# Strong Admissibility, a Tractable Algorithmic Approach

Martin Caminada[1], Sri Harikrishnan[1]

[1]Cardiff University, School of Computer Science & Informatics

**Abstract**

In the current paper, we present two polynomial algorithms for constructing relatively small strongly admissible labellings, with associated min-max numberings, for a particular argument. These labellings can be used as relatively small explanations for the argument's membership of the grounded extension. Although our algorithms are not guaranteed to yield an absolute minimal strongly admissible labelling for the argument (as doing so would have implied an exponential complexity), our best performing algorithm yields results that are only marginally larger. Moreover, the runtime of this algorithm is an order of magnitude smaller than that of the existing approach for computing an absolute minimal strongly admissible labelling for a particular argument. As such, we believe that our algorithms can be of practical value in situations where the aim is to construct a minimal or near-minimal strongly admissible labelling in a time-efficient way.

**Keywords**

strong admissibility, polynomial algorithms

## 1. Introduction

In formal argumentation, one would sometimes like to show that a particular argument is (credulously) accepted according to a particular argumentation semantics, without having to construct the entire extension the argument is contained in. For instance, to show that an argument is in a preferred extension, it is not necessary to construct the entire preferred extension. Instead, it is sufficient to construct a set of arguments that is *admissible*. Similarly, to show that an argument is in the grounded extension, it is not necessary to construct the entire grounded extension. Instead, it is sufficient to construct a set of arguments that is *strongly admissible*.

The concept of strong admissibility was introduced by Baroni and Giacomin [1] as one of the properties to describe and categorise argumentation semantics. It was subsequently studied by Caminada and Dunne [2, 3] who further developed strong admissibility in both its set and labelling form. As a strongly admissible set (labelling) can be used to explain that a particular argument is in the grounded extension (for instance, by using the discussion game of [4]) a relevant question is whether one can identify an explanation that is *minimal*. That is, given an argument $A$ that is in the grounded extension, how can one obtain:
(1) a strongly admissible set that contains $A$, of which the number of arguments is minimal

*SAFA'22: Fourth International Workshop on Systems and Algorithms for Formal Argumentation 2022, September 13, 2022, Cardiff, Wales, United Kingdom*

CaminadaM@cardiff.ac.uk (M. Caminada); HarikrishnanS@cardiff.ac.uk (S. Harikrishnan)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3236/paper3.pdf

among all strongly admissible sets containing $A$, and
(2) a strongly admissible labelling that labels $A$ `in`, of which the number of `in` and `out` labelled arguments (its *size*, cf. [5]) is minimal among all strongly admissible labellings that label $A$ `in`.

It has been found that the verification problem of (1) is NP-complete [6] whereas the verification problem of (2) is co-NP-complete [5]. Moreover, it has also been observed that even computing a $c$-approximation for the minimum size of a strongly admissible set for a given argument is NP-hard for every $c \geq 1$ [6]. This is in sharp contrast with the complexity of the general verification problem of strong admissibility (i.e. verifying whether a set/labelling is strongly admissible, without the constraint that it also has to be minimal) which has been found to be polynomial [3].

The complexity results related to minimal strong admissibility pose a problem when the aim is to provide the user with a relatively small explanation of why a particular argument is in the grounded extension. For this, one can either apply an algorithmic approach that yields an absolute minimal explanation, but has an exponential runtime, or one can apply an algorithmic approach that has a less than exponential runtime, but does not come with any formal guarantees of how close the outcome is to an absolute minimal explanation. The former approach is taken in [6]. The latter approach is taken in our current paper. As the complexity results from [6] prevent us from giving any theory-based guarantees regarding how close the outcome of the algorithm is to an absolute minimal strongly admissible labelling, we will instead assess the performance of the algorithm using a wide range of benchmark examples.

## 2. Preliminaries

In the current section, we briefly restate some of the key concepts of formal argumentation theory, including strong admissibility. For current purposes, we restrict ourselves to finite argumentation frameworks.

**Definition 1.** *An* argumentation framework *is a pair* $(Ar, att)$ *where* $Ar$ *is a finite set of arguments and* $att$ *is a binary relation on* $Ar$*. For any* $x, y \in Ar$ *we say that* $x$ attacks $y$ *iff* $(x, y) \in att$*.*

If $x$ is an argument, we write $x^+$ for the arguments attacked by $x$ and $x^-$ for the arguments that attack $x$. As for notation, we use lower case letters at the end of the alphabet (such as $x$, $y$ and $z$) to denote variables containing arguments, upper case letters at the end of the alphabet (such as $X$, $Y$ and $Z$) to denote program variables containing arguments, and upper case letters at the start of the alphabet (such as $A$, $B$ and $C$) to denote concrete instances of arguments. For current purposes, we apply the labelling-based version of abstract argumentation theory, following [7, 8].

