A New Labelling Algorithm for Generating Preferred Extensions of Abstract Argumentation Frameworks

Samer Nofal[1], Katie Atkinson[2], Paul E. Dunne[2] and Ismail Hababeh[1]
[1]*Department of Computer Science, German Jordanian University, Jordan*
[2]*Department of Computer Science, University of Liverpool, U.K.*

Keywords: Abstract Argumentation, Argumentation Semantics, Labelling Semantics, Argumentation Algorithm, Labelling Algorithm, Preferred Semantics.

Abstract: The field of computational models of argument aims to provide support for automated reasoning through algorithms that operate on arguments and attack relations between them. In this paper we present a new labelling algorithm that lists all preferred extensions of an abstract argumentation framework. The new algorithm is enhanced by a new pruning strategy. We verified our new labelling algorithm and showed that it enumerates preferred extensions faster than the old labelling algorithm.

1 INTRODUCTION

The study of computational argumentation is a major field of artificial intelligence, see for example (Atkinson et al., 2017; Modgil et al., 2013). Abstract argumentation frameworks of (Dung, 1995) (AFs for short) are directed graphs with nodes representing *abstract arguments* while directed edges denote *attacks* between arguments. In spite of their simplicity, AFs are an effective mechanism for decision making in different domains, see for example (Longo and Dondio, 2014; Bench-Capon et al., 2015; Tamani et al., 2015).

A fundamental issue arises in the context of AFs concerning identifying which arguments are collectively accepted in a given AF. To this end, an argumentation semantics defines rules under which one can compute sets of accepted arguments, what are so-called *extensions*. In the literature we find several proposals motivating different argumentation semantics. These varied semantics give a wide range of selection among which one can choose what best fit the needs of the target application, see (Baroni et al., 2011) for a comprehensive review of argumentation semantics. In this paper we are concerned with the problem of listing all extensions of a given AF under *preferred* argumentation semantics, which is one of the most studied semantics. We give a precise definition for preferred semantics in section 2.

A labelling algorithm for listing preferred extensions of a given AF is basically a search algorithm that expands an abstract binary tree typically in a depth-first manner. The nodes of the tree represent different states of the input AF, what are so-called *labellings*. Labellings of a given AF are defined by a total function that maps arguments from the AF to elements from a predefined set of statuses, what are so-called *argument labels*.

The objective of this paper is to present a new, more efficient labelling algorithm that lists all preferred extensions of a given AF. The current state-of-the-art labelling algorithm for preferred extension enumeration is presented in (Nofal et al., 2016). However, in this paper we enhance the algorithm of (Nofal et al., 2016) by a more efficient pruning strategy to speed up preferred extension generation. We implemented our new algorithm and verified that the new pruning strategy resulted in a faster preferred extension enumeration. Although we focus on the problem of listing preferred extensions, we believe that the notions of the new pruning strategy are transferable (with appropriate adjustments) to other computational problems in the context of abstract argumentation frameworks.

In section 2 we give a necessary background material. In section 3 we recall the state-of-the-art labelling algorithm for generating all preferred extensions. In section 4 we present our new labelling algorithm for listing preferred extensions. We verify the efficiency of the new algorithm in section 5. We conclude the paper in section 6.

Nofal, S., Atkinson, K., Dunne, P. and Hababeh, I.
A New Labelling Algorithm for Generating Preferred Extensions of Abstract Argumentation Frameworks.
DOI: 10.5220/0007737503400348
In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 340-348
ISBN: 978-989-758-372-8

2 PRELIMINARIES

In the following definition we recall the notion of abstract argumentation frameworks as introduced in the seminal work of (Dung, 1995).

Definition 1 (Abstract Argumentation Frameworks)**.** *An abstract argumentation framework* AF *is a pair* (A,R) *where* A *is a set of abstract arguments and* $R \subseteq A \times A$ *is called the attack relation.*

We refer to $(x,y) \in R$ as x *attacks* y (or y is *attacked by* x). We denote by $\{x\}^-$ respectively $\{x\}^+$ the subset of A containing those arguments that attack (respectively are attacked by) the argument x, and so we use $\{x\}^\pm$ to represent the set $\{x\}^+ \cup \{x\}^-$. For a set of arguments $S \subseteq A$, we define

$$\begin{array}{lcl} S^- & \equiv & \{\, y \in A \mid \exists\, x \in S \text{ s.t. } y \in \{x\}^-\} \\ S^+ & \equiv & \{\, y \in A \mid \exists\, x \in S \text{ s.t. } y \in \{x\}^+\} \end{array}$$

We denote by $S^\pm$ the union $S^+ \cup S^-$. We say $S \subseteq A$ attacks $T \subseteq A$ (or T is attacked by S) if and only if $S^+ \cap T \neq \emptyset$. $S \subseteq A$ attacks $x \in A$ (or x is attacked by S) if and only if $x \in S^+$. Given a subset $S \subseteq A$, then

- $x \in A$ is *acceptable* w.r.t. S if and only if for every $y \in \{x\}^-$, there is some $z \in S$ for which $y \in \{z\}^+$.
- S is *conflict free* if and only if for each $(x,y) \in S \times S$, $(x,y) \notin R$.
- S is *admissible* if and only if it is conflict free and every $x \in S$ is acceptable w.r.t. S.
- S is a *preferred extension* if and only if it is a maximal (w.r.t. set inclusion) admissible set.

