ConArg: A Constraint-programming Solver for Abstract Argumentation Problems

Stefano Bistarelli[1], Fabio Rossi[1], Francesco Santini[1]

Dipartimento di Matematica e Informatica, Università di Perugia, Italy
[bista,rossi,francesco.santini]@dmi.unipg.it

Abstract. ConArg is a Constraint-programming solver oriented to the solution of problems related to extension-based semantics in Abstract Argumentation. It exploits Gecode, an efficient C++ toolkit for developing constraint-based systems and applications. The properties required by semantics are encoded into constraints, and arguments are assigned to *true* if belonging to a valid extension for that semantics. Searching for solutions of problems (as enumerating extensions or checking argument-acceptance) takes advantage of well-known techniques as local consistency, different heuristics for trying to assign values to variables, and complete search-tree with branch-and-bound.

Description

ConArg (*Arg*umentation with *Con*straints) is a Constraint-programming tool oriented to the solution of problems related to extension-based semantics in Abstract Argumentation [12]. Since the first version of the tool [1,10], it has been updated with the purpose *i)* to solve further problems linked to weighted problems [7,8] and coalitions of arguments [11], and *ii)* to improve its performance over classical semantics, by using a benchmark assembled with random graph-models [2,3,4]. The first version of ConArg [10] was based on the *Java Constraint Programming* solver[1] (*JaCoP*), a Java library that provides a *Finite Domain Constraint Programming* paradigm [14].

Successively, for the sake of performance, a second version of the tool was developed: the current version of ConArg exploits *Gecode 4.4.0*[2], an efficient C++ toolkit for developing constraint-based systems and applications. Some performance comparisons have been already provided in [5,6]. The properties of semantics are encoded into constraints, and arguments are assigned to 1 (*true*) if belonging to a valid extension for that semantics (0 otherwise). Searching for solutions takes advantage of classical techniques, such as local consistency, different heuristics for trying to assign values to variables, and complete search-tree with branch-and-bound. We have also dropped the graphical interface of the first Java implementation, having a textual output only.

[1] http://www.jacop.eu
[2] http://www.gecode.org.

ConArg was submitted to the *International Competition on Computational Models of Argumentation* (ICCMA 2015) [15]. In 2015 the tool did not compete in all the proposed tracks; for instance, it did not participate to credulous and sceptical acceptance of arguments with the grounded semantics. One of the aims for the 2017 edition of ICCMA is to participate in all the proposed tracks, in order to have a thorough comparison.

ConArg can be currently used to:

– enumerate all conflict-free, admissible, complete, stable, grounded, preferred, semi-stable, ideal, and stage extensions;
– return one extension given one of the semantics above;
– check the credulous and sceptical acceptance for the conflict-free, admissible, complete, and stable semantics;
– find the α^γ-semantics described in [7,8].

From the home-page of ConArg[3], it is possible to download the executable of the solver, compiled for Linux i386 and x64 machines. Moreover, still at the same Web-site, we offer a visual interface where to interactively draw abstract frameworks (arguments and attacks as directed edges), and use ConArg as the underlying solver for the requested problem.

The basic command-line usage is described in Fig. 1. Some practical examples are: to enumerate all admissible extensions: *"conarg_gecode -e admissible file.dl"*, to check the sceptical acceptance of argument "a" with the stable semantics *"conarg_gecode -e stable -s a file.dl"*, to compute all α-complete extensions [9] with $\alpha = 3$ *"conarg_gecode -e a-complete -a 3 file.dl"*. An input file *file.dl* follows the ASPARTIX format [13]: e.g., *arg(a)* for defining argument a, and $att(a, b)$ for declaring an attack from a to b.

We briefly show how we map AAFs to *Constraint Satisfaction Problems* (CSPs) [14] in ConArg. A CSP can be defined as a triple $P = \langle V, D, C\rangle$, where C is a set of constraints defined over the variables in V, each with domain D. Given a framework $\langle A, R\rangle$, we define a variable for each argument $a_i \in A$ ($V = \{a_1, a_2, \dots, a_n\}$) and each of these arguments can be taken or not in an extension, i.e., the domain of each variable is $D = \{1, 0\}$. As an example we report conflict-free and stable constraints, which can be respectively used to model the conflict-free and (in combination) stable semantics.

– Conflict-free constraints. If $R(a_i, a_j)$ is in the framework we need to prevent a solution to include both a_i and a_j: $\neg(a_i = 1 \wedge a_j = 1)$. All other possible variable assignments $(a = 0 \wedge b = 1)$, $(a = 1 \wedge b = 0)$ and $(a = 0 \wedge b = 0)$ are permitted.
– Stable constraints. If we have a node a_i with multiple parents (in the Argumentation graph) $a_{f1}, a_{f2}, \dots, a_{fk}$, we need to add a constraint $\neg(a_i = 0 \wedge a_{f1} = 0 \wedge \dots \wedge a_{fk} = 0)$. In words, if a node is not taken in an extension (i.e. $a_i = 0$), then it must be attacked by at least one of the taken nodes, that is at least a parent of a_i needs to be taken in a solution (that is, $a_{fj} = 1$).