**Definition 2.** *Let* $(Ar, att)$ *be an argumentation framework. An* argument labelling *is a function* $\mathcal{L}ab : Ar \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$*. An argument labelling is called an* admissible labelling *iff for each* $x \in Ar$ *it holds that:*

- *if* $\mathcal{L}ab(x) = \texttt{in}$ *then for each* $y$ *that attacks* $x$ *it holds that* $\mathcal{L}ab(y) = \texttt{out}$
- *if* $\mathcal{L}ab(x) = \texttt{out}$ *then there exists a* $y$ *that attacks* $x$ *such that* $\mathcal{L}ab(y) = \texttt{in}$

*$\mathcal{L}ab$ is called a* complete labelling *iff it is an admissible labelling and for each $x \in Ar$ it also holds that:*

- *if $\mathcal{L}ab(x) = \texttt{undec}$ then there is a $y$ that attacks $x$ such that $\mathcal{L}ab(y) = \texttt{undec}$, and for each $y$ that attacks $x$ such that $\mathcal{L}ab(y) \neq \texttt{undec}$ it holds that $\mathcal{L}ab(y) = \texttt{out}$*

As a labelling is a function, we sometimes write it as a set of pairs. Also, if $\mathcal{L}ab$ is a labelling, we write $\texttt{in}(\mathcal{L}ab)$ for $\{x \in Ar \mid \mathcal{L}ab(x) = \texttt{in}\}$, $\texttt{out}(\mathcal{L}ab)$ for $\{x \in Ar \mid \mathcal{L}ab(x) = \texttt{out}\}$ and $\texttt{undec}(\mathcal{L}ab)$ for $\{x \in Ar \mid \mathcal{L}ab(x) = \texttt{undec}\}$. As a labelling is also a partition of the arguments into sets of in-labelled arguments, out-labelled arguments and undec-labelled arguments, we sometimes write it as a triplet $(\texttt{in}(\mathcal{L}ab), \texttt{out}(\mathcal{L}ab), \texttt{undec}(\mathcal{L}ab))$.

**Definition 3** ([9])**.** *Let $\mathcal{L}ab$ and $\mathcal{L}ab'$ be argument labellings of argumentation framework $(Ar, att)$. We say that $\mathcal{L}ab \sqsubseteq \mathcal{L}ab'$ iff $\texttt{in}(\mathcal{L}ab) \subseteq \texttt{in}(\mathcal{L}ab')$ and $\texttt{out}(\mathcal{L}ab) \subseteq \texttt{out}(\mathcal{L}ab')$.*

**Definition 4.** *Let $\mathcal{L}ab$ be a complete labelling of argumentation framework $(Ar, att)$. $\mathcal{L}ab$ is said to be the grounded labelling iff $\mathcal{L}ab$ is the (unique) smallest (w.r.t. $\sqsubseteq$) complete labelling.*

We refer to the *size* of a labelling $\mathcal{L}ab$ as $|\texttt{in}(\mathcal{L}ab) \cup \texttt{out}(\mathcal{L}ab)|$. We observe that if $\mathcal{L}ab \sqsubseteq \mathcal{L}ab'$ then the size of $\mathcal{L}ab$ is smaller or equal to the size of $\mathcal{L}ab'$, but not necessarily vice versa. In the remainder of the current paper, we use the terms smaller, bigger, minimal and maximal in relation to the size of the respective labellings, unless stated otherwise.

The next step is to define a strongly admissible labelling. In order to do so, we need the concept of a min-max numbering [3].[1]

**Definition 5.** *Let $\mathcal{L}ab$ be an admissible labelling of argumentation framework $(Ar, att)$. A* min-max numbering *is a total function $\mathcal{MM}_{\mathcal{L}ab} : \texttt{in}(\mathcal{L}ab) \cup \texttt{out}(\mathcal{L}ab) \rightarrow \mathbb{N} \cup \{\infty\}$ such that for each $x \in \texttt{in}(\mathcal{L}ab) \cup \texttt{out}(\mathcal{L}ab)$ it holds that:*

- *if $\mathcal{L}ab(x) = \texttt{in}$ then $\mathcal{MM}_{\mathcal{L}ab}(x) = max(\{\mathcal{MM}_{\mathcal{L}ab}(y) \mid y$ attacks $x$ and $\mathcal{L}ab(y) = \texttt{out}\}) + 1$ (with $max(\emptyset)$ defined as $0$)*
- *if $\mathcal{L}ab(x) = \texttt{out}$ then $\mathcal{MM}_{\mathcal{L}ab}(x) = min(\{\mathcal{MM}_{\mathcal{L}ab}(y) \mid y$ attacks $x$ and $\mathcal{L}ab(y) = \texttt{in}\}) + 1$ (with $min(\emptyset)$ defined as $\infty$)*

It has been proved that every admissible labelling has a unique min-max numbering [3]. A strongly admissible labelling can then be defined as follows [3].

**Definition 6.** *A* strongly admissible labelling *is an admissible labelling whose min-max numbering yields natural numbers only (no argument is numbered $\infty$).*

As an example (taken from [3]), consider the argumentation framework of Figure 1. Here, the admissible labelling $\mathcal{L}ab_1 = (\{A, C, F, G\}, \{B, E, H\}, \{D\})$ has min-max numbering $\{(A : 1), (B : 2), (C : 3), (E : 4), (F : 5), (G : \infty), (H : \infty)\}$, which means that it is not strongly admissible. The admissible labelling $\mathcal{L}ab_2 = (\{A, C, D, F\}, \{B, E\}, \{G, H\})$ has min-max numbering $\{(A : 1), (B : 2), (C : 3), (D : 1), (E : 2), (F : 3)\}$, which means that it is strongly admissible.

