A Labeling Approach to the Computation of Credulous Acceptance in Argumentation

Bart Verheij
Artificial Intelligence, University of Groningen
b.verheij@ai.rug.nl

Abstract

In recent years, the combinatorics of argumentation with arguments that can attack each other has been studied extensively. Especially, attack graphs (put in the focus of attention by Dung's seminal work 1995) have proven to be a productive tool of analysis. In this paper a new style of algorithm is presented that computes the minimal admissible sets containing or attacking the argument. It is a breadth-first algorithm using labelings. The algorithm is applied to the computation of the preferred and stable extensions of a given attack graph.

1 Introduction

The formal study of argumentation with arguments and their counterarguments in terms of argument attack graphs [Dung 1995] has proven to be fruitful. Dung introduced several different argumentation semantics, of which especially the grounded and preferred semantics have been the topic of further study. A central decision problem is to determine credulous acceptance (see, e.g., [Cayrol, Doutre & Mengin 2003], [Dunne & Bench-Capon 2002, 2003]): given an argument attack graph and an argument, determine whether the argument is an element of some preferred extension.

The present paper provides an algorithm for the computation of credulous acceptance. The approach is based on labelings, which leads to a different kind of algorithm than the more common argument game approaches. Argument games are a kind of dialogues in which a proponent tries to defend an argument, while an opponent tries to attack it. A natural idea in argument game approaches is to defend against the most recently proposed arguments first.[1] As a result, in an argument games approach, it is natural to consider the elements of an attack tree in a depth-first manner.

The main contribution of the paper is a breadth-first algorithm in terms of labelings that answers the credulous acceptance problem for the preferred semantics. Formally proven results about labelings show the correctness of the algorithm. By the nature of the algorithm short proofs and refutations of arguments are returned: the minimal admissible sets containing or attacking a given argument, respectively. As an encore, a straightforward way is provided to construct all preferred extensions of an argumentation framework, essentially by 'gluing' proofs and refutations.

[1] Natural, but not necessary: see e.g. [Cayrol *et al.* 2003].

2 Analyzing attack graphs in terms of sets and in terms of labelings

The starting point of Dung's [1995] work is an argumentation framework, which is essentially a directed graph expressing the attack relations between arguments:

Definition (1). An *argumentation framework* is a pair (*Arguments*, *Attacks*), where *Arguments* is any set, and *Attacks* is a subset of *Arguments* _ *Arguments*. The elements of *Arguments* are the arguments of the theory, the elements of *Attacks* the attacks.

When (*Arg*, *Arg*') is an attack, the argument *Arg* is said to *attack* the argument *Arg*'. A set of arguments *Args* is said to *attack* an argument *Arg* if and only if there is an element of *Args* that attacks *Arg*.

A useful shorthand for the specification of (finite) argumentation frameworks is as a set of lists of arguments. Each list of arguments in the set expresses attacks of the first argument in the list. For instance, the set of lists `{a b1 b2, b1 c1 c2, b2 c3 c4}` denotes the argumentation framework, in which the argument `a` is attacked by `b1` and `b2`, while `b1` is attacked by `c1` and `c2` and `b2` by `c3` and `c4`. It can be useful to require of this shorthand that an attack list contains all attackers of its first element, but this is not necessary and reduces flexibility of the notation.

In the rest of the paper, an argumentation framework `AF` = (*Arguments*, *Attacks*) is silently assumed. Some of Dung's central notions are the following:

Definition (2). 1. A set of arguments *Args* is *conflict-free* if it contains no arguments *Arg* and *Arg*', such that *Arg* attacks *Arg*'.

2. An argument *Arg* is *acceptable* with respect to a set of arguments *Args* if for all arguments *Arg*' in the argumentation framework the following holds:

> If *Arg*' attacks *Arg*, then there is an argument *Arg*'' in *Args*, such that *Arg*'' attacks *Arg*'.

3. A set of arguments *Args* is *admissible* if it is conflict-free and all arguments in *Args* are acceptable with respect to *Args*.

4. An admissible set of arguments *Args* is a *complete extension* if each argument that is acceptable with respect to *Args* is an element of *Args*.

5. A *preferred extension* of an argumentation framework is an admissible set of arguments, that is maximal with respect to set inclusion.