In this paper we are concerned with the following problem: given an AF $H = (A,R)$, enumerate the preferred extensions of H.

We give now a general account of a labelling algorithm that generates all preferred extensions. As said earlier, a labelling algorithm expands a conceptual binary search tree in a depth-first way. The algorithm forks to a left node if it decides to *include* an argument in a current under-construction extension. On the other hand the algorithm forks to a right node if it decides to *exclude* an argument from the current under-construction extension. Once no argument is left un-included or un-excluded, the algorithm backtracks if the current under-construction extension is not admissible. Equally, the algorithm backtracks to find another extension. Take the AF of figure 1, then a labelling algorithm would generate the search tree visualized in figure 2.

In the next section we recall a precise definition of the state-of-the-art labelling algorithm for listing preferred extensions.

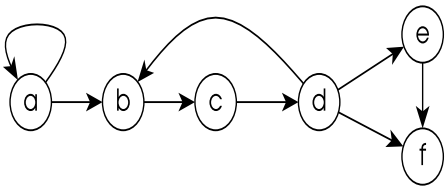

Figure 1: Argumentation Framework 1 (AF_1).

3 THE STATE-OF-THE-ART LABELLING ALGORITHM FOR PREFERRED EXTENSIONS

We recall the state-of-the-art labellling algorithm, presented in (Nofal et al., 2016), for listing all preferred extensions of a given AF. In section 2 we presented an extension-based definition for preferred argumentation semantics. Alternatively, preferred semantics can be described in terms of labellings, which are mappings that relate every argument in a given AF to a label in $\{in, out, undec\}$. For example, let (A,R) be an AF and $S \subseteq A$ be an admissible set then the equivalent labelling for S is described by a total mapping $Lab : A \rightarrow \{in, out, undec\}$ where $S = \{x \mid Lab(x) = in\}$, $S^\pm = \{x \mid Lab(x) = out\}$ and $A \setminus (S \cup S^\pm) = \{x \mid Lab(x) = undec\}$. For a thorough presentation on labelling semantics see (Caminada and Gabbay, 2009). Although a 3-label mapping is probably sufficient for characterizing extensions, additional labels have been found useful for enhancing the efficiency of extension enumeration. Thereby the labelling algorithms of (Nofal et al., 2016; Nofal et al., 2014b; Nofal et al., 2014a) use instead a 5-label total function that maps arguments to labels from $\{in, out, undec, blank, must\_out\}$.

As noted earlier, a labelling algorithm for preferred extension enumeration is basically a depth-first search that explores a conceptual binary tree. At the root node of the search tree, all arguments of the given AF are initially labelled according to the following specification.

Definition 2 (Initial Labelling)**.** *Let* $H = (A,R)$ *be an* AF *and* $S \subseteq A$ *be the set of self-attacking arguments. Then the initial labelling of* H *is defined by the union:* $\{(x, blank) \mid x \in A \setminus S\} \cup \{(y, undec) \mid y \in S\}$.

At any node of the search tree the algorithm transitions to a left node by selecting an argument x with $Lab(x) = blank$ and subsequently modify argument labels as specified in the following definition.

Definition 3 (*in* transitions)**.** *Let* $H = (A,R)$ *be an* AF*,* $Lab : A \rightarrow \{in, out, undec, blank, must\_out\}$ *be a total*

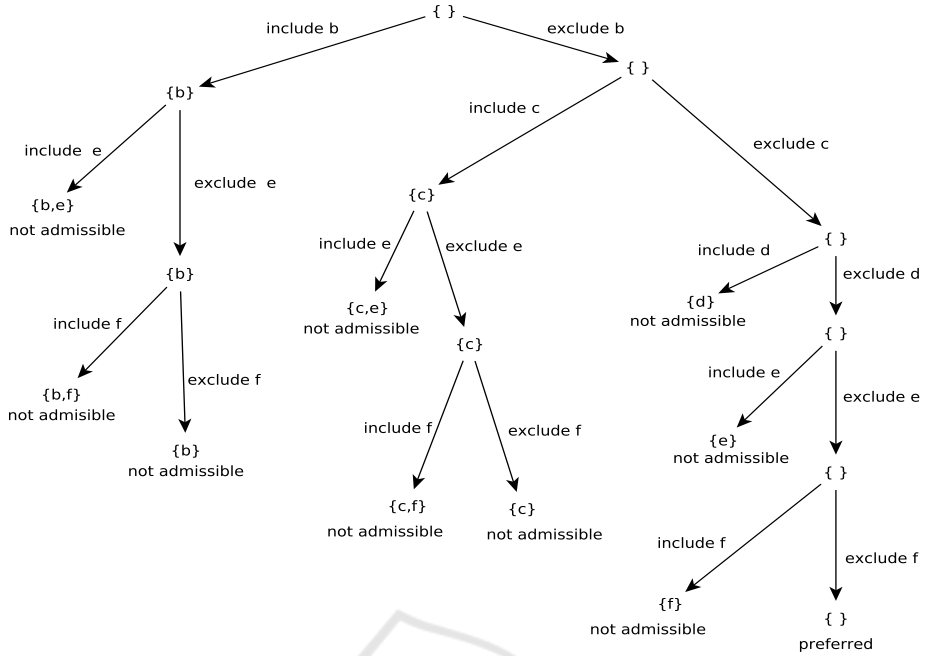

Figure 2: A search tree that would be expanded by a basic labelling algorithm for listing the preferred extensions of AF_1, which is depicted in figure 1.

mapping, and x be an argument with $Lab(x) = blank$ then in_trans(x,H,Lab) is defined by the following actions:

1. $Lab' \leftarrow Lab$.
2. $Lab'(x) \leftarrow in$.
3. *for each* $y \in \{x\}^+$, $Lab'(y) \leftarrow out$.
4. *for each* $y \in \{x\}^-$ *with* $Lab'(y) \neq out$, $Lab'(y) \leftarrow must\_out$.
5. *return* Lab'.