It has been shown that the strongly admissible labellings form a lattice (w.r.t. $\sqsubseteq$) of which the all-undec labelling is the bottom element and the grounded labelling is the top element [3].

[1] Please notice that for the purpose of Definition 5, $\infty$ is defined such that $\forall x \in \mathbb{N} : \infty + x = \infty$ and $max(\{x, \infty\}) = \infty$. Also, the condition that "$\mathcal{L}ab(y) = \texttt{out}$" in the first point of Definition 5 is technically superfluous, but has been added for clarity.

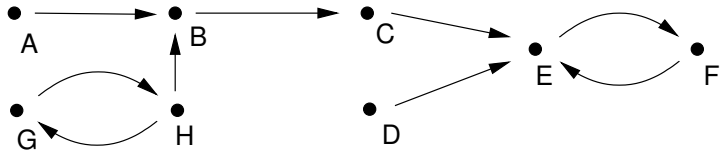


**Figure 1:** An example of an argumentation framework.

## 3. The Algorithms

In the current section, we present an algorithmic approach for computing a relatively small strongly admissible labelling. For this, we provide three different algorithms. The first algorithm (Algorithm 1) basically constructs a strongly admissible labelling bottom-up, starting with the arguments that have no attackers and continuing until the main argument (the argument for which one want to construct the strongly admissible labelling, sometimes also referred to as the argument in question) is labelled `in`. The second algorithm (Algorithm 2) then takes the output of the first algorithm and tries to prune it. That is, it tries to identify only those `in` and `out` labelled arguments that are actually needed for the strongly admissible labelling. The third algorithm (Algorithm 3) then combines Algorithm 1 (which is used as the construction phase) and Algorithm 2 (which is used as the pruning phase). Overall, we assume that it has already been established that the main argument is in the grounded extension and that the aim is merely to find a (relatively small) explanation for this.

### 3.1. Algorithm 1

The basic idea of Algorithm 1 is to start constructing the grounded labelling bottom-up, until we reach the main argument (that is, until we reach the argument that we are trying to construct a strongly admissible labelling for; this argument should hence be labelled `in`). As such, the idea is to take an algorithm for computing the grounded labelling and modify it accordingly. We have chosen the algorithm of [10] for this purpose, as it has been shown to run faster than some of its alternatives (such as [11]). We had to adjust this algorithm in two ways. First, as mentioned above, we want the algorithm to stop once it hits the main argument, instead of continuing to construct the entire grounded labelling. Second, we want it to compute not just the strongly admissible labelling itself, but also its associated min-max numbering.

Obtaining the min-max numbering is important, as it can be used to show that the obtained admissible labelling is indeed *strongly* admissible, through the absence of $\infty$. Additionally, the min-max numbering is also needed for some of the applications of strong admissibility, in particular the Grounded Discussion Game [4] where the combination of a strongly admissible labelling and its associated min-max numbering serves as a roadmap for obtaining a winning strategy.

Instead of first computing the strongly admissible labelling and then proceeding to compute the min-max numbering, the idea is to compute both the strongly admissible labelling and the min-max numbering in just a single pass, in order to achieve the best runtime performance.

To see how the algorithm works, consider again the argumentation framework of Figure 1. Let $C$ be the main argument. At the start of the first iteration of the while loop (line

**Algorithm 1** Construct a strongly admissible labelling that labels $A$ `in` and its associated min-max numbering.

**Input**: An argumentation framework $AF = (Ar, att)$,
an argument $A \in Ar$ that is in the grounded extension of $AF$.
**Output**: A strongly admissible labelling $\mathcal{L}ab$ where $A \in \texttt{in}(\mathcal{L}ab)$,
the associated min-max numbering $\mathcal{MM}_{\mathcal{L}ab}$.