6. A conflict-free set of arguments *Args* is a *stable extension* of an argumentation framework if for any argument *Arg* of the framework that is not in *Args*, there is an argument *Arg*' in *Args*, such that *Arg*' attacks *Arg*.

The complete extension that is minimal with respect to set inclusion (which exists and is unique; see [Dung 1995]) is called the *grounded extension*.

In this paper, instead of sets, labelings are used as analysis tool. An early use of the labeling approach for Dung's abstract argumentation frameworks is by Verheij [1996], continued using a more expressive language in [Verheij 2003b]. Recently, Caminada [2006] has resumed the analysis of argumentation frameworks in terms of labelings.

Definition (3). A pair (*J*, *D*) is a *labeling* if *J* and *D* are disjoint subsets of the set *Arguments* of the argumentation framework. The elements of *J* and *D* are the *justified* and *defeated* arguments, respectively. The elements of $J \cup D$ are *labeled*, other elements of *Arguments unlabeled*.

A convenient shorthand notation for a (finite) labeling (*J*, *D*) is as a list of arguments, some of which appear between parentheses. The arguments in parentheses are those in *D*. For instance, using this shorthand, `a (b1) c1 (b2) c3` denotes the labeling in which the arguments `a`, `c1` and `c3` are justified and the arguments `b1` and `b2` defeated. Grouping the arguments in *J* and in *D* this becomes `a c1 c3 (b1 b2)`. When there are no defeated arguments, this can be indicated as `()`.

The following definition contains the main notions of the labeling approach.

Definition (4). 1. A labeling (*J*, *D*) is *conflict-free* if the set *J* is conflict-free.

2. A labeling (*J*, *D*) has *justified defeat* if for all elements *Arg* of *D* there is an element in *J* that attacks *Arg*.

3. A labeling (*J*, *D*) is *closed* if all arguments that are attacked by an argument in *J* are in *D*.

4. A conflict-free labeling (*J*, *D*) is *attack-complete* if all attackers of arguments in *J* are in *D*.

5. A conflict-free labeling (*J*, *D*) is *defense-complete* if all arguments of which all attackers are in *D* are in *J*.

6. A conflict-free labeling (*J*, *D*) is *complete* if it is both attack-complete and defense-complete.

7. A labeling (*J*, *D*) is a *stage* if it is conflict-free and has justified defeat.

Caminada's [2006] reinstatement labelings are closed complete labelings with justified defeat. The set of labelings of an argumentation framework AF is denoted as $\mathsf{Labelings}_{\mathsf{AF}}$.

The following properties summarize the relations between the set and labeling approach.

Properties (5). Let *J* be a set of arguments and *D* be the set of arguments attacked by the arguments in *J*. Then the following properties obtain:

1. *J* is conflict-free if and only if (*J*, *D*) is a labeling.

2. *J* is admissible if and only if (*J*, *D*) is an attack-complete stage.

3. *J* is a complete extension if and only if (*J*, *D*) is a complete stage.

4. *J* is a preferred extension if and only if (*J*, *D*) is an attack-complete stage with maximal set of justified arguments.

5. *J* is a stable extension if and only if (*J*, *D*) is a labeling with no unlabeled arguments.

Proof omitted.

It is useful to introduce a name for minimal admissible sets:

Definition (6). An *admissible proof* (or *proof*, for short) for an argument *Arg* is an admissible set containing *Arg* that is minimal with respect to set inclusion. An *admissible refutation* (or *refutation*, for short) against an argument *Arg* is an admissible proof for an attacker of *Arg*.

It is important to note that the existence of a refutation of an argument does not only imply that the argument is in no preferred extension, but only that there is a preferred extension attacking the argument.

3 Computing credulous acceptance

Before we turn to the algorithm some remarks about the approach are in use. What is the nature of the 'solutions' we are looking for? Consider an argumentation framework in which there is an argument `a` with two attackers `b1` and `b2`, which on their turn have two attackers each (arguments `c1` to `c4`), and these are again each attacked by two attackers (arguments `d1` to `d8`). The relevant part of the attack graph is shown in Figure 1. It is assumed that the figure shows all attackers at at most three attack levels from argument `a`. There are for instance no other attackers of `a` than `b1` and `b2`. The dots indicate that higher attack levels are omitted.

