THE ANNALS OF "DUNĂREA DE JOS" UNIVERSITY OF GALATI
FASCICLE III, 2023, VOL. 46, NO. 1, ISSN 2344-4738, ISSN-L 1221-454X
ELECTROTECHNICS, ELECTRONICS, AUTOMATIC CONTROL, INFORMATICS
Article DOI: https://doi.org/10.35219/eeaci.2023.1.04

DYNAMIC BACKTRACKING FOR ABSTRACT ARGUMENTATION SYSTEMS

Sabina Costache

Computer Science Department, Dunarea de Jos University of Galati

Abstract: Abstract Argumentation Frameworks (AFs) are a major formalism for practical reasoning, to be used in non-monotonical reasoning of intelligent agents. The paper presents some of the latest researches in the field of efficient computation for different semantics of abstract argumentation systems. An algorithm with backtracking with look-ahead and different heuristics is taken as a basis, to be further developed and to experiment new backtracking -based optimizations in the field of AF, and an adaptation of dynamic backtracking for the same field is shaped for further experiments and improvements.

Keywords: Argumentation frameworks, dynamic backtracking, practical reasoning, Beliefs-Desires-Intentions systems, preferred extensions., non-monotonical reasoning.

1. INTRODUCTION

Abstract argumentation frameworks (AFs) (Dung, 1995) are a simple formalism, but very useful in many contemporary problems, like modelling disputes between two or more agents, (modelling inference in practical reasoning, in general, when more or one and the same agent have/has to choose between competing goals and desires) (Amgoud,2007).

Practical reasoning is an evolving domain which aims to model the complex human-like way of thinking in real world problems like law, diagnosis, task planning and „reasoning toward action " in general. The whole BDI-s architectures (Beliefs/ Desires/ Intentions) use the practice and theory that resides behind these frameworks (Amgoud, 2007). Practical reasoning for BDIs governs *what agents should do* and is mainly consisting of 2 major steps (Amgoud, 2007): deliberation (i.e. identification of goals) and choosing ways to achieve goals (which is, essentially, a decision making task that implies selecting among feasible sets of plans).

A major concern in practical reasoning is that a complete formalization for it in the BDI literature is still missing, and only informal patterns of inference for simple examples are known (Amgoud,2007).

The formal meaning of an AF is given in terms of argumentation semantics. Semantics define the sets of arguments (extensions) that can be used to defend a point of view in a dispute (Alfano, 2017). The problem with most of the argumentation semantics currently useds is that they suffer from a high computational complexity (Dunne, 2009), but the complexity evaluation done so far was mainly focused on 'static' frameworks, whereas in practice AFs are dynamic systems (Baumann, 2011).

Starting within the 2010's, the interest in research for the dynamics in AFs has constantly grown (Baumann, 2021). The idea is, firstly, to dynamically add or delete arguments and attacks (see Section 2), as every dispute in the real world does, and in the meantime, to discover or adapt algorithms to tackle this problem.

An important contribution in using dynamics to better prune the search space and speed up computation of extensions- is the incremental approach from (Alfano, 2017). The authors of (Alfano, 2017) define an influenced set of an AF and an extension under a given semantics as the set of arguments that will be influenced by adding or deleting an attack a→ b (by following the chains of attacks that start with argument b) and apply the extensions' computing algorithm only over the reduced AF (that contains only the influenced arguments). This application of the algorithm only over the influenced set is called incremental, as it avoids re-computing what remains the same when new pieces of information arrive.

The adapted dynamic backtracking that we shall present in Section 3 does intuitively the same thing but with different tools, and maybe, if it will integrate appropriate heuristics, it will do it in a more effective way.

Section 2 introduces the formal background of abstract argumentation and dynamic backtracking , and Section 3 presents a new approach: using dynamic backtracking for dynamic abstract argumentation frameworks.

The paper presents an adaptation of an existing algorithm to suit Argumentation frameworks, starting from the ideas of other works (Nofal, 2016) that have integrated forms of backtracking to approach inference in practical reasoning, which is a problem of interest in many current applications.