After the algorithm finished exploring the left subtree, that is induced by an *in* transition, it expands a right node by an *undec* transition as described in the following definition.

Definition 4 (*undec* Transitions)**.** *Let $H = (A,R)$ be an* AF*, $Lab : A \rightarrow \{in, out, undec, blank, must\_out\}$ be a total mapping and x be an argument with $Lab(x) = blank$. Then und_trans(x,H,Lab) is defined by the following actions:*

1. $Lab' \leftarrow Lab$.
2. $Lab'(x) \leftarrow undec$.
3. *return* Lab'.

A labelling algorithm would reach a leaf node if there are no *blank* arguments, we call such leaf nodes *terminal labellings*.

Definition 5 (Terminal Labellings)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \rightarrow \{in, out, undec, blank, must\_out\}$ be a total mapping. Then Lab is a terminal labelling of H if and only if for each $x \in A$, $Lab(x) \neq blank$.*

We call terminal labellings with $\{x \mid Lab(x) = in\}$ being admissible by *admissible labellings*. It follows directly from the definition of admissible sets that if a terminal labelling, for a given AF, does not map any argument to *must_out* then the set $\{x \mid Lab(x) = in\}$ is admissible.

Definition 6 (Admissible Labellings)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \rightarrow \{in, out, undec, blank, must\_out\}$ be a total mapping. Then Lab is an admissible labelling of H if and only if Lab is terminal and there is no $x \in A$ with $Lab(x) = must\_out$.*

Conversely we denote by *rejected labellings* (or occasionally *dead-end labellings*) the terminal labellings with $\{x \mid Lab(x) = in\}$ being not admissible. It follows directly from the definition of admissible sets that if a terminal labelling, for a given AF, maps an argument to *must_out* then the set $\{x \mid Lab(x) = in\}$ is not admissible.

Definition 7 (Rejected Labellings)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \rightarrow \{in, out, undec, blank, must\_out\}$ be a total mapping. Then Lab is rejected if and only if Lab is terminal and there is $x \in A$ with $Lab(x) = must\_out$.*

We denote by *preferred labellings* the admissible labellings with $\{x|Lab(x) = in\}$ being inclusion-wise

maximal among all admissible labellings.

Definition 8 (Preferred Labellings)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \rightarrow \{in, out, undec, must\_out, blank\}$ be a total mapping. Then Lab is a preferred labelling of H if and only if Lab is admissible and $\{x \mid Lab(x) = in\}$ is maximal (w.r.t. $\subseteq$) among all admissible labellings.*

Now we recall the pruning strategy used in the labelling algorithm of (Nofal et al., 2016). Note that the pruning strategy of (Nofal et al., 2016) is centered around two notions: *labelling propagation* and *hopeless labellings*, which both improved preferred extension enumeration. Labelling propagation is about inferring argument labels by analyzing the current labelling while hopeless labelling are those labellings that never grow to a preferred labelling.

Definition 9 (Labelling Propagation)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \rightarrow \{in, out, undec, must\_out, blank\}$ be a total mapping. Then propagate(H,Lab) is defined by the following actions:*

1. *while $\exists x\, Lab(x) = blank$ s.t. $\forall y \in \{x\}^- Lab(y) \in \{out, must\_out\}$ do*
   1.1. *$Lab(x) \leftarrow in$*
   1.2. *for each $y \in \{x\}^+$ do $Lab(y) \leftarrow out$*

Definition 10 (Hopeless Labellings)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \rightarrow \{in, out, undec, must\_out, blank\}$ be a total mapping. Then Lab is a hopeless labelling of H if and only if there is $x \in A$ with $Lab(x) = must\_out$ such that for all $y \in \{x\}^- Lab(y) \in \{out, must\_out, undec\}$.*

Now we give algorithm 1 that lists all preferred extensions. If algorithm 1 is invoked on a given AF H, the initial labelling of H and an empty set E, then it will return E containing all preferred extensions. Throughout the paper we assume that a call by reference has to be made to invoke an algorithm or a procedure. Referring to line 4 one can check the maximality of a given labelling *Lab* by ensuring that for each preferred extension $S \in E$ generated so far it is the case that $\{x|Lab(x) = in\} \not\subseteq S$. This is true because algorithm 1 builds admissible sets in a descending order with respect to set inclusion, which means maximal sets are visited first.

In the following section we develop an improved algorithm for listing preferred extensions.

