An Evaluation of Algorithms for Strong Admissibility

Martin Caminada[1], Sri Harikrishnan[2]

[1]Cardiff University, Cardiff, United Kingdom
[2]Wirtschaftsuniversität Wien, Vienna, Austria

Abstract

In the current paper, we evaluate the performance of different computational approaches for constructing a strongly admissible labelling for a particular argument. Unlike previous work, which examined different approaches for constructing a *small* strongly admissible labelling for a particular argument, in the current paper we are interested in constructing an *arbitrary* strongly admissible labelling for a particular argument, without any constraints regarding the size of such a labelling. A strongly admissible labelling relies on its associated min-max numbering to show that it is actually strongly admissible. As such, we also examine the additional computational costs of constructing such a min-max numbering. Overall, our analysis leads to a clear recommendation regarding which of the current computational approaches is best fit for purpose.

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. In particular, the strongly admissible sets (resp. labellings) were found to form a lattice with the empty set (resp. the all-`undec` labelling) as its bottom element and the grounded extension (resp. the grounded labelling) as its top element [2, 3].

A strongly admissible set (or 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, therefore, is how to construct a strongly admissible set (or labelling) for a particular argument in a computationally efficient way (that is, with minimal runtime).

SAFA'24: Fifth International Workshop on Systems and Algorithms for Formal Argumentation 2022, September 17, 2024, Hagen, Germany

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

In previous work [5] we focussed on providing algorithms that aim to construct a relatively small[1] strongly admissible labelling for a particular argument (that is, where this argument is labelled `in`). These algorithms are approximations in the sense that although they aim to yield a relatively small strongly admissible labelling, they do not guarantee to yield an absolute smallest strongly admissible labelling,[2] as the task of doing the latter is NP-complete [7], whereas our algorithms are polynomial [5]. Overall, we found that our best performing algorithm (Algorithm 3 of [5]) yields labellings that are only marginally bigger than an absolute smallest strongly admissible labelling, with just a fraction of the runtime [5].

In the current paper, we examine two additional questions that were not examined in [5]. The first question is how to efficiently construct a strongly admissible labelling for an argument, without any requirement regarding the size of such a labelling. The second question is what is the additional cost of constructing the min-max numbering of the strongly admissible labelling, instead of just constructing the labelling itself.

The current paper is structured as follows. First, in Section 2 we briefly reiterate some of the theory regarding strongly admissible sets and strongly admissible labellings. In order to make the paper self-contained, we also include the algorithms stated in [5]. Then, in Section 3, we examine which of the existing algorithms for strong admissibility (ours as well as those of others) are the most efficient when it comes to constructing an arbitrary strongly admissible labelling for a particular argument, regardless the size of such labelling. Then, in Section 4, we examine the additional costs (regarding runtime) of computing the min-max numbering of the strongly admissible labelling, instead of just constructing the labelling itself. We round off with a discussion of the obtained results in Section 5.

2. Preliminaries

In the current section, we briefly restate some of the basic concepts in 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 entities, called arguments, whose internal structure can be left unspecified, and* att *is a binary relation on* Ar*. For any* $x, y \in Ar$ *we say that* x attacks y *iff* $(x, y) \in att$*.*

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.

When it comes to defining argumentation semantics, one can distinguish the *extension approach* and the *labelling approach* [8]. We start with the extensions approach.

[1] Small w.r.t. the *size* of the labelling [6], meaning that the number of `in` or `out` labelled arguments is minimal. The advantage of a small labelling is that this leads to an explanation that can be given in a small number of steps [4].

[2] Please notice that we write "an absolute smallest strongly admissible labelling" instead of "the absolute smallest strongly admissible labelling" as there might be more than one strongly admissible labelling with minimal size for a particular argument

Definition 2. *Let (Ar, att) be an argumentation framework, $x \in Ar$ and $Args \subseteq Ar$. We define x^+ as $\{y \in Ar \mid x$ attacks $y\}$, x^- as $\{y \in Ar \mid y$ attacks $x\}$, $Args^+$ as $\bigcup\{x^+ \mid x \in Args\}$, and $Args^-$ as $\bigcup\{x^- \mid x \in Args\}$. $Args$ is said to be* conflict-free *iff $Args \cap Args^+ = \emptyset$. $Args$ is said to* defend *x iff $x^- \subseteq Args^+$. The characteristic function $F : 2^{Ar} \rightarrow 2^{Ar}$ is defined as $F(Args) = \{x \mid Args$ defends $x\}$.*