Given this partial view of the attack graph, what can be said about the proofs and refutations of the argument `a`? Of course we cannot determine 'complete' proofs and refutations, but we know what the 'three-levels deep' partial proofs and refutations must look like if they exist. The figure shows a possible partial proof (on the left) and a possible partial refutation. In the partial proof, `a` is justified (light grey). Hence both its attackers must be defeated (dark grey). Since a proof is minimal, a partial proof only needs to contain one attacker against `a`'s attackers. Therefore the second attack level allows four different possible partial proofs, one for each minimal choice of `c`'s attacking the `b`'s. In the figure, the arguments `c1` and `c3` are chosen and consequently labeled as justified. The third attack level is then again determined: the attackers of the `c`'s that are justified must all be defeated. As a result, there exist four possible partial proofs of the argument `a`. A similar analysis shows that there are eight possible partial refutations, one of which is shown on

the right in the figure. Each possible partial refutation is determined by a choice between the two b's at attack level 1 followed by two choices from a pair of d's at attack level 3.

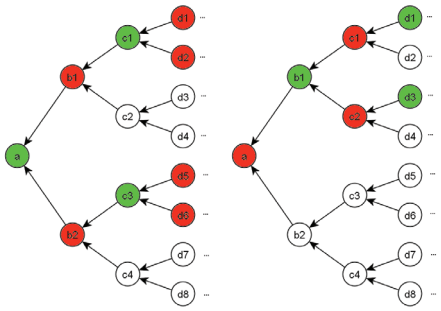

Figure 1: a possible partial proof and refutation for a three levels deep part of an attack graph

Note that possible partial proofs and refutations are indeed only possible, in the sense that they are not necessarily the part of an *actual* proof or refutation. The unshown parts of the attack graph may obstruct a possible partial proof or refutation from being extended to an actual one. For instance, the shown possible partial proof (on the left) is not an actual partial proof if the figure already shows all attackers of the argumentation framework since, in that case, there is no way of defending against the d's. Using the terminology associated with labelings, the defeat of the d's cannot be justified. Similarly, the shown possible partial refutation is not an actual partial refutation if for instance d1 has exactly one attacker that is itself unattacked.

Notwithstanding this possibility of possible partial proofs and refutations becoming 'blocked' by knowing more about the attack graph, each is genuinely possible, in the sense that there exist extended attack graphs in which a possible partial proof or refutation is actualized. For instance, the possible partial proof shown is realized when the attack graph contains exactly one unattacked attacker for each of the defeated *D*'s, and the possible partial refutation is actualized when the attack graph is already complete.

The discussion of the possible partial proofs in this example provides a good illustration of the computational approach in this paper. Formally, two functions on labelings are central in the algorithm: ExtendByAttack and ExtendByDefense. Both return a set of labelings given a labeling as input. The function ExtendByAttack: $\text{Labelings}_{AF} \rightarrow \wp(\text{Labelings}_{AF})$ adds all attackers of the justified arguments of a labeling to the set of defeated arguments:

> $\text{ExtendByAttack}_{AF}((J, D)) := \{(J, D') \in \text{Labelings}_{AF} \mid D'$ is the set D extended with all arguments attacking arguments in $J\}$

The function ExtendByAttack returns a set of labelings that contains one element or is empty. $\text{ExtendByAttack}_{AF}(J, D)$ is empty if and only if J is self-attacking: if one of the attackers of an argument in J is an element of J, then (J, D'), where D' is as in the definition of the function, is not a labeling since this requires that $J \cap D'$ is empty.

The function ExtendByDefense: $\text{Labelings}_{AF} \rightarrow \wp(\text{Labelings}_{AF})$ returns the set of labelings that result from extending the set of justified arguments of the input labeling in a minimal way, such that all defeated arguments of the input labeling are attacked by the justified arguments:

> $\text{ExtendByDefense}((J, D)) := \{(J', D) \in \text{Labelings}_{AF} \mid J'$ is a conflict-free, minimal set of arguments $\supseteq J$, such that for all arguments *Arg* in D there is an argument *Arg*' in J' that attacks *Arg*$\}$[2]

Extension by defense returns a set of labelings that can contain zero, one or several elements. The following example illustrates the two functions.