1: // We start with the type definitions
2: $\mathcal{L}ab : Ar \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$
3: $\mathcal{MM}_{\mathcal{L}ab} : \texttt{in}(\mathcal{L}ab) \cup \texttt{out}(\mathcal{L}ab) \rightarrow \mathbb{N} \cup \{\infty\}$
4: $\texttt{undec\_pre} : Ar \rightarrow \mathbb{N}$
5: $\texttt{unproc\_in} : [X_1, ...X_n]$ ($X_i \in Ar$ for each $1 \leq i \leq n$, with $n \geq 0$)
6:
7: // Next, we initialize and process the arguments that have no attackers
8: $\texttt{unproc\_in} \leftarrow []$ // `unproc_in` becomes the empty list of arguments
9: **for** each $X \in Ar$ **do**
10: $\mathcal{L}ab(X) \leftarrow \texttt{undec}$
11: $\texttt{undec\_pre}(X) \leftarrow |X^-|$
12: **if** $\texttt{undec\_pre}(X) = 0$ **then**
13: add $X$ to the rear of `unproc_in`
14: $\mathcal{L}ab(X) \leftarrow \texttt{in}$
15: $\mathcal{MM}_{\mathcal{L}ab}(X) \leftarrow 1$
16: **if** $X = A$ **then** return $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$
17: **end if**
18: **end for**
19:
20: // We proceed to process the arguments that do have attackers
21: **while** `unproc_in` is not empty **do**
22: let $X$ be the argument at the front of `unproc_in`
23: remove $X$ from `unproc_in`
24: **for** each $Y \in X^+$ with $\mathcal{L}ab(Y) \neq \texttt{out}$ **do**
25: $\mathcal{L}ab(Y) \leftarrow \texttt{out}$
26: $\mathcal{MM}_{\mathcal{L}ab}(Y) \leftarrow \mathcal{MM}_{\mathcal{L}ab}(X) + 1$
27: **for** each $Z \in Y^+$ with $\mathcal{L}ab(Z) = \texttt{undec}$ **do**
28: $\texttt{undec\_pre}(Z) \leftarrow \texttt{undec\_pre}(Z) - 1$
29: **if** $\texttt{undec\_pre}(Z) = 0$ **then**
30: add $Z$ to the rear of `unproc_in`
31: $\mathcal{L}ab(Z) \leftarrow \texttt{in}$
32: $\mathcal{MM}_{\mathcal{L}ab}(Z) \leftarrow \mathcal{MM}_{\mathcal{L}ab}(Y) + 1$
33: **if** $Z = A$ **then** return $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$
34: **end if**
35: **end for**
36: **end for**
37: **end while**
38:
39: // If we get here, A is not in the grounded extension,
40: // so we may want to print an error message

21) it holds that $\mathcal{L}ab = (\{A, D\}, \emptyset, \{B, C, E, F, G, H\})$, $\mathcal{MM}_{\mathcal{L}ab} = \{(A : 1), (D : 1)\}$ and `unproc_in` $= [A, D]$. At the first iteration of the while loop, the argument in front of `unproc_in` ($A$) is selected (line 22). This then means that $B$ gets labelled `out` and $C$ gets labelled `in`. Hence, the algorithm hits the main argument ($C$) at line 33 and terminates. This yields a labelling $\mathcal{L}ab = (\{A, C, D\}, \{B\}, \{E, F, G, H\})$ and associated min-max numbering $\mathcal{MM}_{\mathcal{L}ab} = \{(A : 1), (B : 2), (C : 3), (D : 1)\}$.

Algorithm 1 (especially lines 22 and 30) implements a FIFO queue for the `in` labelled arguments it processes. This is an important difference from the algorithm of [10], which uses a set for this purpose. Using a set will work fine if the aim is merely to compute a strongly admissible labelling (as is the case for [10] where the aim is to compute the grounded labelling). However, if the aim is also to compute the associated min-max numbering, having a set as the basic data structure could compromise the algorithm's correctness.

As an example, consider again the argumentation framework of Figure 1. Let $F$ be the main argument. Now suppose that `unproc_in` is a set instead of a queue. In that case, at the start of the first iteration of the while loop (line 21) it holds that $\mathcal{L}ab = (\{A, D\}, \emptyset, \{B, C, E, F, G, H\})$, $\mathcal{MM}_{\mathcal{L}ab} = \{(A : 1), (D : 1)\}$ and `unproc_in` $= \{A, D\}$. At the first iteration of the while loop, an argument $X$ from `unproc_in` is selected (line 22). As a set has no order, it would be possible to select $A$ (so $X = A$). This then means that $B$ gets labelled `out` and $C$ gets labelled `in`. Hence, at the end of the first iteration of the while loop (and therefore at the start of the second iteration of the while loop) it holds that $\mathcal{L}ab = (\{A, C, D\}, \{B\}, \{E, F, G, H\})$, $\mathcal{MM}_{\mathcal{L}ab} = \{(A : 1), (B : 2), (C : 3), (D : 1)\}$ and `unproc_in` $= \{C, D\}$. At the second iteration of the while loop, suppose $C$ is the selected argument (so $X = C$). This means that $E$ gets labelled `out` and $F$ gets labelled `in`. Hence, at the moment the algorithm hits the main argument ($F$, at line 33) and terminates, it holds that $\mathcal{L}ab = (\{A, C, D, F\}, \{B, E\}, \{G, H\})$ and $\mathcal{MM}_{\mathcal{L}ab} = \{(A : 1), (B : 2), (C : 3), (D : 1), (E : 4), (F : 5)\}$. Unfortunately $\mathcal{MM}_{\mathcal{L}ab}$ is incorrect. This is because `out` labelled argument $E$ is numbered 4, whereas its two `in` labelled attackers $C$ and $D$ are numbered 3 and 1, respectively, so the correct min-max number of $E$ should be 2 instead of 4, which implies that the correct min-max number of $F$ should be 3 instead of 5.