Definition 3. *Let (Ar, att) be an argumentation framework. $Args \subseteq Ar$ is*

- *an admissible set iff $Args$ is conflict-free and $Args \subseteq F(Args)$*
- *a complete extension iff $Args$ is conflict-free and $Args = F(Args)$*
- *a grounded extension iff $Args$ is the smallest (w.r.t. $\subseteq$) complete extension*

As mentioned in the introduction, the concept of strong admissibility was originally introduced by Baroni and Giacomin [1]. For current purposes we will apply the equivalent definition of Caminada [2, 3].

Definition 4. *Let (Ar, att) be an argumentation framework. $Args \subseteq Ar$ is* strongly admissible *iff every $x \in Args$ is defended by some $Args' \subseteq Args \setminus \{x\}$ which in its turn is again strongly admissible.*

It can be shown that each strongly admissible set is conflict-free and admissible [3]. The strongly admissible sets form a lattice (w.r.t. $\subseteq$), of which the empty set is the bottom element and the grounded extension is the top element [3].

The above definitions essentially follow the extension based approach as described in [9]. It is also possible to define the key argumentation concepts in terms of argument labellings [10, 11].

Definition 5. *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 essentially 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 6 ([12])**.** *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 7. *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].

Definition 8. *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 ∞)*

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 9. *A* strongly admissible labelling *is an admissible labelling whose min-max numbering yields natural numbers only (so no argument is numbered ∞).*

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].

The relationship between extensions and labellings has been well-studied [10, 11]. A common way to relate extensions to labellings is through the functions `Args2Lab` and `Lab2Args`. These translate a conflict-free set of arguments to an argument labelling, and an argument labelling to a set of arguments, respectively. More specifically, given an argumentation framework (Ar, att), and an associated conflict-free set of arguments $Args$ and a labelling $\mathcal{L}ab$, $\texttt{Args2Lab}(Args)$ is defined as $(Args, Args^+, Ar \backslash (Args \cup Args^+))$ and $\texttt{Lab2Args}(\mathcal{L}ab)$ is defined as $\texttt{in}(\mathcal{L}ab)$. It has been proven [11] that if $Args$ is an admissible set (resp. a complete or grounded extension) then $\texttt{Args2Lab}(Args)$ is an admissible labelling (resp. a complete or grounded labelling), and that if $\mathcal{L}ab$ is an admissible labelling (resp. a complete or grounded labelling) then $\texttt{Lab2Args}(\mathcal{L}ab)$ is an admissible set (resp. a complete or grounded extension). It has also been proven [3] that if $Args$ is a strongly admissible set then $\texttt{Args2Lab}(Args)$ is a strongly admissible labelling, and that if $\mathcal{L}ab$ is a strongly admissible labelling then $\texttt{Lab2Args}(\mathcal{L}ab)$ is a strongly admissible set.

Now that the formal preliminaries have been stated, the next step is to include the algorithms from [5], as these will form the basis of our analysis.[3] 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 wants to show membership of a strongly admissible set) is labelled `in`. The second algorithm (Algorithm

[3] Another reason for explicitly including these algorithms is that doing so allows us to describe slightly different versions of these, such as Algorithm 1′ (Section 2) and Algorithm 1″ (section 4).

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 $\mathcal{MM}_{\mathcal{L}ab}$.

