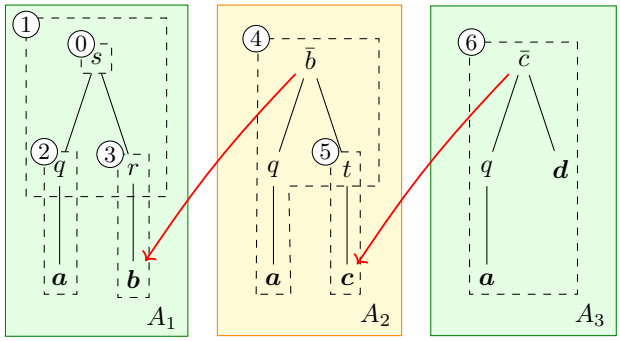
Dispute derivations are among the primary native reasoning methods for assumption-based argumentation (ABA) [1]. Inspired by games for abstract argumentation [2], they are conceptualized as structured disputes in which arguments supporting or challenging a given claim are exchanged between a proponent and an opponent. Beyond serving as a reasoning mechanism for determining the acceptability of claims, dispute derivations also offer a framework for dialectical explication.

MS-DIS implements the rule-based representation of (flexible) ABA disputes first introduced in [3]. This version of ABA disputes further simplifies the earlier graph-based dispute model from [4]—itself based on [5]—by having the proponent and opponent exchange rules, rather than arguments. While arguments are not represented explicitly, they can be reconstructed from the rule-based exchange.

The implementation makes use of multi-shot answer set programming (ASP) using the clingo solver [6]. This enables the incremental grounding and solving of logic programs across multiple iterations, while reducing some of the overhead typically associated with repeated invocations of an ASP solver.

In preliminary experiments, we found that although MS-DIS —unsurprisingly, given its focus on explication—still clearly lags behind inference-focused systems such as aspforaba [7], it outperforms flexABle, a direct implementation of rule-based ABA disputes. Notably, flexABle was the only dispute-based system to participate in the most recent ICCMA competition (2023), and has been shown to be more efficient than earlier Prolog-based implementations of ABA disputes [8]. Nevertheless, as with our decision to submit flexABle to ICCMA'23, we submitted MS-DIS to ICCMA'25 not primarily for its efficiency, but because we consider it valuable to also include systems that use the process of argumentation itself as the basis for reasoning.

The authors are funded by the German Federal Ministry of Research, Technology and Space (BMFTR) via SEMECO (grant no. 03ZU1210B) and KIMEDS (grant no. GW0552B) respectively.




```
m(1,p,pb1,s ← q,r)  m(2,p,pb1,q ← a)  m(3,p,pb1,r ← b)
m(4,o,ob2,b̄ ← q,t)  m(5,o,ob1,t ← c)  m(6,p,pb2,c̄ ← q,d)
```

Fig. 1. Visualisation of a dispute as produced by MS-DIS. The output of MS-DIS is shown at the bottom, while the visualisation is at the top. In the top figure green and yellow boxes indicate proponent and opponent arguments, respectively. Rules appear as subtrees, with heads as roots and bodies as leaves, connected by solid lines. Red arrows mark attacks, assumptions are in boldface, and dashed boxes with numbers indicate the order of rule application in the dispute.

II. An example

Consider the ABA framework $\mathcal{F} = (\mathcal{L}, \mathcal{A}, {}^{-}, \mathcal{R})$, with assumptions $\mathcal{A} = \{a, b, c, d\}$, a contrary relation defined by ${}^{-}(x) = \{\bar{x}\}$ for all $x \in \mathcal{A}$, and the set of rules $\mathcal{R} = \{s \leftarrow q, r; q \leftarrow a;\ r \leftarrow b;\ r \leftarrow c;\ \bar{b} \leftarrow a, t; q \leftarrow a; t \leftarrow c; \bar{c} \leftarrow q, d\}$. A dispute demonstrating the goal statement s to be acceptable with respect to the complete semantics is shown in Figure 1. The top portion of the figure illustrates the arguments implicitly constructed during the dispute, while the bottom part displays the output of MS-DIS at each step.

Each entry in the output has the form m(S, P, T, R), indicating that at step S of the dispute, player P performed a move of type T, involving a rule R. Players are either the proponent (p) or the opponent (o). All moves in this example are backward moves, meaning the player either justifies a previously asserted claim by introducing a rule (types pb1 or ob1), or initiates an attack against a statement introduced by the opposing player (types pb2 or ob2).

III. System description, reasoning tasks

`MS-DIS` consists of (multi-shot) ASP encodings that closely follow the definition of rule-based ABA disputes from [3] to compute possible moves at each step. The second component is a simple Python script that manages multi-shot execution. `MS-DIS` supports both an interactive mode and an automatic mode, the latter being used in the ICCMA competition. Two calls to the ASP solver are needed at each step of a dispute: one to check whether the proponent can make a move that wins the dispute, while the second is to check whether there is a move where the opponent does *not* win. If the first check fails and the second succeeds, the dispute continues.

`MS-DIS` participates in the ABA subtracks addressing the credulous acceptance problem for the admissible and stable semantics. Since credulous acceptance under complete semantics is equivalent to credulous acceptance under the admissible semantics, the simpler encoding for the admissible semantics is used for the complete semantics. The encoding for stable semantics is a slight extension of that for the admissible semantics.

In contrast to `flexABle` [8], `MS-DIS` currently implements only a basic reasoning strategy. Specifically, the version submitted to ICCMA employs a strategy that prioritizes moves which do not involve choices in the search for a winning dispute for the proponent over those that introduce branching.

References

[1] K. Cyras, X. Fan, C. Schulz, and F. Toni, "Assumption-based argumentation: Disputes, explanations, preferences," in *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, and M. Giacomin, Eds., 2018, pp. 365–408.

[2] M. Caminada, "Argumentation semantics as formal discussion," in *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, and M. Giacomin, Eds., 2018, pp. 487–518.

[3] M. Diller, S. A. Gaggl, and P. Gorczyca, "Flexible dispute derivations with forward and backward arguments for assumption-based argumentation," in *CLAR*, ser. LNCS, vol. 13040, 2021, pp. 147–168.

[4] R. Craven and F. Toni, "Argument graphs and assumption-based argumentation," *Artif. Intell.*, vol. 233, pp. 1–59, 2016.

[5] F. Toni, "A generalised framework for dispute derivations in assumption-based argumentation," *Artif. Intell.*, vol. 195, pp. 1–43, 2013.

[6] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot ASP solving with clingo," *Theory Pract. Log. Program.*, vol. 19, no. 1, pp. 27–82, 2019.

[7] T. Lehtonen, J. P. Wallner, and M. Järvisalo, "Declarative algorithms and complexity results for assumption-based argumentation," *J. Artif. Intell. Res.*, vol. 71, pp. 265–318, 2021.

[8] M. Diller, S. A. Gaggl, and P. Gorczyca, "Strategies in flexible dispute derivations for assumption-based argumentation," in *SAFA@COMMA*, ser. CEUR Workshop Proceedings, vol. 3236, 2022, pp. 59–72.

reducto – A Reduct-based Solver for Skeptical Preferred Reasoning

Lars Bengel
Artificial Intelligence Group
University of Hagen
Germany
lars.bengel@fernuni-hagen.de

Julian Sander
Artificial Intelligence Group
University of Hagen
Germany
julian.sander@fernuni-hagen.de

Matthias Thimm
Artificial Intelligence Group
University of Hagen
Germany
matthias.thimm@fernuni-hagen.de

***Abstract*—We present `reducto`, a SAT-based solver for reasoning problems in abstract argumentation. The solver is mainly built on standard SAT-encodings with some improvements. In particular, for skeptical reasoning wrt. preferred semantics it contributes a non-standard approach. `reducto` refrains from iterative maximisation and instead utilises a reduct-based characterisation of preferred semantics to efficiently find counterexamples for the skeptical acceptance of an argument.**

I. Background

An *(abstract) argumentation framework* (AF) is a tuple $F = (A, R)$ where A is a finite set of arguments and R is a relation $R \subseteq A \times A$ [1]. For two arguments $a, b \in A$, the relation aRb means that a attacks b. For a set $S \subseteq A$ we define $S_F^+ = \{a \in A \mid \exists b \in S : bRa\}$. We say that a set $S \subseteq A$ is *conflict-free* iff for all $a, b \in S$ we do not have aRb. A set S *defends* an argument $b \in A$ iff for all a with aRb there is $c \in S$ with cRa. Furthermore, a set S is called *admissible* (AD) iff it is conflict-free and S defends all $a \in S$.