Example (7). Consider the argumentation framework AF = {a b1 b2, b1 c1 c2, b2 c3 c4, c1 d1 d2, c2 d3 d4, c3 d5 d6, c4 d7 d8}. It corresponds to the attack graph shown in Figure 1, assuming that there are no further arguments and attacks than the ones shown. Some representative examples of applying the two functions for this framework are as follows. They result from the repetitive application of the two functions starting with the labeling a (), in which a is justified, as a 'seed':

> ExtendByAttack(a ()) = {a (b1 b2)}
> ExtendByDefense(a ()) = {a ()}
> ExtendByAttack(a (b1 b2)) = {a (b1 b2)}
> ExtendByDefense(a (b1 b2)) = {a (b1 b2) c1 c3, a (b1 b2) c1 c4, a (b1 b2) c2 c3, a (b1 b2) c2 c4}
> ExtendByAttack(a (b1 b2) c1 c3) = {a (b1 b2) c1 c3 (d1 d2 d5 d6)}
> ExtendByDefense(a (b1 b2) c1 c3) = {a (b1 b2) c1 c3}
> ExtendByAttack(a (b1 b2) c1 c3 (d1 d2 d5 d6)) = {a (b1 b2) c1 c3 (d1 d2 d5 d6)}
> ExtendByDefense(a (b1 b2) c1 c3 (d1 d2 d5 d6)) = ∅

Starting with the labeling (a) in which the argument a is defeated, the following are found:

> ExtendByAttack((a)) = {(a)}
> ExtendByDefense((a)) = {(a) b1, (a) b2}
> ExtendByAttack((a) b1) = {(a) b1 (c1 c2)}
> ExtendByDefense((a) b1) = {(a) b1}
> ExtendByAttack((a) b1 (c1 c2)) = {(a) b1 (c1 c2)}
> ExtendByDefense((a) b1 (c1 c2)) = {(a) b1 (c1 c2) d1 d3, (a) b1 (c1 c2) d1 d4, (a) b1 (c1 c2) d2 d3, (a) b1 (c1 c2) d2 d4}
> ExtendByAttack((a) b1 (c1 c2) d1 d3) = {(a) b1 (c1 c2) d1 d3}
> ExtendByDefense((a) b1 (c1 c2) d1 d3) = {(a) b1 (c1 c2) d1 d3}

The functions ExtendByAttack and ExtendByDefense can be extended to $\wp(\text{Labelings}_{AF}) \rightarrow \wp(\text{Labelings}_{AF})$ by stipulating the following (where L denotes a set of labelings):

> ExtendByAttack(L) := {(*J*, *U*) | There is a (*J*', *U*') ∈ L, such that (*J*, *U*) ∈ ExtendByAttack((*J*', *U*'))}
> ExtendByDefense(L) := {(*J*, *U*) | There is a (*J*', *U*') ∈ L, such that (*J*, *U*) ∈ ExtendByDefense((*J*', *U*'))}

We can now formally define the partial proof and partial refutation options of an argument in an argumentation framework. They are the labelings that result from the alternating application of the (nonmonotonic) functions Extend-

[2] A somewhat more exact, but less readable wording of the constraint on sets J' is the following: J' is a conflict-free set of arguments that is minimal with respect to set inclusion among the sets of arguments J'' for which it holds that for all arguments *Arg* in D there is an argument *Arg*' in J' that attacks *Arg*.

ByAttack and ExtendByDefense, starting with different seeds (where n ranges over the natural numbers ≥0):

$\text{PartialProofOptions}_0(Arg) := \{Arg\ ()\}$
$\text{PartialProofOptions}_{2n+1}(Arg) := \text{ExtendByAttack}(\text{PartialProofOptions}_{2n}(Arg))$
$\text{PartialProofOptions}_{2n+2}(Arg) := \text{ExtendByDefense}(\text{PartialProofOptions}_{2n+1}(Arg))$

$\text{PartialRefutationOptions}_0(Arg) := \{(Arg)\}$
$\text{PartialRefutationOptions}_{2n+1}(Arg) := \text{ExtendByAttack}(\text{PartialProofOptions}_{2n}(Arg))$
$\text{PartialRefutationOptions}_{2n+2}(Arg) := \text{ExtendByDefense}(\text{PartialProofOptions}_{2n+1}(Arg))$

The elements of $\text{PartialProofOptions}_n(Arg)$ are referred to as the *partial proof options* of the argument *Arg* at attack level n, the elements of $\text{PartialRefutationOptions}_n(Arg)$ are its *partial refutation options* at attack level n.

