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