We define different semantics by imposing constrains on admissible sets [2]. In particular, an admissible set $S \subseteq A$ is called a

- *complete* (CO) extension iff for every $a \in A$, if S defends a then $a \in S$,
- *preferred* (PR) extension iff there exists no admissible S' with $S \subsetneq S'$,
- *stable* (ST) extension iff $S \cup S_F^+ = A$,
- *grounded* (GR) extension iff S is complete and there is no complete S' with $S' \subsetneq S$.

For a given argumentation framework $F = (A, R)$ and a semantics $\sigma \in \{\mathsf{CO}, \mathsf{PR}, \mathsf{ST}\}$, we denote with $\sigma(F)$ the set of σ-extensions of F. We consider the following reasoning problems for the above semantics [3]:

DC$-\sigma$	Given an argument $a \in A$, decide whether there exists some σ-extension $E \in \sigma(F)$ with $a \in E$,
DS$-\sigma$	Given an argument $a \in A$, decide whether a is contained in all σ-extensions of F,
SE$-\sigma$	Return a σ-extension of F.

II. System Overview

The `reducto` solver supports the following problems: DC-CO, DC-ST, DS-PR, DS-ST, SE-PR, SE-ST. In general, `reducto` uses the classical SAT-reduction technique to solve these problems [4]–[6]. Per default, `reducto` uses CaDiCaL 2.1.3 [7], but it can be used with any SAT-solver via the IPASIR interface. The source code is open source and available on Github[1].

For the problems DC-CO, DC-ST, DS-ST and SE-ST the solver uses mostly standard SAT-encodings and one-shot queries to the SAT-solver. For DS-PR `reducto` uses novel approaches built on a reduct-based characterisation of preferred semantics which we will outline in the following.

III. Reduct-based Approach to Skeptical Preferred Reasoning

The central notion behind our approach is the S-*reduct* [8].

Definition 1. *Let $F = (A, R)$ be an AF and $S \subseteq A$. We define the S-reduct of F as the AF $F^S = (A', R')$ with*

$$A' = A \setminus (S \cup S_F^+), \qquad R' = R \cap (A' \times A').$$

Essentially, the reduct allows us to remove the part of the AF F that is already "solved" by S. Based on this concept, the notion of *vacuous reduct semantics* has been introduced [9].

Definition 2. *Let σ, τ be argumentation semantics and $F = (A, R)$ is an AF. A set $S \subseteq A$ is a σ^τ-extension iff S is a σ-extension and it holds that $\tau(F^S) \subseteq \{\emptyset\}$.*

Intuitively, a set S is a σ^τ extension of F iff it is σ-extension of F and in the reduct F^S there exists no non-empty τ-extension. We denote with $\sigma^\tau(F)$ the set of all σ^τ-extensions of F. It has been shown, that the preferred semantics can be characterised as a vacuous reduct semantics [9], [10].

Theorem 1. *For any AF $F = (A, R)$. It holds that*

$$\mathsf{PR}(F) = \mathsf{AD}^{\mathsf{AD}}(F) = \mathsf{CO}^{\mathsf{CO}}(F).$$

That means, in order to verify whether a complete extension S is preferred, we construct the reduct F^S and verify that it has no non-empty complete set. Related to that is also the concept of *Modularization* of argumentation semantics [11] which is satisfied by both complete and preferred semantics.

Theorem 2. *Let $F = (A, R)$ be an AF and $\sigma \in \{\mathsf{CO}, \mathsf{PR}\}$. If $S \in \sigma(F)$ and $S' \in \sigma(F^S)$, then $S \cup S' \in \sigma(F)$.*

[1] `https://github.com/aig-hagen/reducto`

In our algorithm, we employ the above properties to simplify the argumentation framework during the computation and then, if necessary, to combine the partial results in the end to obtain the required witness for non-acceptance.

A. Algorithm

The algorithm employed by `reducto` for the DS-PR problem is shown in Algorithm 1. Before starting the main routine, we perform some pre-processing on the AF. That includes restricting the AF to the arguments "relevant" to the query a, similar to how it was outlined in [12]. We also explicitly compute the grounded extension of the AF (line 1), which can be done in polynomial time [3], and remove it from the AF (line 6).

We use a standard SAT-encoding for complete semantics, denoted as Ψ_F^{CO}, whose models correspond to complete extensions [13], [14]. Furthermore, we add a clause to ensure that any model is non-empty, defined as $\Psi_F^{ne} = \bigvee_{a \in A} \mathtt{in}_a$. We write $\textsc{Witness}(\Psi)$ for a call to the SAT-solver that returns the set $\{a \in A \mid \omega(\mathtt{in}_a) = \textsc{True}\}$, where ω is a model of Ψ, if Ψ is satisfiable, otherwise it returns FALSE.

Essentially, our algorithm iterates over the non-empty complete extensions of F that do not include the query. If such an extension S attacks the query it represents a witness for its non-acceptance. Otherwise, we consider the reduct F^S and verify with one additional SAT-call whether there is a non-empty complete extension in F^S. In case there is none, we have found a witness for non-acceptance, otherwise we add a complement clause to the encoding and ask the SAT-solver for another complete extension of F. This complement clause is defined as $\mathcal{C}_F(S) = \bigvee_{a \in A \setminus S} \mathtt{in}_a$ for some set S. This clause, for each found complete extension S, ensures not only that the SAT-solver does not find S again, but also that any S' with $S' \subseteq S$ is no valid witness in any following SAT-call.

In most cases of non-acceptance, `reducto` will return an admissible set S that attacks the query argument as a witness. Note that this implies immediately that there exists a preferred extension S' such that $S \subseteq S'$ with $a \in S_F^+$. Thus such an admissible set is sufficient as a witness for the non-acceptance of a. In case such a set does not exist the solver simply returns a preferred extension S such that $a \notin S$.

Algorithm 1 Algorithm for DS-PR.

Input: $F = (A, R)$, $a \in A$
Output: $S \subseteq A$, otherwise YES

1: $S_{\mathsf{GR}} \leftarrow \textsc{Grounded}(F)$
2: **if** $a \in S_{\mathsf{GR}}$ **then**
3: **return** YES
4: **if** $a \in S^+_{\mathsf{GR},F}$ **then**
5: **return** S_{GR}
6: $F \leftarrow F^{S_{\mathsf{GR}}}$
7: $\Psi \leftarrow \Psi_F^{\mathsf{CO}} \wedge \Psi_F^{ne}$
8: $i \leftarrow 0$
9: **while** TRUE **do**
10: $S \leftarrow \textsc{Witness}(\Psi \wedge \neg \mathtt{in}_a)$
11: **if** $S = \textsc{False}$ **then**
12: **if** $i{+}{+} = 0$ **then**
13: **if** $\textsc{Witness}(\Psi \wedge \mathtt{in}_a)$ **then**
14: **return** YES
15: **else**
16: **return** S_{GR}
17: **return** YES
18: **if** $a \in S_F^+$ **then**
19: **return** $S_{\mathsf{GR}} \cup S$
20: $S' \leftarrow \textsc{Witness}(\Psi_{F^S}^{\mathsf{CO}} \wedge \Psi_{F^S}^{ne})$
21: **if** $S' = \textsc{False}$ **then**
22: **return** $S_{\mathsf{GR}} \cup S$
23: **if** $a \in S'^+_F$ **then**
24: **return** $S_{\mathsf{GR}} \cup S \cup S'$
25: **if** $a \in S'$ **then**
26: $\Psi \leftarrow \Psi \wedge \mathcal{C}_F(S \cup S')$
27: **else**
28: $\Psi \leftarrow \Psi \wedge \mathcal{C}_F(S)$

REFERENCES

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995.

