Heureka: A General Heuristic Backtracking Solver for Abstract Argumentation

Nils Geilen and Matthias Thimm

Institute for Web Science and Technologies,
Universität Koblenz-Landau, Germany

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 $AF = (\mathcal{A}, \mathcal{R})$ where $\mathcal{A}$ is a set of arguments and $\mathcal{R} \subseteq \mathcal{A}^2$ an attack relation between arguments. An attack $a \to b \in \mathcal{R}$ models that argument a defeats argument b. An AAF AF is interpreted through the use of *extensions*, i. e., sets of arguments that provide a coherent view on the argumentation represented by AF. An extension $E \subseteq \mathcal{A}$ is *conflict-free* iff there are no $a, b \in E$ with $a \to b$. 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 $a \to b$. 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 wrt. stable, complete, grounded, and preferred semantics. The backtracking approach makes uses of a variety of heuristics to dynamically (re-)order the arguments in order to minimize the backtracking steps. HEUREKA is able to solve the problems of 1) enumerating all extensions (EE), 2) determining a single extension (SE), 3) checking whether an argument is part of at least one extension, i. e., whether it is creduously justifiable (DC), and 4) 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].

---

[1] `https://github.com/nilsgeilen/heureka`

[2] `http://www.dbai.tuwien.ac.at/iccma17`

Algorithm 1 Enumerate All Stable Extensions

```
Input:  $AF = (\mathcal{A}, \mathcal{R})$ AAF
        $h$ heuristic
        $E_{\mathrm{GR}}$ the grounded extension
Output: $\mathcal{E}_{\mathrm{ST}} \subseteq 2^{\mathcal{A}}$ stable extensions

for all $a \in \mathcal{A}$ do
    $\mathcal{L}ab(a) \leftarrow$ UNDEC
for all $a \in E_{\mathrm{GR}}$ do
    set_in($\mathcal{L}ab, a$)
for all $a \in \mathcal{A}$ do
    if $a \to a$ then
        set_out($\mathcal{L}ab, a$)
enumerate_extensions($\mathcal{L}ab$)
procedure enumerate_extensions($\mathcal{L}ab$)
    let $h$ choose next argument $a$, if there is none, stop
    if $\mathcal{L}ab(a) =$ UNDEC then
        $\mathcal{L}ab' \leftarrow \mathcal{L}ab$
        if set_in($\mathcal{L}ab', a$) then
            enumerate_extensions($\mathcal{L}ab'$)
        if set_out($\mathcal{L}ab, a$) then
            enumerate_extensions($\mathcal{L}ab$)
    else
        enumerate_extensions($\mathcal{L}ab$)
```

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 $\mathcal{L}ab$, 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. In a first step, the grounded extension E_{GR} is computed using a purely iterative algorithm which does not require backtracking [4] and its arguments are set to IN in $\mathcal{L}ab$. 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 shortened version of this procedure. The functions SET_IN and SET_OUT set the labelling of the current argument to IN or OUT, respectively, and propagate the changes following the mentioned lookahead strategies. 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.

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 criterions 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 an argument $a \in \mathcal{A}$ and the current partial extension $E \subseteq \mathcal{A}$, i. e., the set of IN-labelled arguments in $\mathcal{L}ab$, 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:

$$h_{\mathrm{UA}}(E, a) = |\{b \mid (b, a) \in \mathcal{R}\} \backslash \{b \mid (c, b) \in \mathcal{R}, c \in E\}|$$

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

$$h_{\mathrm{DR1}}(E, a) = \frac{|\{b \mid (a, b) \in \mathcal{R}\}| + 1}{|\{b \mid (b, a) \in \mathcal{R}\}| + 1}$$

Further heuristics have been implemented on top of well-known graph metrics such as betweenness centrality, eigenvector centrality, path lengths, 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 arbitrary combinations by weighing and summation.

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 defined as

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

where $d_i^+(a)$ is the number of paths of length i originating in a. For the task SE-ST we used the heuristic h_2 defined as

$$h_2(E,a) = h_1(E,a) + \sum_{i=1}^{3} \frac{d_i^-(a)}{(-2)^i} - \frac{|\{b \mid (b,a) \in \mathcal{R}\} \setminus \{b \mid (c,b) \in \mathcal{R}, c \in E\}|}{2}$$

where $d_i^-(a)$ is the number of paths of length i ending in a. This heuristic is influenced by the *matrix exponential* which has been suggested for this use in [2].

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. 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. Artificial Intelligence 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'16). 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. Artificial Intelligence 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. Journal of Artificial Intelligence Research 49, 635–668 (2014)
5. Nofal, S., Atkinson, K., Dunne, P.E.: Looking-ahead in backtracking algorithms for abstract argumentation. International Journal of Approximate Reasoning 78, 265–282 (2016)