Definition (8). A *proof option* is a labeling (J, D), such that there is an $n_0 \geq 0$, such that for all $n \geq n_0$ it holds that $(J, D) \in \text{PartialProofOptions}_n(Arg)$. A *refutation option* is a labeling (J, D), such that there is an $n_0 \geq 0$, such that for all $n \geq n_0$ it holds that $(J, D) \in \text{PartialRefutationOptions}_n(Arg)$.

The lowest value of n_0 as in this definition is the *depth* of the proof or refutation option.

It holds that, if (J, D) is a proof or refutation option, then $(J, D) \in \text{ExtendByAttack}((J, D))$ and $(J, D) \in \text{ExtendByDefense}((J, D))$.

Example (7), continued. The following table shows that there are no proof options and eight refutation options for the argument a.

Partial proof options

0: a
1: a (b1 b2)
2: a (b1 b2) c1 c3, a (b1 b2) c2 c3, a (b1 b2) c1 c4, a (b1 b2) c2 c4
3: a (b1 b2) c1 c3 (d1 d2 d5 d6), a (b1 b2) c2 c3 (d3 d4 d5 d6), a (b1 b2) c1 c4 (d1 d2 d7 d8), a (b1 b2) c2 c4 (d3 d4 d7 d8)
4: none, since the d's cannot be defended against

Partial refutation options

0: (a)
1: (a) b1, (a) b2
2: (a) b1 (c1 c2), (a) b2 (c3 c4)
3+: (a) b1 (c1 c2) d1 d3, (a) b1 (c1 c2) d1 d4, (a) b1 (c1 c2) d2 d3, (a) b1 (c1 c2) d2 d4, (a) b2 (c3 c4) d5 d7, (a) b2 (c3 c4) d5 d8, (a) b2 (c3 c4) d6 d7, (a) b2 (c3 c4) d6 d8

Example (9). AF = {a b c, b h j k, c e f, e g, f f, g e, j a}. This example is used by Cayrol, Doutre & Mengin [2003, p. 379] to illustrate their approach in terms of dialogues. It further illustrates that partial proof and refutation options are gradually constructed and can be discarded. The table below shows that there are two proof options and no refutation options for the argument a. The computation of the proof options is complete after the second level. Since b has three attackers and c has two, there are six possible labelings that must be considered. Four of them are immediately discarded since they are not conflict-free. The remaining two are proof options that correspond to the two minimal admissible sets containing a. The computation of the refutation options shows that none remain from the third level onward. Indeed there exists no admissible set attacking a.

Partial proof options

0: a
1: a (b c)
2: a (b c) e h
~~a (b c) e j~~ a attacks j
a (b c) e k
~~a (b c) f h~~ f attacks itself
~~a (b c) f j~~ a attacks j
~~a (b c) f k~~ f attacks itself
3+: a (b c) e h (g), a (b c) e k (g)

Partial refutation options

0: (a)
1: (a) b, (a) c
2: (a) b (h j k), (a) c (e f)
3+: ~~(a) b (h j k) a~~ a is justified & defeated
~~(a) c (e f) f g~~ f is justified & defeated

Example (10). AF = {a b, b c1 c2, c1 d, c2 e, d c1 c2, e c1 c2}. This argumentation framework shows how the labeling approach deals with 'dependent choices'. When trying to credulously prove a, a choice between c1 and c2 must be made in order to defend against b. Further down the attack graph, again a choice between c1 and c2 becomes available (to defend against d and e). In a minimal proof, the choice must be kept constant, as actually happens in our labeling approach:

0: a; 1: a (b); 2: a (b) c1, a (b) c2; 3+: a (b) c1 (d), a (b) c2 (e)

Since d is already attacked by c1 in the partial proof option a (b) c1 (d) that appears at the third level, the c2-option of attacking d is not considered.

We will show that partial proof and refutation options indeed produce proofs and refutations. The following lemma is needed.

Lemma (11). 1. If (J, D) is a conflict-free labeling, then the elements of $\text{ExtendByAttack}((J, D))$ and $\text{ExtendByDefense}((J, D))$ are conflict-free.

2. A conflict-free labeling (J, D) is attack-complete if and only if $\text{ExtendByAttack}((J, D)) = \{(J, D)\}$.

3. A conflict-free labeling (J, D) has justified defeat if and only if $(J, D) \in \text{ExtendByDefense}((J, D))$.

4. If (J, D) is a conflict-free labeling, such that $(J, D) \in \text{ExtendByAttack}((J, D))$ and $(J, D) \in \text{ExtendByDefense}((J, D))$, then J is admissible.