[2] P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, Eds., *Handbook of Formal Argumentation*. College Publications, 2018.

[3] W. Dvorák and P. E. Dunne, "Computational problems in formal argumentation and their complexity," *Handbook of Formal Argumentation*, vol. 1, 2017.

[4] G. Charwat, W. Dvorák, S. A. Gaggl, J. P. Wallner, and S. Woltran, "Methods for solving reasoning problems in abstract argumentation - A survey," *Artif. Intell.*, vol. 220, pp. 28–63, 2015.

[5] F. Cerutti, S. A. Gaggl, M. Thimm, and J. P. Wallner, "Foundations of implementations for formal argumentation," *Handbook of Formal Argumentation*, vol. 1, 2017.

[6] A. Niskanen and M. Järvisalo, "μ-toksia: An efficient abstract argumentation reasoner," in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020*, D. Calvanese, E. Erdem, and M. Thielscher, Eds., 2020, pp. 800–804.

[7] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "Cadical 2.0," in *Computer Aided Verification - 36th International Conference, CAV 2024*, ser. Lecture Notes in Computer Science, A. Gurfinkel and V. Ganesh, Eds., vol. 14681. Springer, 2024, pp. 133–152.

[8] R. Baumann, G. Brewka, and M. Ulbricht, "Revisiting the foundations of abstract argumentation - semantics based on weak admissibility and weak defense," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*. AAAI Press, 2020, pp. 2742–2749.

[9] M. Thimm, "On undisputed sets in abstract argumentation," in *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, B. Williams, Y. Chen, and J. Neville, Eds. AAAI Press, 2023, pp. 6550–6557.

[10] L. Blümel and M. Thimm, "Revisiting vacuous reduct semantics for abstract argumentation," in *ECAI 2024 - 27th European Conference on Artificial Intelligence, 2024*. IOS Press, 2024, pp. 3517–3524.

[11] R. Baumann, G. Brewka, and M. Ulbricht, "Shedding new light on the foundations of abstract argumentation: Modularization and weak admissibility," *Artif. Intell.*, vol. 310, p. 103742, 2022.

[12] B. Liao and H. Huang, "Partial semantics of argumentation: basic properties and empirical," *J. Log. Comput.*, vol. 23, no. 3, pp. 541–562, 2013.

[13] P. Besnard and S. Doutre, "Checking the acceptability of a set of arguments," in *10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, J. P. Delgrande and T. Schaub, Eds., 2004, pp. 59–64.

[14] F. Cerutti, M. Giacomin, and M. Vallati, "How we designed winning algorithms for abstract argumentation and which insight we attained," *Artif. Intell.*, vol. 276, pp. 1–40, 2019.

Scallop at ICCMA'25

Jean-Marie Lagniez
CRIL
Univ. Artois & CNRS
Lens, France
lagniez@cril.fr

Emmanuel Lonca
CRIL
Univ. Artois & CNRS
Lens, France
lonca@cril.fr

Jean-Guy Mailly
IRIT
Université Toulouse Capitole
Toulouse, France
jean-guy.mailly@irit.fr

***Abstract*—SCALLOP takes over from CRUSTABRI, which was itself a rewriting of COQUIAAS. It participates in the same tracks as CRUSTABRI was involved in, but also in the heuristics track and, without restriction, in the ABA track. It can use IPASIR dynamic libraries, which allows it to participate in the fixed-SAT solver comparison.**

***Index Terms*—argumentation, assumption, sat, solver.**

I. INTRODUCTION

SCALLOP takes over from CRUSTABRI [1], which was itself a rewriting of COQUIAAS [2]. It participates in the same tracks that CRUSTABRI was involved in. Regarding the main track, it remains quite similar to what our previous solver proposed. Some shortcuts were found to reduce the number of calls to the SAT solvers for the DS-PR and SE-ID problems. While these changes improve the solver's performance on toy examples, it is unclear if they will help with competition benchmarks.

Regarding the dynamic track, nothing changed except for bug fixes that prevented the solver from being used on some MacOS and Windows machines.

Unlike its predecessor, Scallop enters the heuristics and the ABA tracks (without limitations). It is also designed to use any SAT solver compatible with the IPASIR library, allowing it to participate in the fixed-SAT solver comparison.

II. THE ABA TRACK

The 2023 edition of the competition established ASPFORABA [3] (and, to a lesser extent, ACBAR [4]) as the clear winners in all categories. CRUSTABRI was not sufficiently prepared and suffered from limitations and performance issues. The most significant addition to SCALLOP is its ABA solver, which employs a different approach than ASPFORABA and ACBAR to solve problems.

ASPFORABA and ACBAR translate the input problems into an intermediate problem, that is then translated into a CNF formula used to feed a SAT solver. However, this translation can produce a large CNF formula, which makes it difficult for the SAT solver to determine its satisfiability. Concerning ASPFORABA, the weak point (which did not prevent it from beating its opponents handily in the last competition) could be the grounding step of the ASP solver. Regarding ACBAR, its preprocessing step, which computes a logically equivalent but acyclic framework, may produce quadratic growth of the problem.

SCALLOP proceeds in a lazy manner using a counterexample-guided abstraction refinement (CEGAR) algorithm. The algorithm begins with a CNF formula that encodes a superset of the extensions. This formula has the advantage of being linear in size with respect to the number of atoms and rules. When the SAT solver returns a solution, we verify its correctness. If it is incorrect, we add clauses such that a set of models that do not encode the correct extensions (including the one returned by the solver) are falsified in the formula. This process continues until a model corresponding to an extension is returned, allowing us to learn the correct CNF step by step.

III. THE HEURISTICS TRACK

SCALLOP enters the heuristics track as a dummy shell script that invokes the regular solver with a timeout. If it fails to find a solution, it returns the default value for the current combination of semantics and query.

IV. INTERFACE WITH IPASIR LIBRARIES

SCALLOP is shipped with the same version of CADICAL [5] that was included in CRUSTABRI. However, this SAT engine can be replaced at execution time by providing to SCALLOP a dynamic library of a SAT solver compatible with the IPASIR interface.

REFERENCES

[1] J.-M. Lagniez, E. Lonca, and J.-G. Mailly, "A SAT-based approach for argumentation dynamics," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, M. Dastani, J. S. Sichman, N. Alechina, and V. Dignum, Eds. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024, pp. 2351–2353.

[2] ——, "CoQuiAAS: A constraint-based quick abstract argumentation solver," in *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*. IEEE Computer Society, 2015, pp. 928–935.

[3] T. Lehtonen, J. P. Wallner, and M. Järvisalo, "Declarative algorithms and complexity results for assumption-based argumentation," *J. Artif. Intell. Res.*, vol. 71, pp. 265–318, 2021.

[4] T. Lehtonen, A. Rapberger, M. Ulbricht, and J. P. Wallner, "Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity," in *International Conference on Principles of Knowledge Representation and Reasoning*. International Joint Conferences on Artificial Intelligence, 2023, pp. 440–450.

[5] A. Fleury and M. Heisinger, "Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020," *Sat Competition*, vol. 2020, p. 50, 2020.

