Heureka: A General Heuristic Backtracking Solver for Abstract Argumentation

Nils Geilen and Matthias Thimm(✉)

Institute for Web Science and Technologies,
Universität Koblenz-Landau, Koblenz, Germany
thimm@uni-koblenz.de

Abstract. The HEUREKA solver is a general-purpose solver for various problems in abstract argumentation frameworks pertaining to complete, grounded, preferred and stable semantics. It is based on a backtracking approach and makes use of various heuristics to optimize the search.

ευρηκα! ευρηκα! – *I have found it! I have found it!*
– Archimedes of Syracuse (287–212 BC)

1 Introduction

An abstract argumentation framework (AAF) as defined by Dung [3] is a tuple $\Gamma = (\mathcal{A}, \mathcal{R})$ where $\mathcal{A}$ is a set of arguments and $\mathcal{R} \subseteq \mathcal{A}^2$ an attack relation between arguments. An attack $a \rightarrow b \in \mathcal{R}$ models that argument a defeats argument b. For any argument set $E \subseteq \mathcal{A}$, let E^+ be the set of arguments which are attacked by an element of E and let E^- be the set of arguments which attack an element of E. An AAF Γ is interpreted through the use of *extensions*, i.e., sets of arguments that provide a coherent view on the argumentation represented by Γ. An extension $E \subseteq \mathcal{A}$ is *conflict-free* iff there are no $a, b \in E$ with $a \rightarrow b$. An extension E is *stable* iff it is conflict-free and for every $b \in \mathcal{A} \setminus E$ there is $a \in E$ with $a \rightarrow b$. Other notions of extensions include complete, grounded, and preferred extensions, see [3] for the formal definitions.

HEUREKA is a software system that implements a direct backtracking approach for solving reasoning problems with respect to stable, complete, grounded, and preferred semantics. The backtracking approach makes use of a variety of heuristics to dynamically (re-)order the arguments to minimize the backtracking steps. HEUREKA is able to solve the problems of

- enumerating all extensions (EE),
- determining a single extension (SE),
- checking whether an argument is part of at least one extension, i.e., whether it is credulously justifiable (DC), and

E. Black et al. (Eds.): TAFA 2017, LNAI 10757, pp. 143–149, 2018.
https://doi.org/10.1007/978-3-319-75553-3_10

– checking whether an argument is part of every extension, i.e., whether it is sceptically justifiable (DS)

with respect to the four mentioned semantics. HEUREKA is written in C++ and available under the LGPL v3.0 licence on GitHub[1].

In the remainder of this paper, we describe the architecture of HEUREKA as it has been submitted to the *Second International Competition on Computational Models of Argumentation (ICCMA'17)*[2]. Note that a slightly shorter version of this paper has also been submitted as a system description to the competition.

2 Backtracking Algorithm

HEUREKA consists of a family of backtracking algorithms, one for each complete, preferred, and stable semantics which are similar to the algorithm defined in [5] but use dynamic heuristics to (re-)order how arguments are processed. The concrete algorithms differ only slightly so we focus our presentation here on the stable semantics and, in particular, on the task of computing all stable extensions.

At any time during the execution, a labelling function $\mathcal{L}ab$, which assigns to each argument either the value IN if it should be contained in the extension, OUT if it should be ruled out, or UNDEC if it is undecided, is maintained by the algorithm that keeps track of the current (partial) extension. A fourth label (BLANK) is used to indicated that an argument is not labelled yet. Let further $\text{IN}(\mathcal{L}ab)$ be the set of all arguments labelled IN by $\mathcal{L}ab$, and therefore the current solution. In a first step, the grounded extension E_{GR} is computed using a purely iterative algorithm which does not require backtracking [4] and an *intial labelling* is constructed. For an AAF $\mathit{\Gamma} = (\mathcal{A}, \mathcal{R})$ with the grounded extension E_{GR} let the *initial labelling* $\mathcal{L}ab_{\text{init}} : \mathcal{A} \to \{\text{IN}, \text{OUT}, \text{UNDEC}, \text{BLANK}\}$ be defined as

$$\mathcal{L}ab_{\text{init}}(a) = \begin{cases} \text{IN} & \text{if } a \in E_{\text{GR}} \\ \text{OUT} & \text{if } a \in E^{+}_{\text{GR}} \\ \text{UNDEC} & \text{if } a \to a \\ \text{BLANK} & \text{otherwise} \end{cases}$$

Using a specific heuristic (see next section) a new argument a is selected and set to IN in $\mathcal{L}ab$. Setting this argument to IN may require that other arguments have to be rejected (because they are attacked by a) or need to be set to IN as well (because all attackers of them are now attacked by some IN-labelled argument), and so on, see [5] for the corresponding lookahead strategies. Those arguments are then marked correspondingly in $\mathcal{L}ab$. This step is repeated until either a stable extension has been determined or a contradiction occurs (an argument is labelled with two different labels). In the latter case, the algorithm backtracks and rejects an argument previously accepted. Algorithm 1 shows a

[1] https://github.com/nilsgeilen/heureka.
[2] http://www.dbai.tuwien.ac.at/iccma17.