One of the conditions of a min-max numbering is that the min-max number of an `out` labelled argument should be the minimal value of its `in` labelled attackers, plus 1. This seems to require that the min-max number of *each* `in` labelled attackers is already known, before being able to assign the min-max number of the `out` labelled argument. At the very least, it would seem that the min-max number of an `out` labelled argument needs to be recomputed each time the min-max number of one of its `in` labelled attackers becomes known. Yet, Algorithm 1 does none of this. It determines the min-max number of an `out` labelled argument as soon as the min-max number of its first `in` labelled attacker becomes known (line 26) without waiting for the min-max number of any other `in` labelled attacker to become available. In spite of this, Algorithm 1 still manages to always yield the correct min-max numbering.

The key to understanding how Algorithm 1 manages to do this is that the `in` labelled arguments are processed in the order of their min-max numbers. That is, once an `in` labelled attacker is identified, any subsequently identified `in` labelled attacker will have a min-max number greater or equal to the first one and will therefore not change the minimal value (in the sense of Definition 5, first bullet point). This avoids having to recalculate the min-max number

of an out labelled attacker when more of its in labelled arguments become available, therefore speeding up the algorithm.

To make sure that arguments are processed in the order of their min-max numbers, we need to apply a FIFO queue instead of the set that was applied by [10]. This FIFO queue is a key component of Algorithm 1, as the correctness of the algorithm critically depends on it. Overall, the correctness of the algorithm can be stated as follows.[2]

**Theorem 1.** *Let $AF = (Ar, att)$ be an argumentation framework and let $A$ be an argument in the grounded extension of $AF$. Let both $AF$ and $A$ be given as input to Algorithm 1. Let $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ be the output of the algorithm. It holds that $\mathcal{L}ab$ is a strongly admissible labelling that labels $A$ in and has $\mathcal{MM}_{\mathcal{L}ab}$ as its min-max numbering.*

It turns out that the algorithm runs in polynomial (cubic) time.

**Theorem 2.** *Let $AF = (Ar, att)$ be an argumentation framework and let $A$ be an argument in the grounded extension of $AF$. Let both $AF$ and $A$ be given as input to Algorithm 1. Let $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ be the output of the algorithm. It holds that Algorithm 1 computes $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ in $O(n^3)$ time*

### 3.2. Algorithm 2

The basic idea of Algorithm 2 is to prune the part of the strongly admissible labelling that is not needed, by identifying the part that actually is needed. This is done in a top-down way, starting by including the main argument (which is labelled in), then including all its attackers (which are labelled out), for each of which a minimally numbered in labelled attacker is included, etc. The idea is to keep doing this until reaching the (in labelled) arguments that have no attackers. Each argument that has not been included by this process is unnecessary for the strongly admissible labelling and can be made undec, resulting in a labelling that is smaller or equal to the strongly admissible labelling one started with.

To see how the algorithm works, consider again the argumentation framework of Figure 1. Let $C$ be the main argument. Suppose the input labelling $\mathcal{L}ab_I$ is $(\{A, C, D\}, \{B\}, \{E, F, G, H\})$ and its associated input labelling numbering $\mathcal{MM}_{\mathcal{L}abO}$ is $\{(A : 1), (B : 2), (C : 3), (D : 1)\}$.[3] At the start of the first iteration of the while loop, it holds that $\mathcal{L}ab_O = (\{C\}, \emptyset, \{A, B, D, E, F, G, H\})$, $\mathcal{MM}_{\mathcal{L}abO} = \{(C : 1)\}$ and unproc_in $= [C]$. The first iteration of the while loop then removes $C$ from unproc_in (line 14), labels its attacker $B$ out (line 16), numbers $B$ with 2 (line 17), adds $A$ to unproc_in (line 20), labels $A$ in (line 21) and numbers $A$ with 1 (line 22). The second iteration of the while loop then removes $A$ from unproc_in (line 14). However, as $A$ does not have any attackers, the for loop (lines 15-24) is skipped. As unproc_in is now empty, the while loop is finished and the algorithm terminates, with $\mathcal{L}ab_O = (\{A, C\}, \{B\}, \{D, E, F, G, H\})$ and $\mathcal{MM}_{\mathcal{L}abO} = \{(A : 1), (B : 2), (C : 3)\}$ being its results.

We now proceed to state some of the formal properties of the algorithm, the first property being correctness.

[2]The formal proof of this result, as well as of some of the other results, had to be omitted due to a lack of space, but can be found in a separate technical report [12].

[3]The reader may have noticed that this was the output of Algorithm 1 for the example that was given in Section 3.1.

**Algorithm 2** Prune a strongly admissible labelling that labels $A$ `in` and its associated min-max numbering.

**Input**: An argumentation framework $AF = (Ar, att)$,
an argument $A \in Ar$ that is in the grounded extension of $AF$, A strongly admissible labelling $\mathcal{L}ab_I$ where $A \in \texttt{in}(\mathcal{L}ab_I)$, the associated min-max numbering $\mathcal{MM}_{\mathcal{L}abI}$.
**Output**: A strongly admissible labelling $\mathcal{L}ab_O \sqsubseteq \mathcal{L}ab_I$ where $A \in \texttt{in}(\mathcal{L}ab_O)$,
the associated min-max numbering $\mathcal{MM}_{\mathcal{L}abO}$.