SMART V1.0

Sandra Hoffmann
Artificial Intelligence Group
University of Hagen
Germany
sandra.hoffmann@fernuni-hagen.de

Isabelle Kuhlmann
Artificial Intelligence Group
University of Hagen
Germany
isabelle.kuhlmann@fernuni-hagen.de

Matthias Thimm
Artificial Intelligence Group
University of Hagen
Germany
matthias.thimm@fernuni-hagen.de

***Abstract*—We present SMART V1.0, a backtracking-based Solver using MAchine learning to complete Reasoning Tasks in abstract argumentation. Specifically, we use a graph neural network to predict co-admissible arguments to a given query argument in order to guide a backtracking-based search algorithm.**

I. Introduction

An *abstract argumentation framework* (AF) [1] is a tuple $F = (\mathsf{Args}, R)$, with Args being a set of arguments and an attack relation $R \subseteq \mathsf{Args} \times \mathsf{Args}$. An argument $a \in \mathsf{Args}$ *attacks* another argument $b \in \mathsf{Args}$ if $(a, b) \in R$. An argument $a \in \mathsf{Args}$ is *defended by* a set of arguments $E \subseteq \mathsf{Args}$ if for all $b \in \mathsf{Args}$ with $(b, a) \in R$, there exists a $c \in E$ with $(c, b) \in R$.

Let $F = (\mathsf{Args}, R)$ be an argumentation framework. A set $E \subseteq \mathsf{Args}$ is

- *conflict-free* if there are no $a, b \in E$ such that $(a, b) \in R$,
- *admissible* if E is conflict-free and each $a \in E$ is defended by E within F,
- *complete* if every argument $a \in \mathsf{Args}$ defended by E is also included in E,
- *preferred* if E is a $\subseteq$-maximal complete extension, and
- *grounded* if E is a $\subseteq$-minimal complete extension.

Two arguments a and $b \in \mathsf{Args}$ are called *co-admissible* if there exists an *admissible* set S such that $\{a, b\} \subseteq S$.

A typical decision problem in the area of abstract argumentation is the problem of deciding whether a given argument is included in at least one extension (*credulous acceptability*) wrt. a given semantics.

Instead of defining argumentation semantics using set theory, we can also use the concept of labelings [2]. A labeling is a total function $L : \mathsf{Args} \to \{in, out, undec\}$ We say that a labeling is complete if it holds that for every $a \in \mathsf{Args}$:

- if a is labeled *in* then all attackers of a are labeled *out*
- if all attackers of a are labeled *out* then a is labeled *in*
- if a is labeled *out* then there is an attacker of a that is labeled *in*
- if a has an attacker that is labeled *in* then a is labeled *out*

In addition to these three labels the SMART solver also uses the labels *must out* as well as *must undec* to mark arguments that have to be set to *out* or *undec*, respectively, at some point during the solution process in order to build a maximal admissible labeling.

Computing such a labeling is a computationally hard problem [3]. Thus, several previous works are concerned with using machine learning-based approaches in order to reduce the runtime, even though the results are not guaranteed to be correct [4]–[7]. Most approaches concentrated on training a model to directly predict the acceptance status of a given argument. The author in [8] used a similar approach to the one presented here, where a GCN was trained on predicting co-admissible arguments to be used as a heuristic combined with a SAT solver.

Instead of using a SAT solver, we employ a dedicated backtracking algorithm and use a prediction, generated by a graph convolutional network (GCN), to guide the search by identifying arguments most likely to be jointly admissible with the query argument.

The SMART V1.0 solver supports solving the credulous acceptability problem for preferred (DC-PR), complete (DC-CO) and grounded (DC-GR) semantics. In the remainder of this description we give an overview on the architecture of the GCN we use for our predictions (Section II) as well as on the backtracking algorithm (Section III). We conclude in Section IV.

II. Architecture of the GCN

The SMART solver implements a specialized GCN that predicts which arguments are co-admissible wrt. a given query argument. We use PyTorch Geometric[1] to implement the GCN. The network consists of a 3-layer architecture:

1) An input GCN layer that transforms node features from the input dimension to 64 features
2) A hidden GCN layer that processes the 64-dimensional representations to 16 features
3) A final linear layer that reduces the 16 features to a single output, followed by a sigmoid activation for binary prediction, where a value of 1 indicates that an argument is predicted to be co-admissible with the query node.

Each GCN layer is followed by ReLU activation, and dropout with a probability of $p = 0.2$ is applied after the first convolutional layer to prevent overfitting.

[1] https://pytorch-geometric.readthedocs.io/en/latest/

For node features, we use each argument's in-degree and out-degree. The query node is represented as a one-hot encoded vector that is concatenated with these features.

During training, we use a Binary Cross-Entropy loss function and an Adam optimizer [9] with a learning rate of 0.01. Ground truth labels for training are provided as a vector where arguments co-admissible with the query argument are assigned a value of 1. During inference, we apply a threshold of 0.7, based on the output of the sigmoid function in the linear layer, to determine if an argument is co-admissible with the query argument.

III. Backtracking Algorithm

The algorithm for determining credulous acceptance of a query argument under preferred or complete semantics follows the approach presented by Nofal et al. in [10]. The authors propose a backtracking-based approach to justify the acceptance status of an argument by employing *global look-ahead pruning strategies*, such as terminating the construction of a labeling early when it becomes evident that it cannot develop into an admissible labeling. We further implemented several enhancements described by Nofal et al. in [11], particularly maintaining separate lists for each argument that track how many of its attackers are labeled *blank* or *undec*, thereby enabling efficient assessment of whether a current labeling will yield a maximal admissible labeling.

Initially, we compute the grounded extension using the algorithmic approach described by the authors in [12]. This grounded extension is incorporated into the initial labeling, with all arguments in the grounded extension labeled *in* and all arguments attacking or attacked by grounded arguments labeled *out*.

In the initial labeling phase, we label the query argument as *in*, all arguments attacking the query argument as *must out*, and all arguments attacked by the query argument as *out*. Self-attacking arguments are assigned the label *undec*.

We then proceed by propagating the labeling through the framework, checking whether any *blank* arguments need to be set to *in* (e.g., when all attackers of a blank argument are labeled *out* or *must out*). When such an argument is identified, all its neighbors are set to *out*. This propagation continues until no further *blank* arguments requiring the *in* label are found.

Proceeding from this initial labeling, we select an argument to continue the algorithm. While the authors in [10] accomplished this by identifying an argument that attacks a *must out* labeled argument and possesses the highest number of neighbors, the SMART solver utilizes the GCN described in Section II to guide the search. When the first new argument is required, the system generates a prediction of arguments co-admissible with the query argument. Subsequently, we select a *blank* argument from these predicted co-admissible arguments that attacks a *must out* argument. If no such argument exists—indicating either limited utility of the prediction or that the argument is predicted to be inadmissible—we revert to the argument selection approach outlined in [10].

The algorithm then branches by exploring two possibilities: setting the selected argument to *in* or to *undec*, and propagating the resulting label changes. This process continues until either a hopeless labeling is reached (e. g., containing a *must out* argument with no remaining *blank* arguments that could attack it) or a terminal labeling is found (no *blank* labeled arguments are attacked by *must out* labeled arguments). Hopeless labelings result in backtracking, while terminal labelings are checked for admissibility (absence of *must out* arguments), which would confirm the existence of a preferred labeling containing the query argument.

IV. Summary

We present SMART V1.0, a backtracking-based **S**olver using **MA**chine learning to complete **R**easoning **T**asks in abstract argumentation. Specifically, we use a graph neural network to predict co-admissible arguments to a given query argument in order to guide a backtracking-based search algorithm.

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*, vol. 77, no. 2, pp. 321–358, 1995.

[2] M. W. A. Caminada and D. M. Gabbay, "A logical account of formal argumentation," *Studia Logica*, vol. 93, no. 2, p. 109, 2009. [Online]. Available: https://doi.org/10.1007/s11225-009-9218-x

[3] W. Dvořák and P. E. Dunne, "Computational problems in formal argumentation and their complexity," in *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, Eds. College Publications, February 2018, ch. 14.

