argmat-clpb: Solve Argumentation Problems using Constraint Logic Programming over Boolean variables[⋆]

Fuan Pu, Guiming Luo, and Yucheng Chen

School of Software, Tsinghua University, Beijing, China
`Pu.Fuan@gmail.com, gluo@tsinghua.edu.cn, chenyc14@163.com`

Abstract. This paper presents a system description of `argmat-clpb`, which chooses Constraint Logic Programming over Boolean variables as its logic engine to encode and solve argumentation problems based on Boolean matrix algebra. We describe the main theory behind `argmat-clpb`, the main software architecture of `argmat-clpb`, and a simple usage for `argmat-clpb`.

1 Boolean matrix algebra representation

The basic elements of abstract argumentation framework (AF) [1] can be represented by Boolean matrices (see [2]). Let $\Delta = \langle \mathcal{X}, \mathcal{R} \rangle$ be a finite AF with $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$, and let $\mathcal{B} = \{0, 1\}$ be the Boolean domain.

- Assume $S \subseteq \mathcal{X}$, then we can represent S by an $n \times 1$ Boolean vector $\mathbf{s} \in \mathcal{B}^{n\times 1}$, whose row indices index the elements in $\mathcal{X}$, such that the i^{th} component of $\mathbf{s}$, denoted by $\mathbf{s}_i$, is defined by $\mathbf{s}_i \overset{\texttt{def}}{=} 1$ if $x_i \in S$; otherwise $\mathbf{s}_i \overset{\texttt{def}}{=} 0$.
- The attack relation $\mathcal{R}$ on $\mathcal{X}$ for Δ can be represented by an $n \times n$ Boolean matrix $\mathbf{A} \in \mathcal{B}^{n\times n}$, whose row and column indices index the elements of $\mathcal{X}$, respectively, such that the entry $\mathbf{A}_{ij}$ is defined by $\mathbf{A}_{ij} \overset{\texttt{def}}{=} 1$ if $x_j \mathcal{R} x_i$; otherwise $\mathbf{A}_{ij} \overset{\texttt{def}}{=} 0$. We call $\mathbf{A}$ the **attack matrix** of Δ. It can be seen that $\mathbf{A}$ is the transpose of the adjacency matrix of the argument graph of Δ.

Obviously, each subset of $\mathcal{X}$ is one-to-one correspondence to a Boolean vector in $\mathcal{B}^{n\times 1}$. In this paper, thus, we may mix a subset and a Boolean vector whenever it is convenient.

The Boolean operations on Boolean variables, such as "$+$" (*logical or*), "$*$" (*logical and*), "$\neg$" (*logical not*), "$\leqslant$" (*logical implication*), "$\oplus$" (*logical disequality*) and "$\equiv$" (*logical equivalence*), can be applied on Boolean vectors element-wisely. The multiplication of $\mathbf{A}$ and $\mathbf{s}$, denoted by $\mathbf{A} \odot \mathbf{s}$, is a Boolean vector $\mathbf{t} \in \mathcal{B}^{n\times 1}$ defined by

$$\mathbf{t}_i = \sum_{j=1}^{n} \mathbf{A}_{ij} * \mathbf{s}_j \tag{1}$$

Then, some important functions of Δ can be also represented by Boolean matrices and Boolean operations, see Table 1 (for more details, please refer to [2]).

[⋆] This work was supported by the Fund NSFC61572279.

Table 1. Boolean matrix representation for some important functions

Functions	Meaningness	Boolean matrix representation
$\mathcal{R}^+(S)$	returns all arguments that are attacked by S	$\mathcal{R}^+(\mathbf{s}) \overset{\text{def}}{=} \mathbf{A} \odot \mathbf{s}$
$\mathcal{R}^-(S)$	returns all arguments that attack S	$\mathcal{R}^-(\mathbf{s}) \overset{\text{def}}{=} \mathbf{A}^{\mathrm{T}} \odot \mathbf{s}$
$\mathcal{N}(S)$	returns all arguments that not attacked by S	$\mathcal{N}(\mathbf{s}) \overset{\text{def}}{=} \neg\mathcal{R}^+(\mathbf{s}) = \neg(\mathbf{A} \odot \mathbf{s})$
$\mathcal{I}(S)$	returns all arguments that do not attack S	$\mathcal{I}(\mathbf{s}) \overset{\text{def}}{=} \neg\mathcal{R}^-(\mathbf{s}) = \neg(\mathbf{A}^{\mathrm{T}} \odot \mathbf{s})$
$\mathcal{F}(S)$	returns all arguments that are defended by S	$\mathcal{F}(\mathbf{s}) \overset{\text{def}}{=} \mathcal{N}(\mathcal{N}(\mathbf{s})) = \neg\big(\mathbf{A} \odot \neg(\mathbf{A} \odot \mathbf{s})\big)$