Proof omitted.

Theorem (12). If (J, D) is a proof or refutation option, then J is admissible.

Proof. Proof and refutation options are the result of the consecutive application of the functions ExtendByAttack and ExtendByDefense starting from seeds that are conflict-free. Since the functions maintain conflict-freeness (Lemma (11) under 1), proof and refutation options are conflict-free. Since proof and refutation options are by definition fixed points of the functions ExtendByAttack and ExtendByDefense as in Lemma (11) under 4, the theorem follows.

Theorem (13). Let AF be a finite argumentation framework, J a set of arguments of the framework and D the set of arguments attacking arguments in J.

1. If J is a proof of an argument *Arg*, then (J, D) is a proof option for *Arg*.

2. If J is a refutation of an argument Arg, then (J, D) is a refutation option for Arg.

Proof. 1. Let J be a proof of Arg. We inductively construct a sequence $(J(n), D(n))$ for natural numbers $n = 0, 1, 2, \ldots$, as follows:

$J(0) := \{Arg\}$
$J(2n+1) := J(2n)$
$J(2n+2) :=$ some minimal subset of J that defends against all elements of $D(2n+1)$
$D(0) := \emptyset$
$D(2n+1) := D(2n) \cup \{Arg' \mid Arg'$ attacks an argument in $J(2n)\}$
$D(2n+2) := D(2n+1)$

The $J(2n+2)$-step is non-constructive, but can be made by the admissibility of J. It follows by induction from the definitions and from the admissibility of J that, for all n, $(J(n), D(n)) \in \text{PartialProofOptions}_n(Arg)$ and $J(n) \subseteq J$. When n is larger then the depth of the attack graph (defined as the maximum length of a non-looping or minimally looping attack sequence[3]), we have that $J(n) = J$ (this uses the minimality of J), and $(J(n), D(n+1)) = (J, D)$ is a proof option. The proof of the second part is similar.

Example (14). AF = {a b, b c e, c d, d e}. This example shows that sometimes a proof option contains redundant justified arguments, and hence does not correspond to a proof (which by definition must be minimal). The argument a has two proof options, viz. a c e (b d) and a e (b), while only the latter corresponds to a proof. The non-minimally admissible set {a c e} arises since an early choice of defense turns out to be redundant further down the attack tree. The trace of partial proof options of a is as follows:
0: a; 1: a (b); 2: a (b) c, a (b) e; 3: a (b) c (d), a (b) e; 4+: a (b) c (d) e, a (b) e
The choice of c at level 2 becomes obsolete. Note also that the proof option a e (b) is already complete at level 2.

Example (15). AF = {a b, b c1 c2, c1 d1 d2 d3, c2 d4, d1 e1, d2 e2, d3 e3, d4 e1, e1 f1 f2, e2 f3, f1 c2, f2 e2, f3 e3}. This example shows that a proof option of minimal depth does not necessarily have a minimal set of justified arguments, hence does not necessarily return a proof. This argumentation framework has two proof options for the argument a. The first is a (b) c1 (d1 d2 d3) e1 e2 e3 (f1 f2 f3) c2 (d4) of depth 7, the second is a (b) c2 (d4) e1 (f1 f2) e2 (f3) e3 of depth 8. The latter has {a, c2, e1, e2, e3} as set of justified arguments, which is a proper subset of {a, c1, c2, e1, e2, e3}, the set of justified arguments of the former. Hence, the only proof of a corresponds to the deeper proof option.

Corollary (16). If Arg has no proof, then Arg has no proof options. If Arg has no refutation, then Arg has no refutation options.

Corollary (17). For finite argumentation frameworks, if Arg has no proof, then there is an $n_0 \geq 0$, such that for all $n \geq n_0$ it holds that $\text{PartialProofOptions}_n(Arg) = \emptyset$. If Arg has no refutation, then there is an $n_0 \geq 0$, such that for all $n \geq n_0$ it holds that $\text{PartialRefutationOptions}_n(Arg) = \emptyset$.