[4] D. Craandijk and F. Bex, "Enforcement heuristics for argumentation with deep reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 5, pp. 5573–5581, Jun. 2022. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20497

[5] I. Kuhlmann and M. Thimm, "Using graph convolutional networks for approximate reasoning with abstract argumentation frameworks: A feasibility study," in *International Conference on Scalable Uncertainty Management*. Springer, 2019, pp. 24–37.

[6] D. Craandijk and F. Bex, "Deep learning for abstract argumentation semantics," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020, p. 1667–1673.

[7] L. Malmqvist, T. Yuan, P. Nightingale, and S. Manandhar, "Determining the acceptability of abstract arguments with graph convolutional networks." in *SAFA@ COMMA*, 2020, pp. 47–56.

[8] L. Malmqvist, "Approximate solutions to abstract argumentation problems using graph neural networks," Ph.D. dissertation, University of York, 2022.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

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

[11] S. Nofal, K. Atkinson, P. E. Dunne, and I. O. Hababeh, "A new labelling algorithm for generating preferred extensions of abstract argumentation frameworks." in *ICEIS (1)*, 2019, pp. 340–348.

[12] S. Nofal, K. Atkinson, and P. E. Dunne, "Computing grounded extensions of abstract argumentation frameworks," *The Computer Journal*, vol. 64, no. 1, pp. 54–63, 11 2019. [Online]. Available: https://doi.org/10.1093/comjnl/bxz138

3 Benchmark Descriptions

ABCGEN – Generator for Assumption-based Argumentation Frameworks with a Clustered Structure

Andreas Niskanen
University of Helsinki
Finland

Masood Feyzbakhsh Rankooh
University of Helsinki
Finland

Tuomo Lehtonen
Aalto University
Finland

Matti Järvisalo
University of Helsinki
Finland

***Abstract*—ABCGEN is a generator for ABAFs with a clustered structure, inspired by StableGenerator for AFs. The aim is to be able to generate instances that admit multiple stable extensions (and by extension complete and preferred ones) and having moderate cyclicity in the rules. These qualities make the instances challenging for solvers.**

I. INTRODUCTION

In this system description, we give an overview on ABCGEN, a benchmark generator for assumption-based argumentation (ABA) frameworks [1]. The generator is submitted for the first time to the dedicated ABA track of the sixth International Competition on Computational Models of Argumentation (ICCMA 2025).

We propose a scheme to generate ABAFs that admit multiple stable assumption sets by partitioning the assumptions of the ABAF and having the assumptions of each partition attack the majority of assumptions outside of it. Our aim is to have many stable (and thus complete and preferred) assumption sets while having acyclicity in the rules. The scheme is based on ideas from the existing abstract argumentation benchmark generator StableGenerator [2], [3].

II. GENERATOR ARCHITECTURE

We have the following parameters: the number of partitions/clusters P, the numbers of assumptions $P_{\mathcal{A}}$ and non-assumption atoms $P_{\mathcal{L}}$ in each partition, the maximum number of rules per atom mra, the maximum rule size mrs, and a contrary probability C. For each i in $1, \ldots, P$, let $\mathcal{A}_i$ be a set of $P_{\mathcal{A}}$ fresh assumptions for partition i, let each assumption in $\mathcal{A}_i$ have a fresh contrary atom, and let $\mathcal{L}_i$ be a set of $P_{\mathcal{L}}$ fresh non-assumption atoms.

The generator works as follows. For each $x \in \mathcal{L}_i$, generate $r \in [1, mra]$ rules for x of size $s \in [1, mrs]$ (r and s selected uniformly at random) such that the rule body consists of s atoms from $\mathcal{L}_i \cup \mathcal{A}_i$. Finally, for each assumption $a \in \mathcal{A} \setminus \mathcal{A}_i$, generate a rule $(\overline{a} \leftarrow x)$ for a randomly selected $x \in \mathcal{L}_i$ with probability C. As a result, ABAFs from ABCGEN cannot have a strongly connected component (SCC) in the underlying rule graph larger than $P_{\mathcal{L}}$, but many of the atoms within each cluster will belong to the same SCC. There are no attacks within the partitions. With C close to 1, the assumptions in each partition attack most assumptions outside the partition. This results in an ABAF with many stable extensions: in particular, if all atoms are derivable, then with $C = 1$ each partition by itself is a stable extension.

III. SUGGESTED PARAMETER VALUES

All of the parameters mentioned can be set by the user. In addition, one can set a random seed (or simply log the random seed used by a particular run of the generator) in order to reproduce the same benchmark set in the future.

In initial tests we have found that with $mra = 3$ and $mrs = 2$ and $P_{\mathcal{A}}$ not much smaller than $P_{\mathcal{L}}$ virtually all atoms within each partition are derivable, and the size of SCCs typically equal roughly half or more $P_{\mathcal{L}}$. In particular, we have tested two sets of benchmarks from ABCGEN, one with a larger number of small clusters, and one with a smaller number of large clusters. We let $C = 0.8$ or $C = 0.9$, and in one set let P, $P_{\mathcal{A}}$ and $P_{\mathcal{L}}$ take a value from $\{25, 30, 35\}$, and in the other set let $P \in \{5, 7\}$ and let $P_{\mathcal{A}}$ and $P_{\mathcal{L}}$ take a value from $\{100, 150, 200\}$. We observed desired effects: reasonably hard instances for common reasoning problems (i.e. the state-of-the-art ABA solver ASPFORABA [4] usually takes multiple seconds to solve an instance and takes more than 900 seconds on multiple instances) with many stable extensions, almost all atoms being derivable, and some cyclicity occurring in rules.

Beyond these concrete values which we have tested, hard instances with the desired properties can be generated with different combinations. For example, having the number of clusters P somewhere between 7 and 25, and $P_{\mathcal{A}}$ and $P_{\mathcal{L}}$ somewhere between 35 and 100 would likely generate similarly interesting instances. Furthermore, different values of mra, mrs and C likely work as well. The lower C is, the fewer stable extensions there is expected to be, and thus a lower value of C might offer an interesting comparison to the instances with a high number of stable extensions, resulting in the values described above.

For acceptance problems, it might make sense to separately query an assumption, a contrary atom, and an atom that is neither an assumption nor a contrary. The reason is that different atom types might behave differently due to the structure of the ABAFs: contrary atoms are derivable from multiple clusters while other atoms only from a single cluster. Thus,

for example, skeptical acceptance for atoms or assumptions might be easier than for contraries.

ACKNOWLEDGMENTS

This work has been financially supported by Research Council of Finland (under grants 347588 and 356046), and Helsinki Institute for Information Technology HIIT. The authors thank the Finnish Computing Competence Infrastructure (FCCI) for computational and data storage resources.

REFERENCES

[1] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni, "An abstract, argumentation-theoretic approach to default reasoning," *Artificial Intelligence*, vol. 93, pp. 63–101, 1997.

[2] F. Cerutti, N. Oren, H. Strass, M. Thimm, and M. Vallati, "A benchmark framework for a computational argumentation competition," in *Proc. COMMA*, ser. FAIA, S. Parsons, N. Oren, C. Reed, and F. Cerutti, Eds., vol. 266. IOS Press, 2014, pp. 459–460.

[3] M. Thimm and S. Villata, "The first international competition on computational models of argumentation: Results and analysis," *Artificial Intelligence*, vol. 252, pp. 267–294, 2017.

[4] T. Lehtonen, J. P. Wallner, and M. Järvisalo, "Declarative algorithms and complexity results for assumption-based argumentation," *Journal of Artificial Intelligence Research*, vol. 71, pp. 265–318, 2021.