2 Encoding Dung's Acceptability Semantics

Now, we encode Dung's acceptability semantics via Boolean matrix algebra. Each semantics is encoded as a finite array of Boolean expressions (constraints) with a vector of Boolean variables. The goal is to find an assignment to the vector of all Boolean variables so that all constraints evaluate to 1, i.e., be *satisfied*. This is a typical Boolean constraint satisfaction problem. If no such satisfying assignment exists, then these Boolean constraints have no solution.

Conflict-free Boolean constraints. Boolean vector $\mathbf{s}$ is conflict-free iff any of the equivalent Boolean constraints below is satisfied:

$$\mathbf{s} * \mathcal{R}^+(\mathbf{s}) \equiv \mathbf{0} \qquad [\texttt{CF1}] \qquad\qquad \mathbf{s} \leqslant \mathcal{I}(\mathbf{s}) \qquad [\texttt{CF3}]$$

$$\mathbf{s} \leqslant \mathcal{N}(\mathbf{s}) \qquad [\texttt{CF2}]$$

Stable Boolean constraints. Boolean vector $\mathbf{s}$ is a stable extension iff any of the equivalent Boolean constraints below is satisfied:

$$\mathbf{s} \equiv \mathcal{N}(\mathbf{s}) \qquad [\texttt{ST1}] \qquad\qquad \mathbf{s} \oplus \mathcal{R}^+(\mathbf{s}) \qquad [\texttt{ST2}]$$

Admissible Boolean constraints. Boolean vector $\mathbf{s}$ is admissible iff any of the equivalent Boolean constraints below is satisfied:

$$\begin{cases} [\texttt{CF?}] \\ \mathbf{s} \leqslant \mathcal{F}(\mathbf{s}) \end{cases} \qquad [\texttt{AD1}] \qquad\qquad \mathbf{s} \leqslant \mathcal{N}(\mathbf{s}) * \mathcal{F}(\mathbf{s}) \qquad [\texttt{AD3}]$$

$$\mathbf{s} \leqslant \mathcal{F}\left(\mathbf{s} * \mathcal{N}(\mathbf{s})\right) \qquad [\texttt{AD4}]$$

$$\begin{cases} [\texttt{CF?}] \\ \mathcal{R}^-(\mathbf{s}) \leqslant \mathcal{R}^+(\mathbf{s}) \end{cases} \qquad [\texttt{AD2}] \qquad\qquad \mathbf{s} \leqslant \mathcal{N}\left(\mathbf{s} + \mathcal{N}(\mathbf{s})\right) \qquad [\texttt{AD5}]$$

where $[\texttt{CF?}]$ is any conflict-free Boolean constraint.

Complete Boolean constraints. Boolean vector $\mathbf{s}$ is complete iff any of the equivalent Boolean constraints below is satisfied:

$$\begin{cases} [\mathtt{CF?}] \\ \mathbf{s} \equiv \mathcal{F}(\mathbf{s}) \end{cases} \qquad [\mathtt{CO1}]$$

$$\mathbf{s} \equiv \mathcal{N}(\mathbf{s}) * \mathcal{F}(\mathbf{s}) \qquad [\mathtt{CO2}]$$

$$\mathbf{s} \equiv \mathcal{F}\left(\mathbf{s} * \mathcal{N}(\mathbf{s})\right) \qquad [\mathtt{CO3}]$$

$$\mathbf{s} \equiv \mathcal{N}\left(\mathbf{s} + \mathcal{N}(\mathbf{s})\right) \qquad [\mathtt{CO4}]$$

$$\mathbf{s} \oplus \mathcal{R}^{+}\left(\mathbf{s} + \mathcal{N}(\mathbf{s})\right) \qquad [\mathtt{CO5}]$$

in which $[\mathtt{CF?}]$ is a conflict-free Boolean constraints.