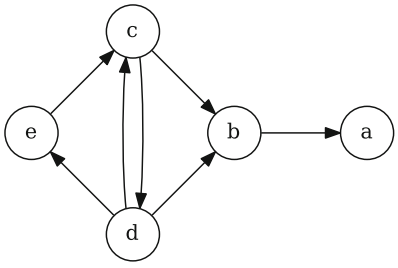
Algorithm 1. Enumerate All Stable Extensions

Input: $\Gamma = (\mathcal{A}, \mathcal{R})$ AAF
h heuristic
$\mathcal{L}ab_{\text{init}}$ initial labelling
Output: $\mathcal{E}_{\text{ST}} \subseteq 2^{\mathcal{A}}$ stable extensions

```
ENUMERATE_EXTENSIONS(Lab_init)

function SET_IN(Lab, a)
    Lab(a) ← IN
    for all b ∈ {a}^- do
        if not SET_UNDEC(Lab, b) then
            return false
    for all b ∈ {a}^+ do
        Lab(b) ← OUT
    for all c ∈ ({a}^+)^+ do
        if {c}^- ⊆ IN(Lab)^+ then
            if Lab(c) = UNDEC then
                return false
            else if not SET_IN(Lab, c) then
                return false
    if IS_STABLE(Lab) then
        add IN(Lab) to E_ST
        return false
    else return true

function SET_UNDEC(Lab, a)
    Lab(a) ← UNDEC
    if |{a}^- \ IN(Lab)^+| = 1 then
        find b ∈ {a}^- \ IN(Lab)^+
        if not SET_IN(Lab, b) then
            return false
    return true

procedure ENUMERATE_EXTENSIONS(Lab)
    let h choose next argument a, if there is none, stop
    if Lab(a) = BLANK then
        Lab' ← Lab
        if SET_IN(Lab', a) then
            ENUMERATE_EXTENSIONS(Lab')
        if SET_UNDEC(Lab, a) then
            ENUMERATE_EXTENSIONS(Lab)
    else ENUMERATE_EXTENSIONS(Lab)
```

shortened version of this procedure. The functions SET_IN and SET_UNDEC set the labelling of the current argument to IN or undec, respectively, and propagate the changes following the mentioned lookahead strategies. For example all arguments attacked ba an argument labelled IN are set to OUT. At the end of SET_IN, the algorithm checks whether the current extension, i.e., the set of

IN-labelled arguments in $\mathcal{L}ab$, is stable, then it is reported as a stable extension and the algorithm backtracks as the current branch cannot contain any more extensions.



step	labelling		
	IN	OUT	UNDEC
1.	$\{a\}$	$\emptyset$	$\emptyset$
2.	$\{a\}$	$\emptyset$	$\{b\}$
3.	$\{a, c\}$	$\emptyset$	$\{b\}$
4.	$\{a, c\}$	$\{b, d\}$	$\{e\}$
5.	$\{a\}$	$\emptyset$	$\{b, c\}$
6.	$\{a, d\}$	$\emptyset$	$\{b, c\}$
7.	$\{a, d\}$	$\{b, c, e\}$	$\emptyset$

Fig. 1. AAF Γ (left) and algorithm steps (right) from Example 1; arguments not present in any set are BLANK

Example 1. Consider the AAF $\Gamma = (\mathcal{A}, \mathcal{R})$ depicted in Fig. 1 (left). Assume our heuristic function determines the following order of arguments: (a, b, c, d, e). In the first step, we determine that the grounded extension is empty and that there is no self-attacking argument, so we start with an empty labelling (all arguments are blank).

1. *decision:* a is picked by the heuristic and set to IN
2. as a consequence of step 1, all attackers/attackees of a are set to UNDEC/OUT respectively, $\{a\}$ is not stable
3. *decision:* c is picked by the heuristic and set to IN
4. as a consequence of step 3, all attackers/attackees of c are set to UNDEC/OUT respectively, $\{a, c\}$ is not stable
5. there are no more arguments which are still undecided, so the algorithm backtracks to the last decision in step 3 and sets c to OUT
6. *decision:* d is picked by the heuristic and set IN
7. as a consequence of step 6, all attackers/attackees of d are set to UNDEC/OUT respectively, $\{a, d\}$ is stable $\Rightarrow$ stop

The backtracking algorithms for preferred and complete semantics are similar to the one for stable semantics. Reasoning problems pertaining to credulous/sceptical justification are solved by the same algorithms but with different termination criteria and slightly different initial steps.

3 Heuristics

While it is clear that the backtracking approach outlined before is a sound and complete procedure to enumerate extensions, its performance is highly dependent

on the order in which arguments are processed. Observe that if this order is perfect, i.e., all arguments within the final extension are processed first, then no backtracking is needed and the algorithm has polynomial runtime. However, this runtime performance cannot, of course, be guaranteed but the choice of the heuristic used in ordering the arguments can deeply influence the runtime in general. HEUREKA comes with a series of different heuristics for this purpose.