Argumentation benchmarks from the wild

Daphne Odekerken
National Police Lab AI, Netherlands Police
Department of Information and Computing Sciences, Utrecht University
d.odekerken@uu.nl

***Abstract*—We present a benchmark generator for abstract argumentation frameworks (AFs) and assumption-based argumentation frameworks (ABAFs). These AFs and ABAFs are derived from ASPIC$^+$ argumentation theories that are similar in structure to the argumentation theories used in a real-life application at the Dutch police.**

***Index Terms*—Benchmark generator, applications, abstract argumentation, assumption-based argumentation**

I. Background

One of the applications of computational argumentation in law enforcement is a system for the intake of police reports on online trade fraud [1]. This system advices citizens whether or not they should submit the report to the police. The advice is obtained automatically by reasoning with argumentation theories in the ASPIC$^+$ formalism. These argumentation theories contain a set of defeasible rules, which model the definition of online trade fraud, and a knowledge base with evidence provided by the citizen.

The set of rules in the argumentation theories has a layered structure: there is one central high-level rule, which gives a general definition of fraud, in combination with rules that further specify the elements of this definition and rules for exceptions. There are 43 rules in total. More details are given in Table 3 of [1].

In the future, we would like to develop applications for more general topics in the law enforcement domain than just online trade fraud. This requires algorithms that can efficiently reason with larger ASPIC$^+$ argumentation theories. We expect that the rule sets for such domains have a similar layered structure. Therefore we developed a script for randomly generating layered argumentation theories of a specified size.

II. Properties of the generated argumentation theories

The generator uses PyArg [2] to randomly generate ASPIC$^+$ argumentation theories with a given number of literals. Each literal is assigned a layer, which informally is the largest number of rule applications to reach this literal from another literal. The notion of literal layers is illustrated in Figure 1. For example, the literal l_1 is on Layer 0, as there is no rule for l_1. Given that the only rule for l_2 is $l_1 \Rightarrow l_2$, the literal l_2 is on Layer 1. Even though l_3 can be derived from $\neg l_0, \neg l_2 \Rightarrow l_3$ (and both $\neg l_0$ and $\neg l_2$ are on Layer 0), it is on Layer 2 since there is a rule $l_2 \Rightarrow l_3$ and l_2 is on Layer 1.

The literal layer distribution is selected to have $\frac{2}{3} \cdot |\mathcal{L}|$ literals with layer 0, each one-tenth of the literals for Layers 1, 2 and 3 and the remaining literals in Layer 4. This distribution is similar to the distribution of the fraud intake application.

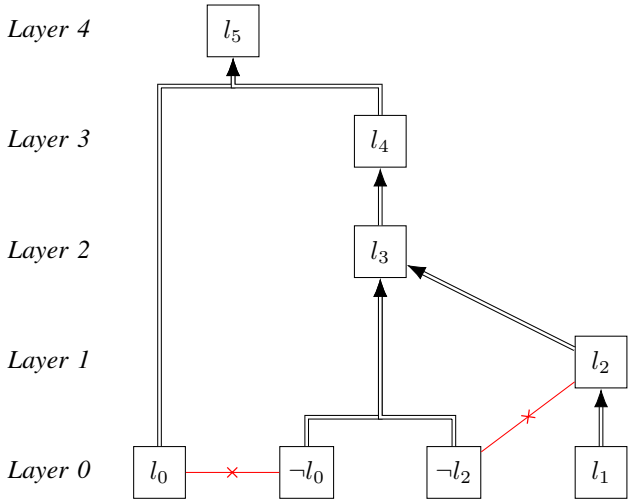


Fig. 1. Sample of a generated rule set, illustrating the layers of literals. Defeasible rules are represented by double-lined arrows and contradicting literals by a red line with a cross in the middle. For readability, not all literals and rules of the argumentation theory are visualised.

After assigning a layer to each literal, the defeasible rules are generated in such a way that they establish this distribution of layers. Just like in the application for fraud intake, there are no strict rules and no ordinary premises. The number of rules is 1.5 times the number of literals. Each rule has at least 1 and at least 5 antecedents.

Contrariness is based on classical negation. Each literal in the language has only its negation as its contradictory. Finally, a consistent set of axioms is randomly selected from the literals with Layer 0.

These steps result in ASPIC$^+$ argumentation theories that are similar in structure to the application at the Dutch police. In order to convert these into instances for ICCMA, we next convert these into AFs as well as ABAFs.

III. Conversion into AFs

Any ASPIC$^+$ argumentation theory can be converted into an abstract argumentation framework by listing all arguments inferred by the argumentation theory (following [3] Definition 5) and identifying all attacks between those arguments. For converting the generated ASPIC$^+$ argumentation theories to AFs we provide a script based on PyArg [2]. Note that

all attacks are rebuttals, because the argumentation theory does not contain ordinary premises (which are required for undermining attacks) or rules contradicting rule applications (which are required for undercutting attacks). These AFs are typically small in size compared to the input argumentation theory. An explanation for this is that many rules do not participate in any argument, as some of their antecedents are not axioms, nor conclusions of any other rule-based argument. This is also in line with the aforementioned application in fraud intake, where it depends on the evidence provided by the citizen which rules apply.

IV. Conversion into ABAFs

Alternatively, the ASPIC$^+$ argumentation theory can be converted into an ABAF. For this conversion we provide a script that implements the method from Definition 24 of [4]. It replaces every defeasible rule from the ASPIC$^+$ argumentation theory by two rules in the ABAF and adds an assumption that this defeasible rule holds. For instance, the ASPIC$^+$ rule $r_0 : l_0, l_4 \Rightarrow l_5$ is replaced by the ABA rules $r_0\_\textit{holds}, l_0, l_4 \rightarrow l_5$ and $\neg l_5 \rightarrow \neg r_0\_\textit{holds}$. Furthermore, the ABAF contains the assumption $r_0\_\textit{holds}$.

If the ASPIC$^+$ framework had $|\mathcal{L}|$ literals, then the ABAF has $4 \cdot |\mathcal{L}|$ atoms because the ASPIC$^+$ framework with $|\mathcal{L}|$ literals is created with $1.5 \cdot |\mathcal{L}|$ rules and the conversion to ABA introduces two new atoms for every ASPIC$^+$ rule. If the ASPIC$^+$ framework had $|\mathcal{R}|$ rules, then the ABA framework has $2 \cdot |\mathcal{R}|$ rules. Finally, note that the ABA framework is flat: no assumption is the consequent of a rule.

V. Usage instructions

The Python package, containing the aforementioned scripts for generating ASPIC$^+$ argumentation theories and converting them to AFs and ABAFs, can be found on GitHub.[1] The README of this repository contains usage instructions.

VI. Discussion

We developed the generator in such a way that it creates ASPIC$^+$ argumentation theories that are similar in structure to the argumentation theory in an actual application at the Dutch police. In addition, we developed scripts for converting these argumentation theories to AFs and ABAFs, which can be used for testing solvers in the ICCMA competition.

Given the small size of the AFs, we expect that solving the problems considered in ICCMA (i.e., identifying extensions and accepted arguments under a given semantics) for these instances does not require state-of-the-art solvers. However, it should be noted that the application in fraud intake required solving the more challenging argumentation problems of identifying stability [1] and relevance [5], to find out whether (and which) additional knowledge could still change the acceptance of specific literals. It would be interesting to consider these dynamic argumentation problems in future ICCMA editions.

[1] https://github.com/DaphneOdekerken/LayeredGeneratorICCMA

References

[1] D. Odekerken, F. Bex, A. Borg, and B. Testerink, "Approximating stability for applied argument-base inquiry," Intelligent Systems with Applications, Vol. 16, pp 200110, 2022.

[2] D. Odekerken, A. Borg, and M. Berthold, "Accessible algorithms for applied argumentation," Proc. of Arg&App, pp. 92–98, 2023.

[3] S. Modgil, and H. Prakken, "A general account of argumentation with preferences," Artificial Intelligence, Vol. 195, pp 361–397, 2013.

[4] J. Heyninck, and C. Strasser, "Relations between assumption-based approaches in nonmonotonic logic and formal argumentation," Proc. of NMR, pp 65–75, 2016.