Algorithm 1: **Old** list-preferred-extensions.

input : $H = (A,R)$, $E \subseteq 2^A$,
$Lab : A \rightarrow \{in, out, undec, blank, must\_out\}$.
output: $E \subseteq 2^A$.

```
1  propagate(H,Lab);
2  if Lab is hopeless then return;
3  if Lab is terminal then
4      if Lab is admissible and maximal then
         E ← E ∪ {{x | Lab(x) = in}};
5      return;
6  select an argument x with Lab(x) = blank;
7  Lab' ← in_trans(x,H,Lab);
8  list-preferred-extensions(H, Lab', E);
9  Lab' ← und_trans(x,H,Lab);
10 list-preferred-extensions(H, Lab', E);
```

4 A NEW LABELLING ALGORITHM FOR PREFERRED EXTENSION ENUMERATION

Our new labelling algorithm is enhanced by a new pruning strategy that we present in section 4.2. In section 4.3 we give a new strategy for selecting arguments that induce *in* and *undec* transitions. We introduce a new labelling scheme in section 4.1.

4.1 A New Labelling Scheme

In section 3 we employed a 5-label total function that maps arguments from a given AF to labels from $\{in, out, must\_out, undec, blank\}$. We add to this scheme two more labels: *must_in* and *must_undec*. Hence we use a 7-label total function that maps arguments from a given AF to labels from $\{in$, *out*, *must_out*, *undec*, *blank*, *must_in*, *must_undec*$\}$. From now on we refer to this 7-label set as $\mathbb{L}$.

Before we give a precise description for those arguments that are eligible to be labelled with *must_in* or *must_undec*, we define first two helpful total mappings that both will streamline the computations around deciding such eligibility. Our total functions are called BLANK and UNDEC. BLANK maps an argument to the number of the attackers that are labeled with either *blank*, *must_in*, or *must_undec*. UNDEC maps an argument to the number of *undec* attackers. A similar idea to the essence of BLANK and UNDEC has been used in (Modgil and Caminada, 2009) for computing the grounded extension. We specify BLANK and UNDEC precisely shortly. Observe that we denote the set of nonnegative integers by $\mathbb{N}_0$.

Definition 11 (BLANK and UNDEC). *Let $H = (A,R)$ be an* AF *and Lab $: A \to \mathbb{L}$ be a total mapping, then* BLANK: $A \to \mathbb{N}_0$ *and* UNDEC: $A \to \mathbb{N}_0$ *are total mappings such that for every $x \in A$*

$\text{BLANK}(x) = |\{\, y \in \{x\}^- : Lab(y) \in \{blank, must\_in, must\_undec\}\}|$, *and*
$\text{UNDEC}(x) = |\{\, y \in \{x\}^- : Lab(y) = undec\}|$.

Now we are ready to define the conditions under which an argument becomes eligible to be labelled with *must_undec* or *must_in*.

A *blank* argument, say x, can be labelled with *must_undec* if x has to join (but not yet) the current set of *undec* arguments because otherwise the under-construction set of *in* arguments, say S, together with x (i.e. $\{x\} \cup S$) will never grow to a preferred extension for one of two reasons as specified in the following definition.

Definition 12 (must_undec Arguments). *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping, and x be an argument with $Lab(x) = blank$ then x is eligible to be labelled with must_undec if it holds that*

$$\exists y \in \{x\}^- \text{ with } Lab(y) \in \{blank, undec, must\_undec\} \text{ s.t. BLANK}(y)=0,$$

or it holds that

$$\exists y \in \{x\}^+ \text{ with } Lab(y) = undec \text{ s.t. } \exists z \in \{y\}^+ \text{ with } Lab(z) \in \{undec, must\_undec\} \text{ and BLANK}(z)=0 \text{ and UNDEC}(z)=1.$$

A *blank* argument, say x, can be labelled with *must_in* only if x must join (but not yet) the current set of *in* arguments, say S, because otherwise S will never grow to a preferred extension.

Definition 13 (must_in Arguments). *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping, and x be an argument with $Lab(x) = blank$ then x is eligible to be labelled with must_in if it holds that*

$$\text{BLANK}(x)=0 \text{ and } \text{UNDEC}(x) \in \{0, |\{y : Lab(y) = undec \text{ and } y \in \{x\}^{\pm}\}|\},$$

or it holds that

$$\exists y \in \{x\}^+ \text{ with } Lab(y) = must\_out \text{ such that BLANK}(y)=1.$$

By using the new labels (i.e. *must_in* and *must_undec*) we identify four cases by which an argument's label can be deduced from the current labelling: in two cases an argument must be eventually labelled with *undec* while in the other two cases an argument must end with the *in* label. The essence of these four cases are similar to the ones introduced in (Doutre and Mengin, 2001), but the implementation is totally new as we show throughout the paper.

The question now is why we do not label an argument with *in* and *undec* immediately instead of *must_in* and *must_undec*. Note that labelling an argument with *in* or *undec* may trigger new changes in the current labelling (e.g. labelling propagation) and in turn these changes may produce further ones and so on. Thus, to process these cascading changes more efficiently we use *must_in* and *must_undec*. We explain more precisely the computational benefit of *must_in* and *must_undec* next.