$\mathcal{L}ab : Ar \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ // We start with the type definitions
$\mathcal{MM}_{\mathcal{L}ab} : \texttt{in}(\mathcal{L}ab) \cup \texttt{out}(\mathcal{L}ab) \rightarrow \mathbb{N} \cup \{\infty\}$
$\texttt{undec\_pre} : Ar \rightarrow \mathbb{N}$
$\texttt{unproc\_in} : [X_1, ...X_n]$ ($X_i \in Ar$ for each $1 \leq i \leq n$) // list of arguments
$\texttt{unproc\_in} \leftarrow []$ // We initialize and process the arguments that have no attackers
for each $X \in Ar$ **do**
    $\mathcal{L}ab(X) \leftarrow \texttt{undec}$
    $\texttt{undec\_pre}(X) \leftarrow |X^-|$
    if $\texttt{undec\_pre}(X) = 0$ **then**
        add X to the rear of `unproc_in`
        $\mathcal{L}ab(X) \leftarrow \texttt{in}$
        $\mathcal{MM}_{\mathcal{L}ab}(X) \leftarrow 1$
        if $X = A$ **then** return $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$
    end if
end for
while `unproc_in` is not empty **do** // We now process the arguments that do have attackers
    let X be the argument at the front of `unproc_in`
    remove X from `unproc_in`
    for each $Y \in X^+$ with $\mathcal{L}ab(Y) \neq \texttt{out}$ **do**
        $\mathcal{L}ab(Y) \leftarrow \texttt{out}$
        $\mathcal{MM}_{\mathcal{L}ab}(Y) \leftarrow \mathcal{MM}_{\mathcal{L}ab}(X) + 1$
        for each $Z \in Y^+$ with $\mathcal{L}ab(Z) = \texttt{undec}$ **do**
            $\texttt{undec\_pre}(Z) \leftarrow \texttt{undec\_pre}(Z) - 1$
            if $\texttt{undec\_pre}(Z) = 0$ **then**
                add Z to the rear of `unproc_in`
                $\mathcal{L}ab(Z) \leftarrow \texttt{in}$
                $\mathcal{MM}_{\mathcal{L}ab}(Z) \leftarrow \mathcal{MM}_{\mathcal{L}ab}(Y) + 1$
                if $Z = A$ **then** return $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$
            end if
        end for
    end for
end while

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 in 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

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: $\mathcal{MM}_{\mathcal{L}abO} : \texttt{in}(\mathcal{L}ab_O) \cup \texttt{out}(\mathcal{L}ab_O) \rightarrow \mathbb{N} \cup \{\infty\}$
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: $\mathcal{L}ab_O(A) \leftarrow \texttt{in}$
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: $\mathcal{L}ab_O(Y) \leftarrow \texttt{out}$
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: $\mathcal{L}ab_O(Z) \leftarrow \texttt{in}$
22: $\mathcal{MM}_{\mathcal{L}abO}(Z) \leftarrow \mathcal{MM}_{\mathcal{L}abI}(Z)$
23: **end if**
24: **end for**
25: **end while**

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) strongly admissible labelling that shows this.

It has been proven that each of the above three algorithms terminates (in polynomial time) and provides a strongly admissible labelling that labels the argument `in` (provided that such a strongly admissible labelling exists, which requires the argument to be in the grounded extension) together with the min-max numbering of this labelling [5]. Also, in Algorithm 2 it holds that $\mathcal{L}ab_O \sqsubseteq \mathcal{L}ab_I$, which implies that its output is smaller or equal to its input [5]. We refer to [5] for a more detailed discussion of algorithms 1, 2 and 3.

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: $\mathcal{MM}_{\mathcal{L}abI} \leftarrow \mathcal{MM}_{\mathcal{L}ab}$
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 aside, although Algorithm 1 has been written with the aim of constructing a strongly admissible labelling (and associated min-max numbering) for a particular argument, it can easily be altered to yield the grounded labelling (and associated min-max numbering) instead. This could be done by commenting out lines 13 and 28, and by adding the following line (line 33) at the end of the algorithm:
return $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$
We refer to the thus modified algorithm as Algorithm 1′. It has been proven that Algorithm 1′ indeed yields the grounded labelling and its associated min-max numbering [5].[4]

3. Computing an Arbitrary Strongly Admissible Labelling

In our previous work [5] we focused our attention on the problem of finding a *small* strongly admissible labelling (for a particular argument) in a time efficient way. In the current section, we proceed to examine the question of how to find an *arbitrary* strongly admissible labelling (for a particular argument) in a time efficient way. That is, we are interested in finding a fast way of constructing a strongly admissible labelling (that labels the argument in question `in`) without regards of how big or small this labelling is.[5]

In order to maintain consistency and to allow for a like-for-like comparison, we examine this question using the same set of benchmark examples as was used in [5]. That is, we use the benchmark examples of ICCMA'17 and ICCMA'19[6] that have a non-empty grounded extension. There are 277 such benchmark examples in total. For each of these benchmark examples, we generate a query argument that is in the grounded extension. When selecting the query argument, we followed the suggestion of ICCMA'17 and ICCMA'19 whenever such a suggestion was available (for instance, when considering the benchmark examples of the *Admbuster* class, we took 'a' to be the queried argument, as this was suggested by the authors of this class).