2. FORMAL BACKGROUND

2.1. Argumentation frameworks

Definition (Dung, 1995). An abstract argumentation framework (AF) is a pair (A, R), where A is a set of abstract arguments (from a given Universe U), and R is a binary attack relation R, subset of AxA, whose elements are called *attacks*. (Thus, an AF is a directed graph where nodes correspond to arguments and edges correspond to attacks).

The following notations hold:
$\{x\}^-$ =the subset of arguments that attack argument x
$\{x\}^+$ =the subset of arguments that are attacked by argument x

The notations can be extended to sets of arguments:
S^+ ={b| exists a in S, a attacks b}= the set of all arguments attacked by S
S^- ={b| exists a in S, b attacks a}= the set of all arguments that attack S

Definition (Dung, 1995). A subset S of A is a *defence* for a in A iff for any b in A such that b attacks a, exists c in S such that c attacks b.

Definition (Dung, 1995). x in A is *acceptable* wrt S iff S is a defence of x towards all its possible attacks.

Definition (Dung, 1995). S is *conflict-free* if for each x,y in SxS, x doesn't attack y (no inner attacks).

Definition (Dung, 1995). S is admissible iff S is conflict -free and every x in S is acceptable with respect to S (S is defending itself against all attacks).

Definition (Dung, 1995). An argumentation semantics specifies criteria for identifying a set of arguments that can be considered „reasonable" together – these sets are called *extensions*.

Definition (Dung, 1995). A *complete* extension S is an admissible set that contains all that it can defend.

Definition (Dung, 1995). A complete extension is *preferred* if it is maximal w.r.t. to set inclusion.

Complete motivations for the argumentation's semantics can be found in (Caminada, 2009).

2.2. Backtracking for arguments

The idea to use backtracking for generating AF extensions under different semantics is not a new one. In (Nofal, 2016), the authors adapt the pruning technique of looking ahead in backtracking to decide acceptance under different semantics and to generate extensions associated to those semantics. Their algorithms and input data can be found here http://sourceforge.net/projects/argtools.

For instance, the improved algorithm for the preferred semantics builds a search-tree associated to including / excluding arguments (label IN/ label OUT) , labelling OUT all arguments attacked by an IN argument. Also, arguments that attack a IN argument are labelled MUST_OUT if there isn't an argument z IN that attacks them, and they become OUT as soon as such a z occurs. If the labelling becomes complete (all arguments being IN, OUT, UNDEC or MUST_OUT), and there still are arguments labelled MUST_OUT, then this partial solution is abandoned as hopeless. UNDEC symbolizes undecided- that is, the argument is neither IN or OUT but we refrain from labelling it strongly (possibly, because of lack of information, or it might be self-attacking; thea meaning is that some extensions contain it, some others are not). The algorithm starts the labeling from the most influential arguments, to shorten the depth of the backtrack search (an useful idea that we shall also keep in our adaptation).

Definition (Nofal, 2016). An argument x is called *influential* if it has no value assigned and for any other y unassigned too, we have $|\{x\}^{\pm}|\geq|\{y\}^{\pm}|$.

The look-ahead functions from Algorithm 4 for computing all preferred extensions (Nofal, 2016), search for arguments labelled MUST-OUT and try to find a predecessor of them (i.e. an attacker) that has not been labelled yet (is BLANK). The MUST_OUT arguments are attackers of the IN arguments that haven't received a defender yet (there is nothing IN attacking them). If there is no such a predecessor, then that argument must be put on the list to be changed its labelling.

Throughout the algorithm, references to special predecessor are continuously maintained and adapted, to ease the choosing of the next argument to be assigned, such that the chances to end in a solution are greater. Finally, what is labelled IN belongs to the admissible extension, what is OUT is outside it, and all the attacks against IN elements are defended (attacked) by IN elements.

The algorithm from (Nofal, 2016), besides using looking -ahead within backtracking, integrates some heuristics to speed -up calculus. Firstly, all arguments without any attacks are lebelled IN. Secondly, every time an argument is labelled IN, a „propagation" takes place, which includes some transformations: arguments attacked by an IN argument are labelled OUT, and the arguments that attack an IN argument become MUST_OUT (-meaning that we must have an IN attacker for them in the end, as we have already explained).