1: // We start with the type definitions
2: $\mathcal{L}ab_O : Ar \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$
3: $\mathcal{MM}_{\mathcal{L}abO} : \texttt{in}(\mathcal{L}ab) \cup \texttt{out}(\mathcal{L}ab) \rightarrow \mathbb{N} \cup \{\infty\}$
4: `unproc_in` : $[X_1, ...X_n]$ ($X_i \in Ar$ for each $1 \leq i \leq n$) // list of arguments
5: // Initialize $\mathcal{L}ab_O$ and include the main argument
6: $\mathcal{L}ab_O \leftarrow (\emptyset, \emptyset, Ar)$ // $\mathcal{L}ab_O$ becomes the all-`undec` labelling
7: `unproc_in` $\leftarrow [A]$
8: $\mathcal{L}ab_O(A) \leftarrow$ `in`
9: $\mathcal{MM}_{\mathcal{L}abO}(A) \leftarrow \mathcal{MM}_{\mathcal{L}abI}(A)$
10:
11: // Next, process the other arguments in a top-down way
12: **while** `unproc_in` is not empty **do**
13: let $X$ be the argument at the front of `unproc_in`
14: remove $X$ from `unproc_in`
15: **for** each attacker $Y$ of $X$ **do**
16: $\mathcal{L}ab_O(Y) \leftarrow$ `out`
17: $\mathcal{MM}_{\mathcal{L}abO}(Y) \leftarrow \mathcal{MM}_{\mathcal{L}abI}(Y)$
18: **if** there is no minimal (w.r.t $\mathcal{MM}_{\mathcal{L}abI}$) `in` labelled (w.r.t $\mathcal{L}ab_I$) attacker of $Y$ that is also labelled `in` by $Lab_O$ **then**
19: Let $Z$ be a minimal (w.r.t $\mathcal{MM}_{\mathcal{L}abI}$) in labelled (w.r.t $Lab_I$) attacker of $Y$
20: Add Z to the rear of `unproc_in`
21: $\mathcal{L}ab_O(Z) \leftarrow$ `in`
22: $\mathcal{MM}_{\mathcal{L}abO}(Z) \leftarrow \mathcal{MM}_{\mathcal{L}abI}(Z)$
23: **end if**
24: **end for**
25: **end while**

**Theorem 3.** *Let $AF = (Ar, att)$ be an argumentation framework, $A$ be an argument in the grounded extension of $AF$, $\mathcal{L}ab_I$ be a strongly admissible labelling where $A$ is labelled* `in` *and $\mathcal{MM}_{\mathcal{L}abI}$ be the associated min-max numbering. Let $AF$, $A$, $\mathcal{L}ab_I$ and $\mathcal{MM}_{\mathcal{L}abI}$ be given as input to Algorithm 2. Let $\mathcal{L}ab_O$ and $\mathcal{MM}_{\mathcal{L}abO}$ be the output of Algorithm 2. It holds that $\mathcal{L}ab_O$ is a strongly admissible labelling that labels $A$* `in` *and has $\mathcal{MM}_{\mathcal{L}abO}$ as its min-max numbering.*

**Theorem 4.** *Let $AF = (Ar, att)$ be an argumentation framework, $A$ be an argument in the grounded extension of $AF$, $\mathcal{L}ab_I$ be a strongly admissible labelling where $A$ is labelled* `in` *and*

*$\mathcal{MM}_{\mathcal{L}ab_I}$ be the associated min-max numbering. Let $AF$, $A$, $\mathcal{L}ab_I$ and $\mathcal{MM}_{\mathcal{L}ab_I}$ be given as input to Algorithm 2. Let $\mathcal{L}ab_O$ and $\mathcal{MM}_{\mathcal{L}abO}$ be the output of Algorithm 2. It holds that $\mathcal{L}ab_O \sqsubseteq \mathcal{L}ab_I$*

It turns out that the algorithm runs in polynomial (cubic) time.

**Theorem 5.** *Let $AF = (Ar, att)$ be an argumentation framework, $A$ be an argument in the grounded extension of $AF$, $\mathcal{L}ab_I$ be a strongly admissible labelling where $A$ is labelled in and $\mathcal{MM}_{\mathcal{L}ab_I}$ be the correct min-max numbering of $\mathcal{L}ab_I$. Let $AF$, $A$, $\mathcal{L}ab_I$ and $\mathcal{MM}_{\mathcal{L}ab_I}$ be given as input to Algorithm 2. Let $\mathcal{L}ab_O$ and $\mathcal{MM}_{\mathcal{L}abO}$ be the output of Algorithm 2. It holds that Algorithm 2 computes $\mathcal{L}ab_O$ and $\mathcal{MM}_{\mathcal{L}abO}$ in $O(n)^3$ time.*

### 3.3. Algorithm 3

The idea of Algorithm 3 is to combine Algorithm 1 and Algorithm 2, by running them in sequence. That is, the output of Algorithm 1 is used as input for Algorithm 2.