[4] In [5] Algorithm 1 was referred to as "the algorithm of Lemma 9".
[5] It can be mentioned that ICCMA'17 and ICCMA'19 describe the somewhat similar task of finding an arbitrary extension for a particular semantics, without any further constraints of what this extension should look like.
[6] See http://argumentationcompetition.org/

Similar to what was done in [5], we conducted our experiments on a MacBook Pro 2020 with 8GB of memory and an Intel Core i5 processor. To run the ASPARTIX system we used clingo 5.6.2. We set a timeout limit of 1000 seconds and a memory limit of 8GB per query.

In [5] it was found that the output of Algorithm 3 is on average 32% smaller than the output of Algorithm 1. As for runtime, however, Algorithm 3 takes on average 103.51% more time (more than double) than Algorithm 1. Figure 1 depicts a detailed comparison regarding the runtime of Algorithm 1 versus the runtime of Algorithm 3. In this figure, as well as in similar figures further on in the paper, each dot represents one of the 277 benchmark examples we tested for. Each dot below the dashed line represents an example where the algorithm on the vertical axis (in this case: Algorithm 1) ran faster than the algorithm on the horizontal axis. (in this case: Algorithm 3)

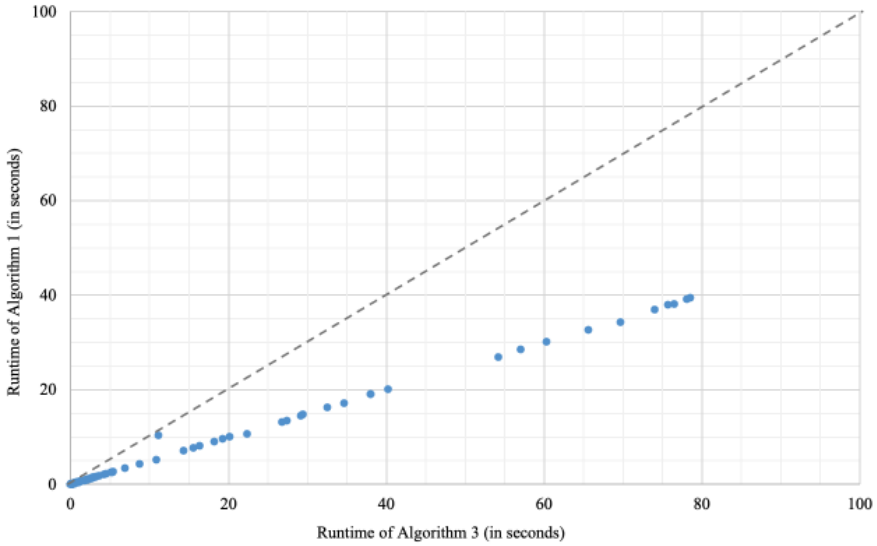

Figure 1: The runtime of Algorithm 1 compared with the runtime of Algorithm 3.

We are not aware of any dedicated third party algorithms or other computational approaches that aim to construct an arbitrary strongly admissible labelling in the shortest possible time. We are, however, aware of computational approaches that aim to either construct a minimal strongly admissible labelling (the ASPARTIX approach of [7]) or a maximal strongly admissible labelling (that is, to construct the grounded labelling). As for the former, Figure 2 depicts a detailed comparison regarding the runtime of Algorithm 1 versus the runtime of the ASPARTIX approach of [7] for constructing a minimal strongly admissible labelling. On average, the ASPARTIX approach of [7] for constructing a smallest strongly admissible labelling takes 540.78% more time (more than 6 times as much) than Algorithm 1 on the 277 benchmark examples that we tested for.[7]

[7]This is despite the fact that the ASPARTIX approach of [7] only constructs a (minimal) strongly admissible labelling,