3 System Architecture

In this work, we choose Constraint Logic Programming over Boolean variables (CLPB) systems [3], the modern Constraint Programming solvers, as our tool to solve these Boolean constraints. CLPB is a declarative formalism for reasoning about propositional formulas. It is an instance of the general CLP scheme that extends logic programming with reasoning over Boolean domains [4]. Many Prolog systems (e.g., SWI-Prolog[1], CHIP, SICStus Prolog) are equipped with CLPB systems. The solver contains predicates for checking the consistency and entailment of a constraint with respect to previous constraints, and for computing particular solutions to the set of previous constraints. We select CLPB mainly based on following critical reasons: (1) CLPB systems are algebraically oriented, and thus they are more suitable for encoding and solving our Boolean algebra problems; (2) CLPB systems provide plentiful operations on Boolean variables, and have abilities to handle any Boolean expressions with little conversion; (3) CLPB systems provide more flexible interfaces, and they can support variable quantification, conditional answers and easy symbolic manipulation of formulas.

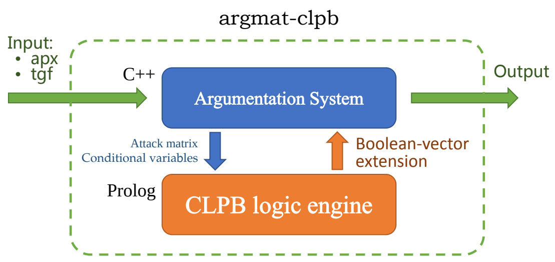

Fig. 1. `argmat-clpb` system architecture

Figure 1 shows the system architecture of `argmat-clpb`, which consists of two main parts. The argumentation system, implemented by C++, is to parse and manage the information of an AF. The argumentation system also realizes the reasoning tasks of grounded semantics by iteratively calculating the Boolean-matrix-based characteristic function $\mathcal{F}(\mathbf{s})$ from Boolean vector $\mathbf{s}^{(0)} = \mathbf{0}^n$ (i.e. the Boolean vector representation of the emptyset),

$$\mathbf{s}^{(k)} = \mathcal{F}(\mathbf{s}^{(k-1)}) = \neg\big(\mathbf{A} \odot \neg(\mathbf{A} \odot \mathbf{s}^{(k-1)})\big).$$

[1] http://www.swi-prolog.org/

Since the grounded extension is included in all complete extensions and stable extensions (if exists), and all arguments that are attacked by the grounded extension are not in any complete and stable extensions, we thus can use the grounded extension to product some initially conditional variables for speeding the processes of solving complete and stable semantics.

The CLPB logic engine is the core logic of `argmat-clpb`, responsible for encoding, solving and reasoning argumentation problems of the four semantics mentioned in Section 2. The CLPB logic engine is implemented using Prolog language based on SWI-Prolog platform (see [5] for a system description). It receives an attack matrix of an AF, as well as some conditional variables, from the argumentation system, and returns Boolean-vector extensions to the argumentation system. Then, the argumentation system parses the Boolean-vector extension(s) and outputs the results with specified format. All previously mentioned Boolean constraint models have been encoded using the CLPB engine, and the codes have been submitted into an online Prolog interpreter.[2] You can try our codes on your browser without installing any components on your system. For more implementation details, please refer to [2]. All source codes is available on the website of our project `argumatrix`[3].

4 Usage

`argmat-clpb` currently implements all reasoning tasks of the above four semantics, as well as grounded semantics, but conflict-free and admissible semantics are not part of the ICCMA2017. `argmat-clpb` supports the standard interface of the requirements of ICCMA2017, and supports all encodings mentioned in this work. You can test our encodings by specifying the encoding number after the problem option. For instance, you can use the following command to test the encoding [CO1] for complete semantics:

```
./argmat-clpb.out -p EE-CO1 -f <file> ...
```

Of course, you can directly use the option "`-p EE-CO`", since we have specified a default encoding for the semantics. The same goes for stable semantics.

References

1. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Journal of Artificial Intelligence **77**(2) (September 1995) 321–357
2. Fuan, P., Guiming, L., JIANG, Z.: Encoding argumentation semantics by boolean algebra. IEICE Transactions on Information and Systems **100**(4) (2017) 838–848
3. Codognet, P., Diaz, D.: A simple and efficient boolean solver for constraint logic programming. Journal of Automated Reasoning **17**(1) (1996) 97–128
4. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. The journal of logic programming **19** (1994) 503–581
5. Triska, M.: The boolean constraint solver of SWI-Prolog: System description. In: FLOPS. Volume 9613 of LNCS. (2016) 45–61

[2] `http://swish.swi-prolog.org/p/argmat-clpb.pl`.

[3] `https://sites.google.com/site/argumatrix/`