4.2 A New Pruning Strategy

We develop a precise description for our new pruning strategy by defining a number of constructs in this section. We start with two important sets MUST_IN and MUST_UNDEC that respectively hold *must_in* and *must_undec* arguments temporarily until they are eventually labeled with *in* and *undec*. The advantage of using these sets is that we confine computations to those arguments that truly need further processing, and in consequence we avoid scanning all arguments unnecessarily. One might wonder why we utilize two related notions for apparently the same purpose, for example the set MUST_IN and the label *must_in* seem to denote the same arguments. We note that although the two notions refer to the same set of arguments but they allow for different levels of access: by using *must_in* it is computationally easy to check the status of a specific argument at any point of the extension enumeration while the set of MUST_IN enables an efficient access to the collection of those arguments that need to be finalized with *in*. Throughout this section we show exactly the usage of MUST_IN and *must_in* as well as the two related structures MUST_UNDEC nad *must_undec*. We refine now the initial labelling and hopeless labellings, taking into account the new labels *must_in* and *must_undec*.

Definition 14 (New Initial Labelling). *Let $H = (A,R)$ be an* AF, *S be the set of self attacking arguments and T be the set of $\{x : |\{x\}^-| = 0\}$. Then the initial labelling of H is defined by the union of the following sets:*

$$\{(x, blank) \mid x \in A \setminus (S \cup T)\} \cup$$
$$\{(x, must\_undec) \mid x \in S\} \cup$$
$$\{(x, must\_in) \mid x \in T\}.$$

Definition 15 (New Hopeless Labellings). *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping. Then Lab is a hopeless labelling of H if and only if there is x with $Lab(x) = must\_out$ such that* BLANK$(x)=0$.

We introduce the notion of *non-maximal* labellings to describe those labellings with an *undec* (or *must_undec*) argument being attacked by only *out* (or

must_out) arguments. This is because by labelling an argument, say x, with *undec* we mean to find a preferred extension excluding x. But if at some point x becomes acceptable to the current set of *in* arguments, then we better backtrack because the current labelling will never be maximal. Such labellings are hopeless in the sense that they will never grow to a preferred extension although they might be admissible. Nonetheless, we call such labellings non-maximal to distinguish them from the hopeless labellings that are inevitably not admissible.

Definition 16 (Non-maximal Labellings)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping. Then Lab is a non-maximal labelling of H if and only if there exists x with $Lab(x) \in \{undec, must\_undec\}$ such that* BLANK*(x)=0 and* UNDEC*(x)=0.*

Now we describe our new labelling propagation. Basically, labelling propagation might be invoked at any point of the search to identify those arguments that are eligible for *must_in* and *must_undec* labels.

Definition 17 (New Labelling Propagation)**.** *Let $H = (A,R)$ be an* AF*, $Lab : A \to \mathbb{L}$ be a total mapping, $x \in A$ be an argument,* BLANK: $A \to \mathbb{N}_0$ *and* UNDEC: $A \to \mathbb{N}_0$ *be total mappings,* MUST_IN $\subseteq A$ *and* MUST_UNDEC $\subseteq A$ *be sets of arguments then propagate(x, H, Lab,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*) is defined by:*

1. *for each $y \in \{x\}^{\pm}$ do*
1.1. *if y is eligible for must_undec then*
1.1.1. *$Lab(y) \leftarrow$ must_undec.*
1.1.2. MUST_UNDEC $\leftarrow$ MUST_UNDEC $\cup \{y\}$.
1.2. *if y is eligible for must_in then*
1.2.1. *$Lab(y) \leftarrow$ must_in.*
1.2.2. MUST_IN $\leftarrow$ MUST_IN $\cup \{y\}$.
1.3. *for each $z \in \{y\}^{\pm}$ do*
1.3.1. *if z is eligible for must_undec then*
1.3.1.1. *$Lab(z) \leftarrow$ must_undec.*
1.3.1.2. MUST_UNDEC $\leftarrow$ MUST_UNDEC $\cup \{z\}$.
1.3.2. *if z is eligible for must_in then*
1.3.2.1. *$Lab(z) \leftarrow$ must_in.*
1.3.2.2. MUST_IN $\leftarrow$ MUST_IN $\cup \{z\}$.
1.4. *if Lab is hopeless or non-maximal then return false.*
2. *return true.*

Now we expand *in* transitions to include labelling propagation.

Definition 18 (new *in* Transitions)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping, s be an argument with $Lab(s) \in \{blank, must\_in\}$,* BLANK: $A \to \mathbb{N}_0$ *and* UNDEC: $A \to \mathbb{N}_0$ *be total mappings,* MUST_IN $\subseteq A$ *and* MUST_UNDEC $\subseteq A$ *be sets of arguments then in-trans (s,H,Lab,*BLANK*,*UNDEC*,*MUST_UNDEC*,*MUST_IN*) is defined by:*

1. *$Lab(s) \leftarrow$ in.*
2. *for each $x \in \{s\}^{+}$ with $Lab(x) \neq$ out do*
2.1. *for each $y \in \{x\}^{+}$ do*
2.1.1. *if $Lab(x) =$ undec then* UNDEC$(y) \leftarrow$ UNDEC(y)*-1.*
2.1.2. *if $Lab(x) \in \{blank, must\_undec\}$ then* BLANK$(y) \leftarrow$ BLANK(y)*-1.*
2.2. BLANK$(x) \leftarrow$ BLANK(x) *- 1.*
2.3. *$Lab(x) \leftarrow$ out.*
2.4. *if propagate(x,H,Lab,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*)=false then return false.*
3. *for each $x \in \{s\}^{-}$ with $Lab(x) \notin \{out, must\_out\}$ do*
3.1. *$Lab(x) \leftarrow$ must_out.*
3.2. *if propagate(x, H, Lab,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*) = false then return false.*
4. *return true.*

Similarly, we expand *undec* transitions to include labelling propagation.