[5] D. Odekerken, T. Lehtonen, A. Borg, J.P. Wallner, and M. Järvisalo, "Argumentative reasoning in ASPIC+ under incomplete information," Proc. of KR, pp 531–541, 2023.

Argumentation Framework Subsampling Generator

Lars Malmqvist
The Tech Collective
Denmark
lama@thetechcollective.eu

***Abstract*—This paper introduces benchmark abstract argumentation frameworks generated via systematic subsampling of existing instances. Four distinct methods are employed: random, degree-based, Breadth-First Search (BFS), and community-based subsampling. These techniques yield derived frameworks that preserve specific structural properties of the source frameworks while reducing their scale, thereby generating a spectrum of computational challenges for evaluating argumentation solvers.**

I. Introduction

Reasoning tasks over Abstract Argumentation Frameworks (AFs) [1], such as determining extensions under various semantics, are computationally demanding. Evaluating and advancing solver performance necessitates diverse benchmark suites. This work contributes such a suite by generating derived AFs through systematic subsampling of source frameworks. This approach yields instances with controlled structural properties, reflecting characteristics of real-world argumentation structures while varying in scale and complexity. This is particularly useful in machine learning applications, where large amounts of diverse but related samples can help approximation accuracy [2].

II. Subsampling Methodology

We implemented four distinct subsampling methods. Given a source $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ and a proportion parameter $p \in (0, 1]$, a subsampled $\mathsf{AF}' = (\mathsf{A}', \mathsf{R}')$ is generated where $|\mathsf{A}'| \approx k = \lceil p \cdot |\mathsf{A}| \rceil$ and $\mathsf{R}' = \{(a, b) \in \mathsf{R} \mid a, b \in \mathsf{A}'\}$. The methods are:

A. Method 1: Random Subsampling

Random subsampling selects a subset $\mathsf{A}' \subseteq \mathsf{A}$ of size k uniformly at random. This baseline method is computationally efficient but may disrupt significant structural features of the original AF, such as attack chains or central conflicts. Formally:

$$\mathsf{A}' = \text{Sample}(\mathsf{A}, k) \tag{1}$$

where $\text{Sample}(S, n)$ returns n elements chosen uniformly at random from set S. The attack relation R' is induced by A'.

B. Method 2: Degree-based Subsampling

This method selects arguments based on their connectivity, hypothesizing that high-degree arguments are structurally significant. Arguments with high in-degree or out-degree often represent critical points within the argumentative structure. Optionally, extension membership frequency can be incorporated. A score is assigned:

$$\text{score}(a) = \text{in-degree}(a) + \text{out-degree}(a) + \alpha \cdot \text{extension-count}(a) \tag{2}$$

where extension-count(a) is the frequency of a in preferred extensions (if available, weighted by α). Arguments are selected greedily based on score:

$$\mathsf{A}' = \{a_1, ..., a_k\} \text{ s.t. } \text{score}(a_i) \geq \text{score}(a_{i+1}) \forall i \tag{3}$$

This method tends to preserve dense regions, potentially increasing instance difficulty.

C. Method 3: BFS (Breadth-First Search) Subsampling

BFS-based subsampling, akin to snowball sampling, preserves local topological structure by exploring the neighborhood around a seed argument. It aims to maintain connected components and attack/defense chains, which are crucial in argumentation dynamics. Unlike random or degree-based sampling, BFS ensures the preservation of local connectivity. The process starts from a randomly selected seed $a_0 \in \mathsf{A}$.

Algorithm 1 BFS Subsampling

```
Select random a_0 ∈ A
Initialize A' = {a_0}, Q = [a_0] (Queue)
while |A'| < k and Q ≠ ∅ do
    Dequeue a from Q
    for each b s.t. (a, b) ∈ R or (b, a) ∈ R do
        if b ∉ A' then
            A' ← A' ∪ {b}; Enqueue b in Q
            if |A'| = k then break
return A'
```

The algorithm explores both incoming and outgoing attacks to capture the argument's local context.

D. Method 4: Community-based Subsampling

Real-world AFs often exhibit community structure (dense intra-group connections, sparse inter-group connections), representing distinct sub-topics or perspectives. This method preserves this modular organization. Community detection (e.g., using the Louvain method on the underlying undirected graph derived from R) identifies communities $C_1, ..., C_m$. Arguments are then sampled proportionally from each community.

This method contrasts with random (structure-agnostic) and BFS (local focus) sampling by maintaining the global organization of the AF.

Algorithm 2 Community-based Subsampling

Detect communities $C_1, ..., C_m$ in AF (e.g., via modularity optimization)
Initialize $\mathsf{A}' = \emptyset$
for each community C_i **do**
    Compute quota $q_i = \lceil \frac{|C_i|}{|\mathsf{A}|} \cdot k \rceil$
    $S_i = \text{Sample}(C_i, \min(q_i, |C_i|))$
    $\mathsf{A}' \leftarrow \mathsf{A}' \cup S_i$
return A'

III. Implementation

A Python implementation is provided, processing argumentation frameworks in the standard .af format. It comprises a library (`subsampling_lib.py`) implementing the core methods using NetworkX for graph operations, and a command-line tool (`af_subsample.py`) for batch generation. The tool processes source directories, applies selected subsampling methods with configurable parameters (e.g., proportion p, number of samples per method), and produces structured output directories containing the derived AFs in .af format. This modular design supports both specific generation tasks and large-scale benchmark creation, with reusable library components.

IV. Benchmark Characteristics

The generated benchmarks offer controlled variation:

A. Size and Structural Properties

1) **Scalability**: Variable sizes ($|\mathsf{A}'| \approx p \cdot |\mathsf{A}|$) allow testing solver scalability.
2) **Structural Diversity**: Each method preserves distinct structural aspects: random (baseline), degree-based (hubs/centrality), BFS (local topology/chains), community-based (global modularity).
3) **Difficulty Gradient**: The combination of methods and proportions p yields instances spanning a range of computational complexities.

B. Empirical Observations

Preliminary empirical evaluation using standard argumentation solvers indicates relative difficulty patterns:

- Degree-based sampling (especially for $p > 0.7$) tends to produce the most computationally challenging instances, likely due to the preservation of dense, interconnected subgraphs.
- Random sampling generally results in easier instances, attributed to the disruption of complex attack structures.
- BFS sampling yields instances of intermediate difficulty, potentially varying with the seed choice.
- Community-based sampling tends to scale the difficulty relative to the original framework's structure and the sampling proportion p.

These observations align with the theoretical understanding that structural properties, particularly cycles and dense connectivity preserved by methods like degree-based sampling, significantly impact the complexity of argumentation reasoning.

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*, vol. 77, no. 2, pp. 321–358, 1995.

[2] Malmqvist, L., Yuan, T., and Nightingale, P. (2024). Approximating Problems in Abstract Argumentation with Graph Convolutional Networks. *Artificial Intelligence*, 336, 104209.

The KWT Benchmark Generator for ICCMA 2025

Isabelle Kuhlmann
Artificial Intelligence Group
University of Hagen
Germany
isabelle.kuhlmann@fernuni-hagen.de

Matthias Thimm
Artificial Intelligence Group
University of Hagen
Germany
matthias.thimm@fernuni-hagen.de

***Abstract*—We present a generator for abstract argumentation frameworks that produces instances which are particularly challenging for the task of deciding skeptical acceptability w.r.t. preferred semantics.**

I. Introduction

Over the past years, the International Competition on Computational Models of Argumentation (ICCMA)[1] has significantly contributed to the evaluation of argumentation systems. Nevertheless, the process of generating benchmarks for different argumentation formalisms remains a perpetual task that possesses numerous challenges (see [1] for a discussion).