**Algorithm 3** Construct a relatively small strongly admissible labelling that labels $A$ `in` and its associated min-max numbering.

**Input**: An argumentation framework $AF = (Ar, att)$,
an argument $A \in Ar$ that is in the grounded extension of $AF$.
**Output**: A strongly admissible labelling $\mathcal{L}ab$ where $A \in \texttt{in}(\mathcal{L}ab)$, the associated min-max numbering $\mathcal{MM}_{\mathcal{L}ab}$.

1: run Algorithm 1
2: $\mathcal{L}ab_I \leftarrow \mathcal{L}ab$
3: $\mathcal{MM}_{\mathcal{L}ab_I} \leftarrow \mathcal{MM}_{\mathcal{L}ab}$
4: run Algorithm 2
5: $\mathcal{L}ab \leftarrow \mathcal{L}ab_O$
6: $\mathcal{MM}_{\mathcal{L}ab} \leftarrow \mathcal{MM}_{\mathcal{L}abO}$

As an example, consider again the argumentation framework of Figure 1. Let $C$ be the main argument. Running Algorithm 1 yields a labelling $(\{A, C, D\}, \{B\}, \{E, F, G, H\})$ with associated numbering $\{(A : 1), (B : 2), (C : 3), (D : 1)\}$ (as explained in Section 3.1). Feeding this labelling and numbering into Algorithm 2 then yields an output labelling $(\{A, C\}, \{B\}, \{D, E, F, G, H\})$ with associated output numbering $\{(A : 1), (B : 2), (C : 3)\}$ (as explained in Section 3.2).

Given the properties of Algorithm 1 and Algorithm 2, we can prove that Algorithm 3 correctly computes a strongly admissible labelling and its associated min-max numbering, yields an output that is smaller or equal to the output of Algorithm 1, and runs in polynomial (cubic) time.

**Theorem 6.** *Let $AF = (Ar, att)$ be an argumentation framework and let $A$ be an argument in the grounded extension of $AF$. Let both $AF$ and $A$ be given as input to Algorithm 3. Let $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ be the output of the algorithm. It holds that $\mathcal{L}ab$ is a strongly admissible labelling that labels $A$ `in` and has $\mathcal{MM}_{\mathcal{L}ab}$ as its min-max numbering.*

*Proof.* This follows from Theorem 1 and Theorem 3. □

**Theorem 7.** *Let $AF = (Ar, att)$ be an argumentation framework, $A$ be an argument in the grounded extension of $AF$. Let $AF$ and $A$ be given as input to Algorithm 1 and Algorithm 3. Let $\mathcal{L}ab_I$ and $\mathcal{MM}_{\mathcal{L}abI}$ be the output of Algorithm 1 and let $\mathcal{L}ab_3$ and $\mathcal{MM}_{\mathcal{L}ab3}$ be the output of Algorithm 3. It holds that $\mathcal{L}ab_3 \sqsubseteq \mathcal{L}ab_I$*

*Proof.* This follows from Theorem 4, together with Theorem 1 and the way Algorithm 3 is defined (by successively applying Algorithm 1 and Algorithm 2) □

**Theorem 8.** *Let $AF = (Ar, att)$ be an argumentation framework and let $A$ be an argument in the grounded extension of $AF$. Let both $AF$ and $A$ be given as input to Algorithm 3. Let $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ be the output of the algorithm. It holds that Algorithm 3 computes $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ in $O(n^3)$ time.*

*Proof.* This follows from Theorem 2 and Theorem 5. □

## 4. Empirical Results

Now that the formal properties of our algorithms have been stated, the next step is to empirically evaluate their performance. For this, we looked at the benchmark examples that have applied in ICCMA'17 and ICCMA'19,[4] of which we used the 277 example argumentation frameworks whose grounded extension is not empty. We implemented our algorithms in C++ and ran the performance tests on a MacBook Pro 2020 with 8GB of memory and an Intel Core i5 processor.

Compared to the grounded labelling, we found that Algorithm 1 yields an output that is smaller than the grounded labelling in 63% of the examples, with the average output size being 76% of the size of the grounded labelling. Algorithm 3 yields an output that is smaller than the grounded labelling in 88% of the examples, with the average output size being just 25% of the size of the grounded labelling. These findings are relevant as the only previously available polynomial algorithms for strong admissibility are those for computing the grounded extension/labelling (e.g. [10]).

Next, we compared the output of our best performing algorithm (Algorithm 3) for finding a small strongly admissible labelling with the size of an absolute minimal strongly admissible labelling for the argument in question, for which we used the ASPARTIX encodings of [6]. We found that in 91% of the examples, the output of Algorithm 3 is of the same size as an absolute minimal strongly admissible labelling for the argument in question, with the average output being just 3% bigger than the size of an absolute smallest strongly admissible labelling for the argument in question.