Definition 19 (New *undec* Transitions)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping, x be an argument with $Lab(x) \in \{blank, must\_undec\}$,* BLANK: $A \to \mathbb{N}_0$ *and* UNDEC: $A \to \mathbb{N}_0$ *be total mappings,* MUST_IN $\subseteq A$ *and* MUST_UNDEC $\subseteq A$ *be sets of arguments then und-trans(x,H,Lab,*BLANK*,*UNDEC*,*MUST_UNDEC*,* MUST_IN*) is defined by:*

1. *$Lab(x) \leftarrow$ undec.*
2. *for each $y \in \{x\}^{+}$ do*
2.1. UNDEC$(y) \leftarrow$ UNDEC(y)*+1.*
2.2. BLANK$(y) \leftarrow$ BLANK(y)*-1.*
3. *return propagate(x, H, Lab,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*).*

Before we make a new branch (by *in* and *undec* transitions) we finalize the label of *must_in* and *must_undec* arguments with *in* and *undec* respectively. Every time we relabel a *must_in* (respectively *must_undec*) argument with *in* (respectively *undec*) we may find that some *blank* arguments have become eligible for either *must_in* or *must_undec*. Thus, a change in some argument's label may cause other changes in the current labeling and so forth. We define this recurrent process by *labelling broadcasting*.

Definition 20 (Labelling Broadcasting)**.** *Let $H = (A,R)$ be an* AF *and $Lab : A \to \mathbb{L}$ be a total mapping,*

BLANK: $A \rightarrow \mathbb{N}_0$ *and* UNDEC: $A \rightarrow \mathbb{N}_0$ *be total mappings,* MUST_IN $\subseteq A$ *and* MUST_UNDEC $\subseteq A$ *be sets of arguments then broadcast(Lab,H,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*) is defined by:*

1. *while* MUST_IN $\neq \emptyset$ *or* MUST_UNDEC $\neq \emptyset$ *do*
1.1. *while* MUST_IN $\neq \emptyset$ *do*
1.1.1. *remove an argument x from* MUST_IN.
1.1.2. *if in-trans(x, H, Lab,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*)=false then return false.*
1.2. *while* MUST_UNDEC $\neq \emptyset$ *do*
1.2.1. *remove an argument x from* MUST_UNDEC.
1.2.2. *if und-trans(x, H, Lab,* BLANK*,* UNDEC*,* MUST_IN*,* MUST_UNDEC*)=false then return false.*
2. *return true.*

Using the new *in* transition, *undec* transition, and labelling broadcasting, we present algorithm 2. Let $H = (A,R)$ be an AF, *Lab* be the initial labelling of H, and for each $x \in A$ let BLANK(x) be equal to $|\{x\}^-|$ while UNDEC(x) be equal to 0, and let MUST_IN be the set $\{x : |\{x\}^-| = 0\}$, MUST_UNDEC be the set $\{x \mid (x,x) \in R\}$, and E be an empty set then if we call algorithm 2 on H, *Lab*, BLANK, UNDEC, MUST_IN, MUST_UNDEC, and E, then the algorithm returns E containing all preferred extensions of H. Figure 3 shows a running of algorithm 2. In the next section we add a further enhancement, which is a new argument selection strategy.

4.3 A New Argument Selection Strategy

Referring to line 6 of algorithm 2, there are many possible selection strategies. One possibility is to pick arguments randomly. Another strategy may depend on some heuristic measures, such as the number of adjacent arguments, which is the strategy used in (Nofal et al., 2016). See (Geilen and Thimm, 2017) for more discussions on heuristic-based selection strategies that depend on the current AF labelling and/or its underlying graph structure properties. Here we introduce a different selection strategy that relies on argument history profile as we explain next.

Our selection strategy is simple. Every time we reach a hopeless labelling because of a *must_out* argument, say x, we mark such x as a failure point and later we give priority to the *blank* attackers of x to be selected for inducing a transition. In other words, our selection strategy prioritizes those arguments that might soon produce a hopeless labelling, instead of delaying the awareness of hopeless labelling possibly until a very late point of the search.

Now we come to the specifications of our selection strategy. We note that a minor modification has to be introduced to algorithm 2 such that every time we detect a hopeless labelling because of a *must_out* argument, say x, the algorithm has to push x on top of a stack structure denoted by $\mathbb{S}$. Algorithm 3 implements our selection strategy. Note that the algorithm either returns a selected argument or it returns -1 to indicate that the current labelling is either hopeless or terminal.

5 VERIFYING THE EFFICIENCY OF THE NEW ALGORITHM

We implemented our new algorithm using the C++ programming language. The source code of the implementation can be found at https://sourceforge.net/projects/argtools. We evaluated the new algorithm using benchmark A of the second international competition of computational models of argumentation 2017 (ICCMA17) (ICC,). Benchmark A includes 350 AFs, for more details see (ICC,). We carried out the evaluation on a system with an intel-core-i7 processor and four gigabytes of system memory. For each problem instance we limit the memory space to one gigabyte. In contrast, ICCMA17 uses a more powerful environment with an intel-xeon processor and with four gigabytes of memory being allocated for each problem instance. However, with respect to the running time we follow ICCMA17, and hence, set a timeout of 10 minutes for each problem instance.