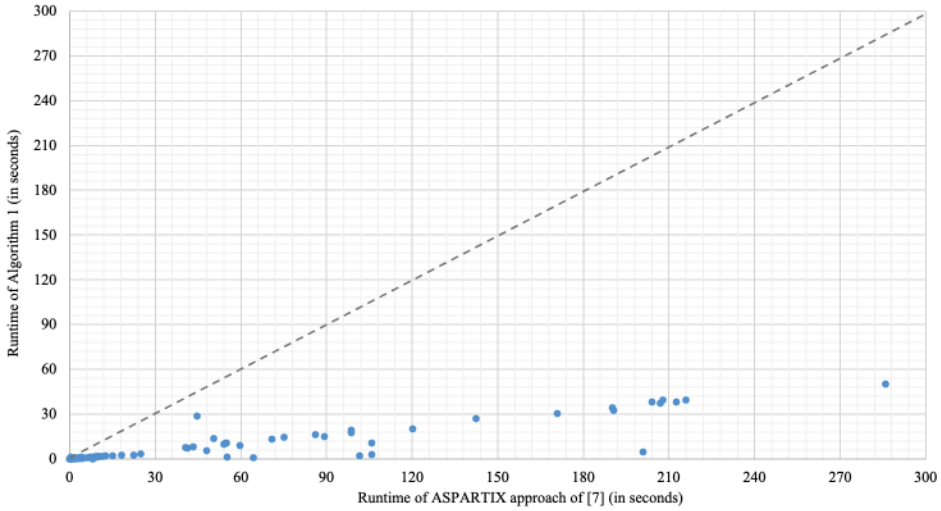

Figure 2: The runtime of Algorithm 1 compared with the runtime of the ASPARTIX approach of [7].

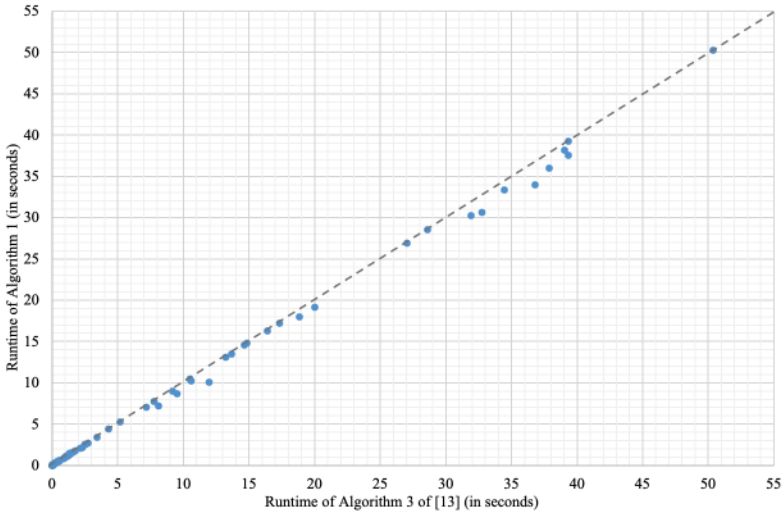

Figure 3: The runtime of Algorithm 1 compared with the runtime of Algorithm 3 of [13].

As for comparing the performance of Algorithm 1 with an algorithm for constructing the grounded labelling, we have chosen Algorithm 3 of [13] as a benchmark, as it is an imperative program (just like Algorithm 1) allowing for a like-for-like comparison. Figure 3 depicts a

whereas Algorithm 1 constructs both a strongly admissible labelling and its associated min-max numbering.

detailed comparison regarding the runtime of Algorithm 1 versus the runtime of Algorithm 3 of [13]. On average, Algorithm 3 of [13] takes 2.99% more time than Algorithm 1 on the 277 benchmark examples that we tested for.[8]

4. Computing a Min-Max Numbering

Algorithms 1, 2 and 3 construct not only a strongly admissible labelling but also its associated min-max numbering. The relevance of min-max numberings goes beyond merely determining (by the absence of ∞) whether its associated admissible labelling is strongly admissible or not. A min-max numbering also serves as the basis for playing the Grounded Discussion Game [4], which can be used for explaining why a particular argument is in the grounded extension. This explanation is done in an interactive way, consisting of a structured discussion between human and computer. The min-max numbering serves as a roadmap regarding the moves that should be done by the computer in order to win the discussion (see [4] for details).

Although a min-max numbering does have its usefulness, the question is what are the computational resources needed to construct it. In particular, what is the additional runtime that is required for constructing the min-max numbering of a strongly admissible labelling?