2.3. Dynamic backtracking

Backtracking is, essentially, depth -first search. A major weakness of the algorithm is that after a failure to assign a value, it traces back to the latest assigned variable, regardless of its relation to the cause of inconsistency.

In dependency-directed backtracking or in backjumping, we backtrack to the source of the problem, but the partial solution built between the current point and the source of conflict is blindly deleted. Dynamic backtracking (Ginsberg, 1993) solves this problem by remembering for each variable a list of eliminated values, each being associated to the couple variable-value that makes the eliminated value impossible. When we no longer have valid values for a position, we go back to the last conflict- that is, the last assigned variable that eliminates values for the current position, and we un-assign it, and we also delete this last assigned variable from the list of „eliminators of values" of any variable.

In Section 3, we will write this algorithm from an argumentation semantics' computation perspective. As, from our knowledge, dynamic backtracking in computing AFs' semantics has not been yet addressed, we intend to explore its possible benefits in improving efficiency, starting from the idea that argumentation is dynamic by its nature, and that dynamic backtracking performs better than looking ahead, for instance, in general.

3. DYNAMIC BACKTRACKING FOR ARGUMENTATION FRAMEWORKS

Dynamic backtracking was introduced by Ginsberg in 1993 (Ginsberg, 1993) and is briefly presented following.

Being given a Constraint Satisfaction Problem, let P be the set of currently assigned variables and *e* an elimination mechanism (that associates to each variable *i* a set of impossible values, according to constraints originating from the other assigned variables, at each step of the search).

STEP 1. Set P=E_i ={} for any i from I (E_i is the set of values that have been eliminated for the variable i);

STEP 2. If P covers all variables from I, return P. Otherwise, select i unassigned and compute E_i according to already set variables;

STEP 3. Let S be the set of unassigned variables. If S non-empty, select *v* from S. Add *(i,v)* to P and return to Step 2.

STEP 4.If S is empty, then E_i=V_i (all possible values of *i* have been eliminated). Let E be the set of all variables appearing in the explanations for each eliminated value.

STEP 5.If E is empty, return failure. Otherwise, let (j, v_j) be the last entry of P such that j belongs to E. Remove (j,v_j) from P and, for each variable *k* assigned a value after *j,* remove from E_k any eliminating explanation involving *j,* then set *i=j* and return to Step 3.

The main difference (and advantage) between dynamic backtracking and dependency -directed backtracking is that it only saves no-good (conflicts) information based on currently assigned variables, by using the eliminatory explanations (which are dynamically dropped when they are no longer relevant). As the experiments of Ginsberg show, this leads to significance improvements, and "increasing computational savings as the problems become more difficult".

We have adapted following the above algorithm to match our practical reasoning problem.
Dynamic backtracking for computing preferred extensions in AFs

Let A be the set of arguments, and let R a subset of AxA be the associated attack relation, E is finally the set of all admissible extensions.

STEP 1. Sort A descending, starting from the most influential arguments.
Initialize:
1. P= {} the set of already set arguments,

```
2.      $E_a$={} for any a in A, $E_a$= values
    eliminated for argument a, paired with the
    argument/value that eliminated them;

STEP 2. if P=A  (i.e.-finished labelling arguments),
if  P is admissible  then
            if {x in A| value (x)=IN} is not a subset of any set of E
                then  E=E U {x| value (x)=IN}
                else let a be the next argument ,

STEP 3. Let S= $V_a \setminus E_a$   ($V_a$ = possible values for a,
$E_a$= eliminated values for a)
        If S ≠ {} then  choose first value  v in S,
(S={IN, OUT, UNDEC),
                Add (a,v) → P

                If v=IN, (a,v) must be included in the eliminating explanations for all
                arguments b such that a attacks b, for value IN  (b cannot be also IN)
            else (S={}, $V_a$ =$E_a$)
                Let $El_a$= set of eliminating arguments for a
                    if $El_a$={} then - FAILURE
                    else let (a’, va’) be the last entry such that a’ is from $El_a$;
                        remove (a’, va’) from P;
                        for any b argument assigned a value after a’, remove from $E_b$ any
                        eliminating explanation that involves a’.
```

4. SIMULATIONS

We have investigated the performance of the backtracking with look-ahead and heuristics and dynamic backtracking on the same data sets from http://sourceforge.net/projects/argtools and we have obtained, so far, the results in Table 1. We have used a Intel 2.30 GHz processor. The data is a collection of argumentation frameworks with more than 100 nodes and complex attack relations.