The objective of this evaluation is to verify that the new algorithm enumerates preferred extensions faster than the old algorithm. We found that the new algorithm is able to solve successfully 233 AFs out of benchmark A. Note that the implementation of the old algorithm is included in ArgTools (first version), which is a labelling-based solver that was submitted to the first version of the competition ICCMA15 (Thimm and Villata, 2017). In fact, the old algorithm did not solve any AF of benchmark A. In its second version, submitted to ICCMA17, ArgTools includes some (not all) aspects of the new algorithm. ArgTools (version 2) enumerated all preferred extensions successfully for 157 AFs (ICC,). Another labelling-based solver is Heureka (Geilen and Thimm, 2017), which also participated in ICCMA17. Heureka implements the old algorithm but with a profound heuristic-based argument selection strategy. Heureka enumerated all preferred extensions for 178 AFs of benchmark A.

Algorithm 2: **New** list-preferred-extensions.

input : $H = (A,R)$, $Lab : A \to \mathbb{L}$, BLANK : $A \to \mathbb{N}_0$, UNDEC : $A \to \mathbb{N}_0$, MUST_IN $\subseteq A$, MUST_UNDEC $\subseteq A$, $E \subseteq 2^A$.
output : $E \subseteq 2^A$.

```
1  if broadcast(Lab,H,BLANK,UNDEC,MUST_IN,MUST_UNDEC)=false then return;
2  if Lab is terminal then
3      if Lab is admissible and maximal then
4          E ← E ∪ {{x | Lab(x) = in}} ;
5      return;
6  select an argument x with Lab(x) = blank;
7  Lab' ← Lab, BLANK' ← BLANK, UNDEC' ←UNDEC;
8  MUST_IN' ← MUST_IN, MUST_UNDEC' ← MUST_UNDEC;
9  if in_trans(x,H,Lab',BLANK',UNDEC',MUST_IN',MUST_UNDEC') = true then
10     list-preferred-extensions(H, Lab',BLANK',UNDEC',MUST_IN',MUST_UNDEC', E);
11 if und_trans(x,H,Lab,BLANK,UNDEC,MUST_IN,MUST_UNDEC) = true then
12     list-preferred-extensions(H,Lab,BLANK,UNDEC,MUST_IN,MUST_UNDEC, E);
```

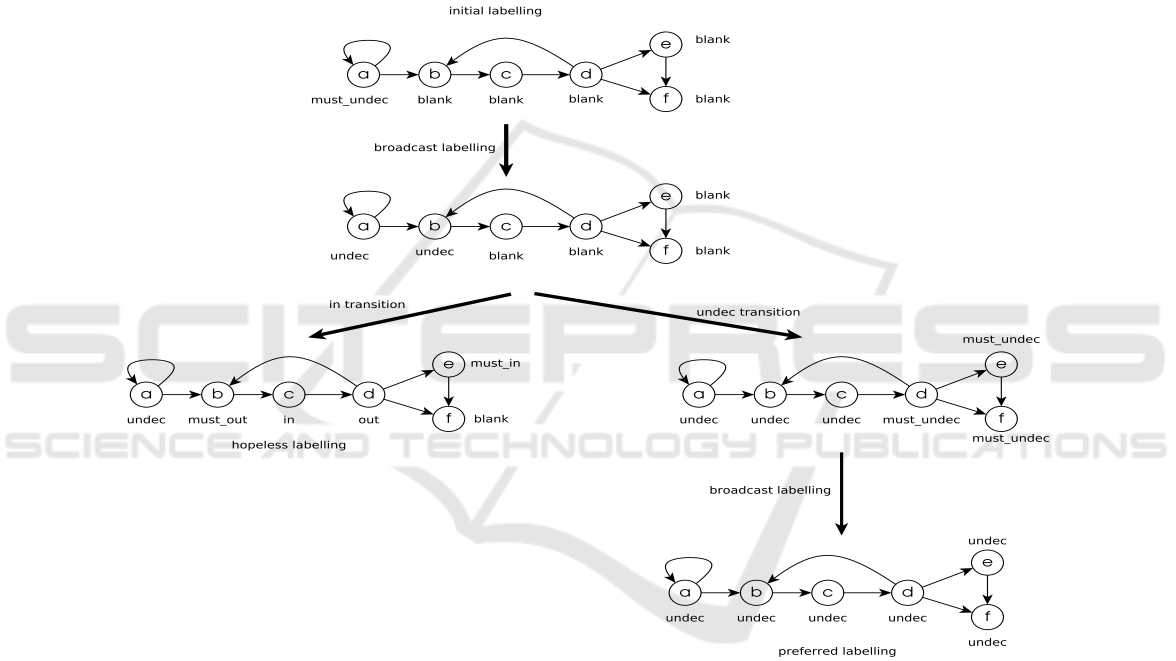

Figure 3: The search tree that is expanded by algorithm 2 in listing the preferred extensions of AF_1 depicted in figure 1.