On the basis of the above, an algorithm can be built that computes proof options of minimal depth for an argument Arg given a finite argumentation framework AF:

Step 0: L := {*Arg* ()}.
Step 1: L' := ExtendByDefense(ExtendByAttack(L)).
Step 2: If there is a labeling $(J, D) \in$ L' $\cap$ L, then return L' $\cap$ L.
Step 3: If L = L', then stop and return L.
Step 4: Goto step 1.

For finite argumentation frameworks, the algorithm stops since extension by attack and by defense either discards a labeling, keeps it constant or increases the set of labeled arguments in the labeling. When Arg has a proof, the algorithm returns the proof options for Arg of minimal depth. When Arg has no proof, the algorithm returns the empty set (Corollary (17)). By leaving out Step 2, the algorithm returns all proof options of Arg, hence all its proofs (Theorem (13)). When Step 0 is replaced by L := {(*Arg*)}, the algorithm computes refutation options.

The algorithm has been implemented (in Delphi 7 under Windows XP) and tested using a set of test examples. It computes all proof and refutation options for all arguments of a given argumentation framework. One detail of the implementation is not obvious: the way in which extension by attack and by defense has actually been implemented. The algorithm keeps track of the newly labeled arguments. For extension by attack, only attackers of the arguments newly labeled as justified are added. For extension by defense, only defenses against the arguments newly labeled as defeated are added. To find minimal defense sets as in the definition of extension by defense, the minimality is checked of all sets that contain exactly one attacker against all arguments that are newly labeled as defeated and that are not yet attacked by a justified argument.

Example (18). DOUBLE(n) := {a_{i0} $a_{(i+1)0}$ $a_{(i+1)1}$, a_{i1} $a_{(i+1)0}$ $a_{(i+1)1}$ | i = 0, ..., n - 1}. DOUBLE*(n) := DOUBLE(n) $\cup$ {a_{i0} b_i | i = 1, ..., n}. This example illustrates the exponential complexity of the computation of proof options (cf. the results by Dunne & Bench-Capon [2003]). The number of proof/refutation options of DOUBLE(n) grows exponentially, while DOUBLE*(n) has only one proof option or one refutation option. The nature of DOUBLE*(n) is such that it may be necessary to consider all (partial) proof/refutation options of DOUBLE(n) to find this unique solution.

Example (19). DEEP&WIDE(n, m) := {a b, b c_{i0}, c_{ij} d_{ij}, d_{kj} $c_{(k+1)j}$ | i = 0, 1, ..., n - 1; j = 0, 1, ..., m - 1; k = 0, 1, ..., m - 2}. DEEP&WIDE*(n, m) := DEEP&WIDE(n, m) $\cup$ {b z}. This example illustrates the possible advantage of the breadth-first character of our approach. The argument a has exactly one proof {a, z}. It corresponds to the proof option a (b) z. The c-defenses against b all fail at the end of the c-d sequences, since the final d's cannot be defended against. Since the proof option has depth 2, our approach quickly finds it. Of course, the full width of the example (which increases with the parameter n) must be considered at attack

[3] An attack sequence is minimally looping of it is not non-looping, while all its proper initial parts are.

level 2, but then the algorithm stops. For higher values of *n*, a depth-first approach will have a lower chance of entering the z-branch quickly and a higher chance of lingering in the c-d chains. For higher values of *m*, this is more costly.

The algorithm above has been applied to compute the preferred and stable extensions of argumentation frameworks, as follows:

Step 0: Find all proofs and refutations of all arguments.
Step 1: Find the set *DA* of all credulously ambiguous arguments.
Step 2: Choose a labeling (*J*, *D*) of *DA* in which all elements of *DA* are labeled (the 'disambiguation').
Step 4: If, for all elements of *DA*, there exists a proof or a refutation option that is compatible with (*J*, *D*), then goto Step 5, else goto Step 6.
Step 5: Collect all proof and refutation options compatible with (*J*, *D*). Output their union as a preferred extension. When it has no unlabeled arguments, output it as a stable extension.
Step 6: Choose the next disambiguation and goto Step 4. If none exists, stop.

This algorithm uses the properties that the union of compatible admissible sets is again admissible and that two different preferred extensions must differ on some credulously ambiguous argument.