In order to answer this question, we compare the runtime of Algorithm 1 with the runtime of a modified version of Algorithm 1 which only constructs the strongly admissible labelling itself (without also constructing its min-max numbering). This can be done by commenting out the lines 12, 21 and 27, and by returning only $\mathcal{L}ab$ (instead of both $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$ in lines 13 and 28. We refer to the thus modified algorithm as Algorithm 1″.

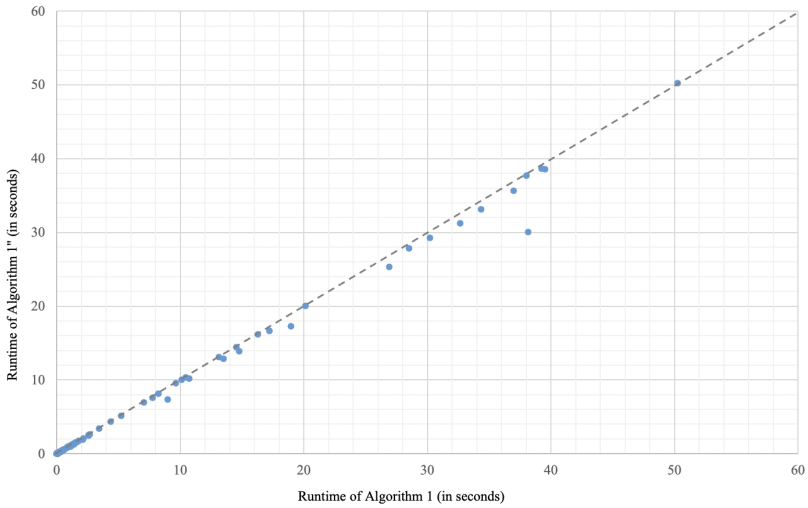

Figure 4: The runtime of Algorithm 1 compared with the runtime of Algorithm 1″.

[8]This is despite the fact that Algorithm 3 of [13] only constructs a strongly admissible (grounded) labelling, whereas Algorithm 1 constructs both a strongly admissible labelling and its associated min-max numbering.

Figure 4 depicts the results of Algorithm 1″ versus the runtime of Algorithm 1 (each dot represents one of the 277 benchmark examples that we tested for). On average, Algorithm 1 takes just 1.83% more time than Algorithm 1″.

Similar to the way in which we modified Algorithm 1, one may wonder what would happen if the min-max numbering was no longer constructed by Algorithm 3. That is, what if we change Algorithm 3 (call it Algorithm 3″) to include Algorithm 1″ (instead of Algorithm 1) and Algorithm 2″ (instead of Algorithm 2), where Algorithm 2″ does not use or yield any min-max numbering? Unfortunately, this turns out not to be possible. The point is that Algorithm 2 (in lines 18 and 19) requires the min-max numbering that is produced by Algorithm 1.

To see the essence of the problem, consider the argumentation framework of Figure 5. Suppose the argument in question is E. Algorithm 1 would yield the strongly admissible labelling $(\{A, C, E\}, \{B, D\}, \emptyset)$ and associated min-max numbering $\{(A : 1), (B : 2), (C : 3), (D : 4), (E : 5)\}$, which are subsequently fed into Algorithm 2. By the time Algorithm 2 reaches argument B, it will choose B's attacker A instead of E, as A has the smallest min-max number, and the algorithm will terminate. However, without the min-max numbering provided by Algorithm 1, Algorithm 2 would not know whether to choose A or E, which means it might enter an infinitive loop if it chooses E consistently.[9] As such, the fact that we can remove the concept of a min-max numbering from Algorithm 1 does not imply we can also remove the concept of a min-max numbering from Algorithm 2, at least not in any straightforward way.

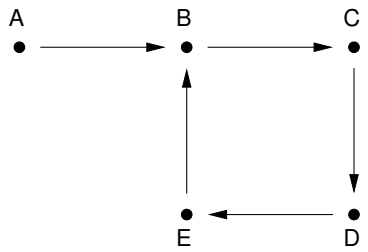

Figure 5: An example argumentation framework.

There is, however, at least one additional way in which we can assess the computational costs of constructing the min-max labelling. Recall that Algorithm 1 can be modified in order to compute the grounded labelling by commenting out lines 13 and 28, and by adding the following line (line 33) at the end of the algorithm:
return $\mathcal{L}ab$ and $\mathcal{MM}_{\mathcal{L}ab}$
The thus modified algorithm is referred to as Algorithm 1′.