6 CONCLUSION

We presented a new labelling algorithm that lists all preferred extensions of a given AF. We evaluated the new algorithm and our findings verified that the new algorithm enumerates preferred extensions significantly faster than the old algorithm. The obtained speedup is due to the new pruning strategy of the new algorithm. We plan to study the impact of our new pruning strategy in the context of other computational problems in the field of abstract argumentation.

Lastly we note that the labelling algorithm of (Nofal et al., 2014b) for preferred extension enumeration enhanced the previous algorithm of (Doutre and Mengin, 2001; Caminada, 2007). Nevertheless, the algorithm of (Nofal et al., 2014b) has been improved further in (Nofal et al., 2016) by a look-ahead strategy. In this work we build on the algorithm of (Nofal et al., 2016) as we explained throughout the paper. Another mainstream research concerns building reduction-based solvers, see some examples in (Thimm and Villata, 2017). For computational complexity of abstract argumentation see for example (Dunne and Wooldridge, 2009). For a survey on methods for solving different computational problems of AFs see the article of (Charwat et al., 2015).

Algorithm 3: **New** selecting an argument for transitions.

input : $H = (A,R)$, $Lab : A \rightarrow \mathbb{L}$, $\mathbb{S}$ is a stack of arguments.
output: $x \in A$, $\mathbb{S}$.
while $\mathbb{S} \neq \emptyset$ **do**
  $y \leftarrow$ pop an argument from top of $\mathbb{S}$;
  if $Lab(y) = must\_out$ **then**
    if $\text{BLANK}(y) > 0$ **then** return $x \in \{y\}^-$ with $Lab(x) = blank$ **else** return -1;
foreach $y \in A$ *with* $Lab(y) = must\_out$ **do**
  if $\text{BLANK}(y) > 0$ **then** return some $x \in \{y\}^-$ with $Lab(x) = blank$ **else** return -1;
if $\exists x$ with $Lab(x) = blank$ such that $\text{UNDEC}(x) > 0$ **then** return x;
if $\exists x$ with $Lab(x) = blank$ **then** return x **else** return -1;

REFERENCES

International competition of computational models of argumentation 2017. *http://argumentationcompetition.org/2017*. Organized by: Sarah A. Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran.

Atkinson, K., Baroni, P., Giacomin, M., Hunter, A., Prakken, H., Reed, C., Simari, G. R., Thimm, M., and Villata, S. (2017). Towards artificial argumentation. *AI Magazine*, 38(3):25–36.

Baroni, P., Caminada, M., and Giacomin, M. (2011). An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410.

Bench-Capon, T. J. M., Atkinson, K., and Wyner, A. Z. (2015). Using argumentation to structure e-participation in policy making. *T. Large-Scale Data- and Knowledge-Centered Systems*, 18:1–29.

Caminada, M. (2007). An algorithm for computing semi-stable semantics. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007, Hammamet, Tunisia, October 31 - November 2, 2007, Proceedings*, pages 222–234.

Caminada, M. and Gabbay, D. M. (2009). A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145.

Charwat, G., Dvorák, W., Gaggl, S. A., Wallner, J. P., and Woltran, S. (2015). Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63.

Doutre, S. and Mengin, J. (2001). Preferred extensions of argumentation frameworks: Query answering and computation. In *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, pages 272–288.

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

Dunne, P. E. and Wooldridge, M. (2009). Complexity of abstract argumentation. In *Argumentation in Artificial Intelligence*, pages 85–104.

Geilen, N. and Thimm, M. (2017). Heureka: A general heuristic backtracking solver for abstract argumentation. In *second international competition on computational argumentation models*.

Longo, L. and Dondio, P. (2014). Defeasible reasoning and argument-based systems in medical fields: An informal overview. In *2014 IEEE 27th International Symposium on Computer-Based Medical Systems, New York, NY, USA, May 27-29, 2014*, pages 376–381.

Modgil, S. and Caminada, M. (2009). Proof theories and algorithms for abstract argumentation frameworks. In *Argumentation in Artificial Intelligence*, pages 105–129.

Modgil, S., Toni, F., Bex, F., Bratko, I., Chesñevar, C., Dvořák, W., Falappa, M., Fan, X., Gaggl, S., García, A., González, M., Gordon, T., Leite, J., Možina, M., Reed, C., Simari, G., Szeider, S., Torroni, P., and Woltran, S. (2013). The added value of argumentation. In Ossowski, S., editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 357–403. Springer Netherlands.

Nofal, S., Atkinson, K., and Dunne, P. E. (2014a). Algorithms for argumentation semantics: Labeling attacks as a generalization of labeling arguments. *J. Artif. Intell. Res. (JAIR)*, 49:635–668.

Nofal, S., Atkinson, K., and Dunne, P. E. (2014b). Algorithms for decision problems in argument systems under preferred semantics. *Artif. Intell.*, 207:23–51.

Nofal, S., Atkinson, K., and Dunne, P. E. (2016). Looking-ahead in backtracking algorithms for abstract argumentation. *Int. J. Approx. Reasoning*, 78:265–282.

Tamani, N., Mosse, P., Croitoru, M., Buche, P., Guillard, V., Guillaume, C., and Gontard, N. (2015). An argumentation system for eco-efficient packaging material selection. *Computers and Electronics in Agriculture*, 113(0):174 – 192.

Thimm, M. and Villata, S. (2017). The first international competition on computational models of argumentation: Results and analysis. *Artif. Intell.*, 252:267–294.