THE MATRIXX SOLVER FOR ARGUMENTATION FRAMEWORKS

Maximilian Heinrich
Intelligent Systems Department
Computer Science Institute
Leipzig University
mheinrich@informatik.uni-leipzig.de

ABSTRACT

MatrixX is a solver for Abstract Argumentation Frameworks. Offensive and defensive properties of an Argumentation Framework are notated in a matrix style and the corresponding rows and columns of the matrix are reduced according to the chosen semantics. This procedure is implemented through the use of hashmaps in order to accelerate the calculation time. The solver works for stable and complete semantics.

1 Introduction

Main idea for the MatrixX solver is the observation that an Argumentation Framework (AF) can be written in a matrix style, where the offensive and defensive properties of the framework are displayed as rows and columns. The solver then shrinks the matrix in a systematic way. This procedure is inspired by Knuth's Algorithm X [1]. Therefore, matrix approach plus Algorithm X results in the chosen solver name. The solver can be found at:

`https://github.com/kmax-tech/MatrixX`.

MatrixX supports stable and complete semantics according to ICCMA specifications[1] [2]. The paper starts with a very short recap about Argumentation Frameworks, after that the operating principle of the solver is explained on a practical example.

2 Argumentation Frameworks

Argumentation Frameworks, first introduced by Dung [3], are a directed Graph $F = (A, R)$ with a set of arguments A and an attack relation $R \subseteq A \times A$. Regarding semantics, we state that a set $S \subseteq A$ is conflict-free (cf) if S does not attack any of its elements and S is admissible (adm) if S is conflict-free and S defends all its elements against their attackers. A node is defended if all of its attackers are getting attacked. In addition the set S is complete (co) if it is admissible and S contains all elements it defends. The set S is stable (st) if it is conflict-free and each argument, which is not in S, is getting attacked [4]. For further specification we say that for a node $a \in A$ the set off_a contains all nodes which a attacks, formally stated as $off_a = \{b|(a, b) \in R\}$. In the paper we will refer to off_a as the offensive properties of node a. Vice versa, with defensive properties def_a we denote the set of all nodes which are attacking a, stated as $def_a = \{b|(b, a) \in R\}$. The nodesize n of an AF is specified as $|A|$, which is the number of arguments in A. For a better illustration of the solver mechanics we take Figure 1 with $E = (\{a, b, c, d\}, \{(a, b), (b, c), (c, d), (d, a)\})$ as an example, which has the stable interpretations $\sigma_{st} = \{\{a, c\}, \{d, b\}\}$ and complete interpretations $\sigma_{co} = \{\{\}, \{a, c\}, \{d, b\}\}$.

3 Matrix Representation of AFs

A matrix representation M_A for an AF A with nodesize n is a $n \times n$ matrix, where each row and column represents a corresponding node of the AF. For better orientation we use the nodes directly as indizes for the matrix. If a node attacks another node the entry in the matrix is marked with 1 otherwise 0 is used. More formally, we start with a zero

[1] More precise: [CE-ST,SE-ST,DC-ST,DS-ST,CE-CO,SE-CO,DC-CO,DS-CO]

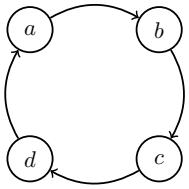

Figure 1: An Example AF E

nodes	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	0	0	0	1
d	1	0	0	0

DEF (columns) / OFF (rows)

Table 1: Matrix M_E

The AF E and its matrix representation M_E. If we look in the matrix e.g. at the row for a we see an entry at b, meaning that a attacks b and vice versa that b is attacked by a

matrix and add the entries $\forall a \in A : b \in off_a \rightarrow m_{ab} = 1$. Because offensive and defensive properties are closely related, we also obtain the defensive properties of all nodes with this construction process. If we look at the rows of the created matrix we can see the offensive properties of the corresponding node and if we look at a column, we get the defensive properties of the represented node, every existing relation is marked with 1. The solver operates on shrinking the offensive and defensive entries in M, where the rows and columns are removed independently. In order to talk about the removal procedure we introduce the sets OFF and DEF, where OFF is representing all rows and DEF the columns of the matrix. At the beginning it is the case that $A = OFF = DEF$. In addition, offensive and defensive properties of the nodes are calculating w.r.t. to the existing rows and columns. In this context, for a node a, $off_a = \{b|(a, b) \in R \wedge b \in DEF\}$ and $def_a = \{b|(b, a) \in R \wedge b \in OFF\}$. For our example Figure 1, which has nodesize 4 the corresponding matrix representation is stated in Table 1. The stable semantics can now be obtained following Algorithm 1.