With the *KWT Benchmark Generator*, we focus on generating benchmarks for problems related to abstract argumentation frameworks [2]; more precisely, our aim is to provide challenging benchmarks for the task of deciding skeptical acceptability w.r.t. preferred semantics. Although this task is generally Π_2^P-complete [3], in practice it can often be solved much more efficiently by exploiting some "shorcuts". For instance, each argument in the grounded extension (which can be computed in polynomial time) and each argument in the ideal extension (whose corresponding problems are Θ_2^P-complete) is also skeptically accepted w.r.t. preferred semantics. W.r.t. a set of ICCMA'17 benchmarks, it has been pointed out that a majority of arguments that are skeptically accepted under preferred semantics is also included in the grounded extension [4]. Thus, in the majority of cases, it is sufficient to solve a less complex problem, which may distort the interpretation of experimental results. To address this problem, we propose the use of the *KWT Benchmark Generator*[2], which is designed to circumvent the aforementioned problem.

II. Preliminaries

An (abstract) *argumentation framework* (AF) [2] is a pair $F = (\mathsf{Arg}, R)$, with Arg being a set of arguments and $R \subseteq \mathsf{Arg} \times \mathsf{Arg}$ a relation between those arguments. An argument $a \in \mathsf{Arg}$ *attacks* an argument $b \in \mathsf{Arg}$ if $(a, b) \in R$. Moreover, we define the set of arguments attacking a given argument a as $a_F^- = \{b \mid (b, a) \in R\}$, and the set of arguments being attacked by a as $a_F^+ = \{b \mid (a, b) \in R\}$. In the same fashion we define E_F^- and E_F^+ for a set $E \subseteq \mathsf{Arg}$. We call an argument $a \in \mathsf{Arg}$ *defended* by a set of arguments $E \subseteq \mathsf{Arg}$ if every argument $b \in \mathsf{Arg}$ that attacks a is itself attacked by some argument $c \in E$, i.e., if $a_F^- \subseteq E_F^+$.

Further, a set $E \subseteq \mathsf{Arg}$ is *conflict-free* if $E \cap E_F^+ = \emptyset$. If a set $E \subseteq \mathsf{Arg}$ is conflict-free and each $a \in E$ is defended by E, we call E admissible (ad). We call sets of jointly acceptable arguments *extensions*, which can be defined under various semantics. The classical semantics, following the seminal work by Dung [2], are defined as follows:

- A set $E \subseteq \mathsf{Arg}$ is *complete* (co) iff it is admissible, and if E defends $a \in \mathsf{Arg}$ then $a \in E$.
- A set $E \subseteq \mathsf{Arg}$ is *grounded* (gr) iff E is complete and $\subseteq$-minimal.
- A set $E \subseteq \mathsf{Arg}$ is *preferred* (pr) iff E is complete and $\subseteq$-maximal.
- A set $E \subseteq \mathsf{Arg}$ is *stable* (st) iff it is complete and $E \cup E_F^+ = \mathsf{Arg}$.

In addition, we define a set $E \in \mathsf{Arg}$ to be an *ideal* (id) extension [5] if E is admissible, for every preferred extension E', it holds that $E \subseteq E'$, and E is $\subseteq$-maximal with these two properties. Note that the grounded and the ideal extension of an AF are each a uniquely defined, and that the former is always a subset of the latter.

III. The KWT Benchmark Generator

The main idea behind the *KWT Benchmark Generator* is to avoid (to a high degree) producing argumentation frameworks that are "easy" to solve, i. e., instances in which arguments that are skeptically accepted under preferred semantics are also in the grounded or the ideal extension. Another "easy" case regarding this task occurs when arguments are attacked by some admissible set—such arguments are never skeptically accepted w.r.t. pr—and deciding this is a problem in NP. The generator takes the parameters

- num_{args}: the total number of arguments,
- num_{pa}: the number of arguments to be skeptically accepted under preferred semantics,
- num_{cred}: the number of arguments to be contained in at least one preferred extension,
- num_{pref}: the number of preferred extensions,
- num_{ideal}: the number of arguments in the ideal extension,

in addition to 7 further parameters that control the probability of attacks between different sets of arguments. More precisely,

[1] http://argumentationcompetition.org/index.html

[2] Note that this generator was first used in [4] and later described in more detail in [4].

these parameters set the probabilities of arguments in the ideal extension to be attacked and to attack back, respectively, the probabilities of credulously accepted arguments to be attacked and to attack back, the probabilities of skeptically accepted arguments that are not contained in the ideal extension to be attacked and to attack back, and the probability of further random attacks between unaccepted arguments. Given these parameters, a random AF F is generated as follows:

1) The set Arg of num_{args} arguments is created and arguments are associated to sets S_{pa} (skeptically accepted arguments w.r.t. preferred semantics), S_{ideal} (arguments in the ideal extension), S_{cred} (arguments that are credulously accepted w.r.t. preferred semantics), S_{unacc} (arguments that are not credulously accepted w.r.t. preferred semantics), such that $S_{ideal} \subseteq S_{pa} \subseteq S_{cred}$, $S_{cred} \cup S_{unacc} = \mathsf{Arg}$, and the corresponding cardinalities are respected. Finally, sets $E_1, \ldots, E_{num_{pref}}$ (the preferred extensions) are created by adding all arguments from S_{pa} and randomly drawn arguments from $S_{cred} \setminus S_{pa}$.
2) For every argument $a \in S_{ideal}$, random attackers from S_{unacc} are sampled. For each of these attackers b, another argument from S_{ideal} is sampled that attacks b. This ensures that the grounded extension will be empty and that the ideal extension is capable of defending itself (thus forming an admissible set).
3) For every argument $a \in S_{pa} \setminus S_{ideal}$, attacks from unaccepted arguments are sampled in a similar way (to ensure an empty grounded extension). Furthermore, every such argument a must be defended by each preferred extension. Thus, for each preferred extension E, some arguments are sampled to defend a.
4) For every preferred extension E and $a \in E \setminus S_{pa}$, attackers for a are sampled from $\mathsf{Arg} \setminus E$ and corresponding defenders are defined within E.
5) Additional random attacks are added between arguments in S_{unacc}.
6) In order to avoid having stable extensions (which may also ease computation of arguments that are skeptically accepted under preferred semantics, since every stable extension is also preferred), we add another self-attacking argument and some attacks between this argument and arguments from S_{unacc}.

Note that due to the random approach of generating an argumentation graph, it may not necessarily be the case that the number of skeptically/credulously accepted arguments (w.r.t. preferred semantics) as well as the number of arguments in the ideal extension exactly match the given parameters. However, our experiments in [4] showed that it is indeed relatively hard to decide skeptical acceptance (w.r.t. preferred semantics) for most arguments in the resulting graph.

The graph generator[3] and an example demonstrating its usage[4] we used can be found online.

[3]http://tweetyproject.org/r/?r=kwt_gen
[4]http://tweetyproject.org/r/?r=kwt_gen_ex

IV. Conclusion

We described the *KWT Benchmark Generator*, which aims at producing challenging argumentation frameworks to evaluate the task of deciding skeptical acceptance under preferred semantics by largely avoiding the production of instances that can be solved by reverting to a less complex problem.

References

[1] I. Kuhlmann and M. Thimm, "A discussion of challenges in benchmark generation for abstract argumentation." in *Arg&App@ KR*, 2023, pp. 78–84.
[2] P. M. Dung, "On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games," *Artificial Intelligence*, vol. 77, no. 2, pp. 321–358, 1995.
[3] P. E. Dunne and T. J. M. Bench-Capon, "Coherence in finite argument systems," *Artificial Intelligence*, vol. 141, no. 1/2, pp. 187–203, 2002.
[4] I. Kuhlmann, T. Wujek, and M. Thimm, "On the impact of data selection when applying machine learning in abstract argumentation," in *Computational Models of Argument*. IOS Press, 2022, pp. 224–235.
[5] P. M. Dung, P. Mancarella, and F. Toni, "Computing ideal sceptical argumentation," *Artificial Intelligence*, vol. 171, no. 10-15, pp. 642–674, 2007.