

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.