The algorithm used for comparison is the one resumed in Section 2 (Nofal, 2016). As far as we know, it is the most recent approach to inference in an Argumentation Framework based on backtracking. It integrates global looking-ahead and a set of heuristics that we have briefly described in Section 2. Our hope is that future integration of those heuristics into dynamic backtracking for arguments will lead to improved total time.

Table 1. Experimental results

Set of data	Backtracking with look-ahead (time in seconds)	Adapted dynamic backtracking
input0	0.24	4.59
input1	0.26	4.86
input2	0.31	4.73
input3	0.28	5.16
input4	0.28	4.76
input5	0.31	4.72
input6	0.29	4.75
input7	0.34	5.69
input8	0.29	4.81
input9	0.28	5.25
input10	0.24	5.00
input11	0.36	4.77
input12	0.29	4.83
input13	0.28	4.91
input14	0.27	4.92
input15	0.23	5.09
input16	0.32	5.16
input17	0.30	4.93
input18	0.24	4.79
input19	0.28	4.89
input20	0.27	5.00

5. CONCLUSIONS AND FUTURE WORK

Because dynamic backtracking is completed and always terminated, so does the algorithm in Section 3, as it is a light adaptation of dynamic backtracking. The efficiency of algorithm in Section 3 (in time and space) should be compared to other backtracking approaches to arguments’ semantics’ computing -like we have already done with the approach from (Nofal, 2016). Although we have found inferior performance, it would be interesting to integrate new heuristics to improve this, like, for instance, iterative broadening (Ginsberg, 1992), and also to investigate how to include the „propagation“ and the other heuristics from (Nofal, 2016), to obtain a better performance.

Also, starting from dynamic constraint satisfaction programs, we could adapt the algorithm in Section 3 for the dynamic version of AFs, and then compare it to the approach of (Alfano, 2017). Dynamic constraint satisfaction contextually adds and removes the constraints that shape the search space and could therefore be assimilated with the addition or removal of attacks and arguments in an Argumentation Framework.

REFERENCES

Alfano G., Greco S., (2017) Efficient Computation of Extensions for Dynamic Abstract Argumentation Frameworks: An Incremental Approach, Proceedings of the Twenty-Sixth Int. Joint Conf. on Artificial Intelligence, 49-55.

Amgoud, L., Prade H., (2007) Formalizing practical reasoning under uncertainty: an argumentation-based approach*, IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07)*, Fremont, CA, USA, pp. 189-195, doi: 10.1109/IAT.2007.15.

Baumann R.,(2011) Splitting an argumentation framework. *LPNMR*, pages 40–53.

Baumann R., Ulbricht M., (2021) On cycles, Attackers and supporters- A Contribution to the Investigation of Dynamics in Abstract Argumentation, Ringo Baumann, Marus Ulbricht, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 1780-1786/

Caminada, M., Gabbay D., (2009) A logical account of formal argumentation, *Stud. Log.* ***93*** (2-3) 109- 145.

Dung P. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, **77**: , 321–357.

Dunne P., Woolridge M., (2009): „Complexity of abstract argumentation“, *Argumentation in Artificial Intelligence*, 85-104.

Ginsberg M.L., Harvey, W.D. (1992). Iterative broadening, Artificial Intelligence, **55,** 367-383.

Ginsberg, M., (1993) Dynamic backtracking, Journal of Artificial Intelligence Research **1** 25-46.

Nofal S., Atkinson K., Dunne P., (2016) Looking-ahead in backtracking algorithms for abstract argumentation, International Journal of Approximate Reasoning, **78**, 265-282.