[3] `http://www.dmi.unipg.it/conarg/`.

```
USAGE:
conarg_gecode  [-s <string> ] [-c <string> ] [-a <double> ] -e <string>
[--] [--version] [-h] <string>

Where:
-s <string> ,  --skeptical <string>
Test an argument for sceptical acceptance (conflict-free, admissible,
     complete and stable semantics only).
-c <string> ,  --credulous <string>
Test an argument for credulous acceptance (conflict-free, admissible,
     complete and stable semantics only).
-a <double> ,  --alpha <double>
Alpha consistency budget (for alpha extensions only).
-e <string> ,  --extension <string>
(required)  Extensions to be enumerated (conflict-free,  admissible,
     complete, stable, preferred, grounded, semi-stable, ideal, stage, a-
     conflict-free,  a-admissible, a-complete, a-stable, a-preferred, a-
     grounded semantics).
--,  --ignore_rest
Ignores the rest of the labeled arguments following this flag.
--version
Displays version information and exits.
-h,  --help
Displays usage information and exits.
<string>
(required)  Input File  in Aspartix format.
```

Fig. 1. How to call ConArg from command-line.

Moreover, if a node a_i has no parent in the graph, it has to be included in every extension, i.e., $\neg(a_i = 0)$.

Preferred extensions are found by assigning as more arguments as possible to 1 while searching for complete extensions. For this we use the Gecode heuristics *INT_VAL_MAX* (such value is always 1 in our model).

Given a semantics, the credulous acceptance for an argument a is checked by setting that argument to 1 and then halting as soon as an extension containing a is found (i.e., a is credulously accepted). In the worst case, all the search tree is explored without any result, i.e., a is *not* credulously accepted. Checking the sceptical acceptance is a dual problem: given a semantics, we set a to 0 and then we stop as soon as an extension containing a is found (i.e., a is *not* credulously accepted). In the worst case, all the search tree is explored without any result, i.e., a is sceptically accepted.

In the future we would like to extend ConArg to solve coalition-based problems [11], and labelling-based extensions, where having an assignment domain wider than just $\{true, false\}$ suggests the use of a constraint-based solver. Further possible extensions concern Bipolar Argumentation Frameworks, or Constrained-Argumentation Frameworks, where additional used-defined constraints can be adopted to select only some extensions of a given semantics (e.g., "when a is in, then also b must be in"). In addition, we are currently exploring applications of our tool as a reasoning engine for Cybersecurity problems and Decision-making.

References

1. S. Bistarelli, D. Pirolandi, and F. Santini. Solving weighted argumentation frameworks with soft constraints. In *ERCIM International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP)*, volume 6384 of *LNCS*, pages 1–17, 2009.
2. S. Bistarelli, F. Rossi, and F. Santini. Benchmarking hard problems in random abstract AFs: The stable semantics. In *Computational Models of Argument - Proceedings of COMMA*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 153–160. IOS Press, 2014.
3. S. Bistarelli, F. Rossi, and F. Santini. Efficient solution for credulous/sceptical acceptance in lower-order dung's semantics. In *26th IEEE International Conference on Tools with Artificial Intelligence, (ICTAI)*, pages 800–804. IEEE Computer Society, 2014.
4. S. Bistarelli, F. Rossi, and F. Santini. Enumerating extensions on random abstract-AFs with ArgTools, Aspartix, ConArg2, and Dung-O-Matic. In *Computational Logic in Multi-Agent Systems - 15th International Workshop, CLIMA XV*, volume 8624 of *LNCS*, pages 70–86. Springer, 2014.
5. S. Bistarelli, F. Rossi, and F. Santini. A first comparison of abstract argumentation reasoning-tools. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, volume 263 of *FAIA*, pages 969–970. IOS Press, 2014.
6. S. Bistarelli, F. Rossi, and F. Santini. A comparative test on the enumeration of extensions in abstract argumentation. *Fundam. Inform.*, 140(3-4):263–278, 2015.
7. S. Bistarelli, F. Rossi, and F. Santini. A collective defence against grouped attacks for weighted abstract argumentation frameworks. In *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference, FLAIRS*, pages 638–643. AAAI Press, 2016.
8. S. Bistarelli, F. Rossi, and F. Santini. A relaxation of internal conflict and defence in weighted argumentation frameworks. In *Logics in Artificial Intelligence - 15th European Conference, JELIA*, volume 10021 of *LNCS*, pages 127–143. Springer, 2016.
9. S. Bistarelli and F. Santini. A common computational framework for semiring-based argumentation systems. In *ECAI 2010 - 19th European Conference on Artificial Intelligence*, volume 215 of *FAIA*, pages 131–136. IOS Press, 2010.
10. S. Bistarelli and F. Santini. Conarg: A constraint-based computational framework for argumentation systems. In *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 605–612. IEEE Computer Society, 2011.
11. S. Bistarelli and F. Santini. Coalitions of arguments: An approach with constraint programming. *Fundam. Inform.*, 124(4):383–401, 2013.
12. P. M. Dung. 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.
13. U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.
14. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.
15. M. Thimm, S. Villata, F. Cerutti, N. Oren, H. Strass, and M. Vallati. Summary report of the first international competition on computational models of argumentation. *AI Magazine*, 37(1):102, 2016.