4 Related research

Chesñevar & Simari [2006] and Dung, Kowalski & Toni [2006] also deal with computational aspects of argumentation. Both focus on a different language than Dung's [1995] attack graphs. Chesñevar & Simari include a notion of dialectical constraints and Dung, Kowalski & Toni treat assumption-based frameworks. Dunne & Bench-Capon [2002, 2003] use attack graphs and pay attention to the length of disputes that show whether an argument can be successfully justified or refuted. They prove a relation between their formally defined notions of complexity of a dispute instance and the rank of an argument (Theorem 4 on p. 232). Their analyses focus on theoretical complexity results rather than actual computation. Doutre & Mengin [2001] and Cayrol *et al.* [2003] use a subset enumeration algorithm for the computation of preferred extensions and a dialogue approach for credulous acceptance. They apply terminology related to graph theory, which is formally close to some of the labeling terminology used here. Verheij's [2003a] ArguMed software computes stable extensions for an argumentation language extending that of Dung's. The algorithm computes the grounded extension and then tries to extend it to a stable extension. The mentioned papers contain further references.

5 Conclusion

In this paper, the credulous acceptance problem has been approached in terms of labelings. The new notions of (partial) proof and refutation options have been proposed and formally analyzed. As an application, an algorithm has been described that computes whether an argument is credulously acceptable. Variants of the algorithm compute all minimal admissible sets of arguments containing or attacking a given argument. A negative result (which also applies to existing argument game approaches) is that in a special kind of situation a non-minimal admissible set can be returned (Examples (14) and (15)), but always alongside the minimal ones (Theorem (13)). The examples show that this is unavoidable: since computing admissibility requires that the attack tree is gradually explored, only a kind of 'tree-minimality' can be maintained. To ensure set-minimality, an ultimate check is needed. An innovation is that it takes the labeling approach. The labeling approach leads to an algorithm of a different flavour than the more common argument game approaches. In argument game approaches, it is a natural choice to follow up on the last proposed argument, which leads to a depth-first consideration of the attack tree. In contrast, the algorithm proposed here is the first breadth-first algorithm, which hence avoids the unnecessarily deep consideration of attack graphs (cf. Example (19)). As an application of the algorithm, it is shown how preferred and stable extensions can be computed given the proofs and refutations of the arguments in an argumentation framework by a kind of gluing. Hence, the connection between the computation of credulous acceptance and of preferred extensions is made explicit, in contrast with Cayrol, Doutre & Mengin's work, who compute preferred extensions in terms of set enumeration and credulous acceptance in terms of dialogues. The algorithm has been implemented and tested using a set of examples. It has been made available for download (http://www.ai.rug.nl/~verheij/comparg/).

References

Caminada, M. (2006). On the Issue of Reinstatement in Argumentation. *Technical report UU-CS-2006-023*.

Cayrol, C., Doutre, S., & Mengin, J. (2003). On Decision Problems Related to the Preferred Semantics for Argumentation Frameworks. *Journal of Logic and Computation 13*(3), pp 377-403.

Chesñevar, C. I., & Simari, G. R. (2006). An Abstract Model for Computing Warrant in Skeptical Argumentation Frameworks, *11th Intl. Workshop on Non-Monotonic Reasoning (NMR 2006)*.

Doutre, S., & Mengin, J. (2001). Preferred Extensions of Argumentation Frameworks: Query Answering and Computation. In *IJCAR 2001, LNAI 2083*, pp 272-288. Berlin: Springer-Verlag.

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

Dung, P. M., Kowalski, R. A., & Toni, F. (2006). Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligence 170*, pp 114-159.

Dunne, P. E., & Bench-Capon, T. J. M. (2002). Research note. Coherence in finite argument systems. *Artificial Intelligence 141*, pp 187-203.

Dunne, P. E., & Bench-Capon, T. J. M. (2003). Two party immediate response disputes: Properties and efficiency. *Artificial Intelligence 149*, pp 221-250.

Verheij, B. (1996). Two approaches to dialectical argumentation: admissible sets and argumentation stages. In J.-J. C. Meyer & L. C. van der Gaag (eds.), *NAIC'96. Proceedings of the Eighth Dutch Conference on Artificial Intelligence* (pp 357-368). Utrecht: Universiteit Utrecht.

Verheij, B. (2003a). Artificial argument assistants for defeasible argumentation. *Artificial Intelligence 150*(1-2), pp 291-324.

Verheij, B. (2003b). DefLog: on the logical interpretation of prima facie justified assumptions. *Journal of Logic and Computation 13*(3), pp 319-346.