In general, a heuristic h is a function $h : 2^{\mathcal{A}} \times \mathcal{A} \to \mathbb{R}$ that maps the current partial extension $E \subseteq \mathcal{A}$, i.e., the set of IN-labelled arguments in $\mathcal{L}ab$, and an argument $a \in \mathcal{A}$ to a real number $h(E, a)$. A large value $h(E, a)$ indicates that a should be likely included in the extension E and should be processed earlier than arguments with lower score. Some of our heuristics are defined independently of E and therefore need not to be recomputed after every modification of E. In general, however, HEUREKA allows for dynamic heuristics that are updated after every step.

A simple example of such a heuristic is the number of undefeated aggressors, i.e., the number of arguments which attack a but are not defeated by E. The number of undefeated aggressors $h_{\mathrm{UA}}(E, a)$ should be used as a negatively weighted component in a compound heuristic as every aggressor increases the vulnerability of an argument.

$$h_{\mathrm{UA}}(E, a) = \left|\{a\}^- \backslash E^+\right|$$

Another example which is independent of E is the ratio of an argument's in-degree and out-degree:

$$h^{\div}_{\deg}(E, a) = \frac{|\{a\}^+| + \epsilon}{|\{a\}^-| + \epsilon} \text{with } \epsilon \in \mathbb{R}$$

Path-based heuristics have proven useful in many cases. Let $d_i^+(a)$ be the number of paths of length i originating in a and let $d_i^-(a)$ be the number of paths of length i ending in a. The path-based components h^+_{path} and h^-_{path} map an argument to a combination of its outgoing paths or ingoing paths respectively.

$$h^+_{\mathrm{path}}(E, a) = \sum_{i=1}^{k} \alpha^i d_i^+(a)$$

$$h^-_{\mathrm{path}}(E, a) = \sum_{i=1}^{k} \beta^i d_i^-(a)$$

These heuristics can be combined into more complex path-based heuristics like $h^{\Sigma}_{\mathrm{path}} = h^-_{\mathrm{path}} + h^+_{\mathrm{path}}$ or $h^{\Pi}_{\mathrm{path}} = (-1) \cdot (h^-_{\mathrm{path}} + \epsilon) \cdot (h^+_{\mathrm{path}} + \epsilon)$.

Further heuristics have been implemented on top of well-known graph metrics such as betweenness centrality, eigenvector centrality, and matrix exponential. Another approach are SCC-based heuristics, which order arguments according to the ordering number of the strongly connected component, which they are part of, thus implementing ideas on SCC-recursiveness [1]. On top of the individual heuristics, HEUREKA also allows heuristics to be combined arithmetically.

For ICCMA'17, we fixed a heuristic for every problem based on a small experimental evaluation. For all tasks except SE-ST (enumerating some stable extension) we used the heuristic h_1, i.e., h^+_{path} with fixed parameters $\alpha = 0.5$ and $k = 3$, defined as

$$h_1(E, a) = \sum_{i=1}^{3} \frac{d_i^+(a)}{2^i}$$

This heuristic shows the power of an argument to defend and defeat arguments. For the task SE-ST we used the heuristic h_2, which combines h^+_{path} with h^-_{path} and h_{UA}.

$$h_2(E, a) = h_1(E, a) + \sum_{i=1}^{3} \frac{d_i^-(a)}{(-2)^i} - \frac{|\{a\}^- \backslash E^+|}{2}$$

This heuristic is influenced by the *matrix exponential* which has been suggested for this use in [2].

Later a systematic evaluation of the implemented heuristics has been conducted. During this evaluation $h^{\div}_{\text{deg}}$ has proven most useful for solving problems under stable semantics while h^{Π}_{path} worked best when solving problems under complete or preferred semantics. For some graphs the performance could be substantially increased by adding an SCC-based component to the heuristic. In future work, heuristics could be explored, which also discriminate between OUT and UNDEC arguments intead of only analysing the partial extension.

4 Summary

We presented HEUREKA, a general-purpose argumentation solver based on the backtracking paradigm. The solver is backed by a number of heuristics that (dynamically) order the arguments of an abstract argumentation framework to minimize the number of necessary backtracking steps. During ICCMA'17, all results returned by HEUREKA have been correct. It landed in the center field for most tasks, while it was the fastest to find the grounded extension. Current and future work comprises analytical and empirical evaluation of the solver and its heuristics, as well as the development of new heuristics and combinations thereof.

References

1. Baroni, P., Giacomin, M., Guida, G.: SCC-recursiveness: a general schema for argumentation semantics. Artif. Intell. **168**(1–2), 162–210 (2005)
2. Corea, C., Thimm, M.: Using matrix exponentials for abstract argumentation. In: Proceedings of the First Workshop on Systems and Applications of Formal Argumentation (SAFA 2016), pp. 10–21, September 2016

3. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–357 (1995)
4. Nofal, S., Atkinson, K., Dunne, P.E.: Algorithms for argumentation semantics: labeling attacks as a generalization of labeling arguments. J. Artif. Intell. Res. **49**, 635–668 (2014)
5. Nofal, S., Atkinson, K., Dunne, P.E.: Looking-ahead in backtracking algorithms for abstract argumentation. Int. J. Approx. Reason. **78**, 265–282 (2016)