As for runtime, we first compared the runtime of our Algorithm 3 with the runtime of Algorithm 3 of [10] for computing the grounded labelling.[5] We found that the differences in runtime are minimal, with the former algorithm taking just 0.02 seconds (3%) longer than the

[4] See http://argumentationcompetition.org/

[5] We modified the latter algorithm to return the grounded labelling instead of the grounded extension. We also made minor corrections by removing some duplicate code.

latter algorithm.[6] Next, we compared the runtime of our Algorithm 3 with the runtime of the ASPARTIX encodings of [6] for computing an absolute minimal strongly admissible labelling.[7] We found that the latter approach has a runtime that is 12.5 seconds (907%) longer than the former approach. That is, our Algorithm 3 is able to provide an answer in less than a tenth of the time it would have taken to compute an absolute minimal answer. A more detailed analysis of our results, including a pointer to the source code of our software, can be found in [12].

## 5. Discussion

In the current paper, we provided two polynomial algorithms (Algorithm 1 and Algorithm 3) for constructing a relatively small strongly admissible labelling and its associated min-max numbering, for a particular argument. The correctness and complexity of the algorithms has been stated in a formal way, with proofs available in [12].

The fact that the $c$-approximation problem for minimal strong admissibility is NP-complete for every $c \geq 1$ means that polynomial algorithms (such as ours) cannot give any formal guarantees of how close their output is to having the size of an absolute minimal strongly admissible labelling for the argument in question. As such, we had to rely on an empirical evaluation of this (using the benchmark examples of ICCMA'17 and ICCMA'19). Overall, our results indicate that the output of Algorithm 3 is only marginally larger (3%) than an absolute minimal strongly admissible labelling for the argument in question, while taking less than a tenth of the runtime that would be needed to compute such a minimal strongly admissible labelling.

As a strongly admissible labelling and its associated min-max numbering can serve as an explanation of membership of the grounded extension, our algorithms can be useful for obtaining, in a time-efficient way, an explanation that is not overly long. Such an explanation can then for instance be used as the basis for the argument-based discussion game of [4], where the combination of the strongly admissible labelling and its associated min-max numbering serves as a roadmap for selecting the moves to be played by the computer when reacting to the user's possible counterarguments [4].

In the current paper, we focussed on strong admissibility, but a similar development of algorithms and associated empirical analysis could be done as future research in the context of admissible labellings and ideal labellings, in order to obtain relatively small explanations for (credulous) preferred semantics and ideal semantics, which could be used in the respective discussion games for these semantics [13].

## References

[1] P. Baroni, M. Giacomin, On principle-based evaluation of extension-based argumentation semantics, Artificial Intelligence 171 (2007) 675–700.

[6] Please be aware, however, that Algorithm 3 of [10] does *not* yield the min-max numbering, whereas Algorithm 3 does.

[7] Please notice that the latter is currently the only implementation for minimal strong admissibility.

[2] M. Caminada, Strong admissibility revisited, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Computational Models of Argument; Proceedings of COMMA 2014, IOS Press, 2014, pp. 197–208.
[3] M. Caminada, P. Dunne, Strong admissibility revised: theory and applications, Argument & Computation 10 (2019) 277–300.
[4] M. Caminada, A discussion game for grounded semantics, in: E. Black, S. Modgil, N. Oren (Eds.), Theory and Applications of Formal Argumentation (proceedings TAFA 2015), Springer, 2015, pp. 59–73.
[5] M. Caminada, P. Dunne, Minimal strong admissibility: a complexity analysis, in: H. Prakken, S. Bistarelli, F. Santini, C. Taticchi (Eds.), Proceedings of COMMA 2020, IOS Press, 2020, pp. 135–146.
[6] W. Dvořák, J. Wallner, Computing strongly admissible sets, in: H. Prakken, S. Bistarelli, F. Santini, C. Taticchi (Eds.), Proceedings of COMMA 2020, IOS Press, 2020, pp. 179–190.
[7] M. Caminada, On the issue of reinstatement in argumentation, in: M. Fischer, W. van der Hoek, B. Konev, A. Lisitsa (Eds.), Logics in Artificial Intelligence; 10th European Conference, JELIA 2006, Springer, 2006, pp. 111–123. LNAI 4160.
[8] M. Caminada, D. Gabbay, A logical account of formal argumentation, Studia Logica 93 (2009) 109–145. Special issue: new ideas in argumentation theory.
[9] M. Caminada, G. Pigozzi, On judgment aggregation in abstract argumentation, Autonomous Agents and Multi-Agent Systems 22 (2011) 64–102.
[10] S. Nofal, K. Atkinson, P. Dunne, Computing grounded extensions of abstract argumentation frameworks, The Computer Journal 64 (2021) 54–63.
[11] S. Modgil, M. Caminada, Proof theories and algorithms for abstract argumentation frameworks, in: I. Rahwan, G. Simari (Eds.), Argumentation in Artificial Intelligence, Springer, 2009, pp. 105–129.
[12] M. Caminada, S. Harikrishnan, Strong Admissibility, a Tractable Algorithmic Approach (proofs), Technical Report, Cardiff University, 2022. Https://arxiv.org/abs/2204.03551.
[13] M. Caminada, Argumentation semantics as formal discussion, in: Handbook of Formal Argumentation, volume 1, College Publications, 2018, pp. 487–518.