Algorithm 1 MatrixX - Stable Semantics Calculator

Require: AF matrix M, with information about OFF, DEF and the current extension σ

$op\_range = OFF \cap DEF$
if $\exists i \in op\_range$, where $def_i = \emptyset$ **then** ▷ a node which is not attacked exists
    select i
    C = node_choosen(M,i) ▷ apply `Node_Chosen` function to copy of M
    return L = [C]
else
    select $n \in op\_range$, where $|def_n|$ equals min ▷ select node with least attackers
    C = node_choosen(M,n) ▷ apply `Node_Chosen` function to copy of M
    N = node_not_choosen(M,n) ▷ apply `Node_Not_Chosen` function to copy of M
    return L = [C,N]
end if

function NODE_CHOSEN(matrix M, node i)
    $\sigma = \sigma \cup \{i\}$
    $OFF = OFF \setminus (i \cup off_i \cup def_i)$ ▷ erase rows
    $DEF = DEF \setminus (i \cup off_i)$ ▷ erase columns
    return M
end function

function NODE_NOT_CHOSEN(matrix M, node i)
    $OFF = OFF \setminus \{i\}$ ▷ erase rows
    return M
end function

The algorithm is applied iteratively for each generated matrix. If it is the case that for a matrix $OFF = DEF = \emptyset$, a stable interpretation is found and the current extension σ is added to the list of interpretations. If OFF is empty but DEF is not, we abandon the current matrix. In the following we illustrate the procedure with Table 1 as example. As we start no offensive or defensive information have been erased, resulting in $OFF \cap DEF = A$. In addition the current extension is $\sigma = \emptyset$. No node is unattacked. We therefore choose the node with the least number of attackers, due to heuristic reasons. In our case we have multiple options, we decide to use node d and create two copies of M_E, further referenced as submatrices. For the first submatrix we apply the function `Node_Chosen`. Hence we add d to our current extension σ, remove $\{d, a, c\}$ from OFF and erase $\{d, a\}$ from DEF. The set OFF is representing possible

node combinations, which can be chosen from. Therefore if d is chosen to be in the extension σ we remove it from OFF. Node d attacks a and gets attacked by c. This means that a and c cannot be together with d in the same extension and are getting removed from OFF, too. DEF is representing the extensionality of the AF. In order to attain stable semantics all nodes must be either in the extension σ or get attacked. For that reason only d and a can be removed from DEF. Node is d is removed because it is in the extension and a because it is getting attacked. This means that c needs to be eliminated from DEF by another node later in order to obtain a stable interpretation. For the second submatrix we apply the function `Node_Not_Chosen`. For this submatrix we have decided that d shall not be included in the extension σ, therefore this node is not able to attack any other nodes. Consequently we just remove d from OFF. No elements from DEF are getting erased. The two obtained submatrices are shown in Table 2 and 3. We continue to apply Algorithm 1 for the generated submatrices till all stable interpretations are obtained.

nodes	~~a~~	b	c	~~d~~
~~a~~	~~0~~	~~1~~	~~0~~	~~0~~
b	~~0~~	0	1	~~0~~
~~c~~	~~0~~	~~0~~	~~0~~	~~1~~
~~d~~	~~1~~	~~0~~	~~0~~	~~0~~

Table 2: M_A after `Node_Chosen` function

nodes	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	0	0	0	1
~~d~~	~~1~~	~~0~~	~~0~~	~~0~~

Table 3: M_A after `Node_Not_Chosen` function

For complete semantics a slight variation of Algorithm 1 is used. Here the function `Node_Not_Chosen` removes i from OFF, but for all following def_j evaluations node i is still considered. The complete semantics demands that all attackers must be attacked from the elements in σ, therefore offensive properties of nodes can only be erased in case a node is selected to be in the extension. In order to obtain the complete interpretations for each generated submatrix the elements in op_range are checked. If this set contains no unattacked nodes and each node in the current extension σ is unattacked, we found a complete interpretation. If the former condition is removed one could also easily calculate the admissible interpretations, though the solver does not support this semantics.

4 Conclusion

Goal during construction of the solver was that the rows and columns in the matrix M_A are erased as fast as possible. With increasing nodesize n the size of the matrix grows quadratically, aggravating the evaluation. In order to do this more efficiently the solver instead works with hashmaps. Consequently an entry for a node only contains information about its defensive and offensive properties. This way zero entries in the corresponding Matrix M_A are omitted. This procedure was also inspired by Knuth's Algorithm X, which uses the concept of dancing links. In addition the solver copies the matrix after each step, therefore backtracking and restoring of previous configurations is not required in order to test all node combinations for an extension. For additional speed-up of the calculation time we first evaluate the grounded interpretation of the AF, which is used as initial input. Self-attacking nodes are also handled separately.

References

[1] Donald E. Knuth. Dancing links. *Millennial Perspectives in Computer Science*, pages 187–214, 2000.

[2] Jean-Marie Lagniez, Emmanuel Lonca, Jean-Guy Mailly, and Julien Rossit. Introducing the fourth international competition on computational models of argumentation. In *Proceedings of the Third International Workshop on Systems and Algorithms for Formal Argumentation co-located with the 8th International Conference on Computational Models of Argument (COMMA 2020), September 8, 2020*, pages 80–85, 2020.

[3] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[4] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Design and results of the second international competition on computational models of argumentation. *Artif. Intell.*, 279, 2020.