Of course, we could compare the performance of Algorithm 1′ with the performance of a modified version of Algorithm 1′ that constructs only the grounded labelling (without its associated min-max numbering). However, it would be more interesting to compare the performance of Algorithm 1′ with an algorithm that was optimised for constructing the grounded labelling

[9] Apart from that, not having access to the min-max numbering of Algorithm 1 could also cause Algorithm 2 to yield a bigger strongly admissible labelling than it would otherwise have yielded.

only. For this, we will use Algorithm 3 of [13]. Although somewhat similar to our Algorithm 1′, Algorithm 3 of [13] uses a set instead of an ordered list for processing the arguments, as it does not need to provide any min-max numbering.[10]

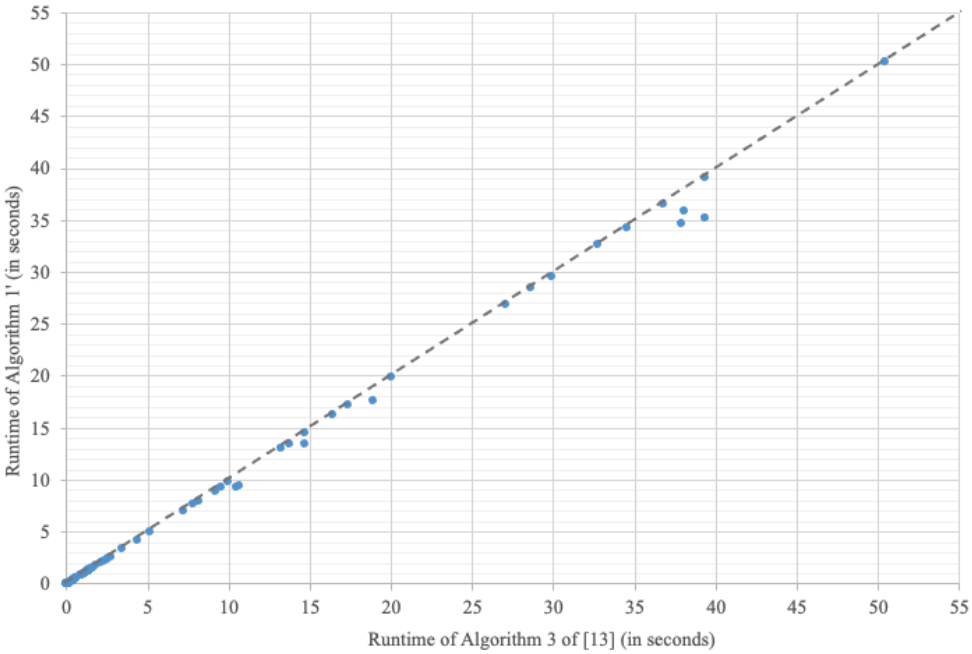

Figure 6: The runtime of Algorithm 1′ compared to the runtime of Algorithm 3 of [13].

Figure 6 depicts the runtime of Algorithm 1′ versus the runtime of Algorithm 3 of [13] (each dot represents one of the 277 benchmark examples that we tested for). On average, Algorithm 3 of [13] takes 3.62% more time than Algorithm 1.

5. Discussion

Our empirical results indicate that when it comes to constructing an arbitrary strongly admissible labelling (for a particular argument) Algorithm 1 provides a relatively efficient way (regarding runtime) of doing so. In the examples we tested for, Algorithm 1 clearly outperformed some of its alternatives (Algorithm 3 by a factor 2, the ASPARTIX approach of [7] by a factor of more than 6, and Algorithm 3 of [13] by almost 3%). Moreover, where some of its alternatives only yield a strongly admissible labelling, Algorithm 1 yields both a strongly admissible labelling and its associated min-max numbering.

[10] See [5] for a discussion of why an ordered list is necessary when the aim is to construct the min-max numbering as well.

One of the things we did not do in our current work is to compare the performance of Algorithm 1 with μ-toksia [14], which is the fastest correct algorithm for grounded semantics that was submitted to ICCMA'19.[11] However, we did run some additional experiments with respect to the ASPARTIX encodings for grounded semantics [15] and found that Algorithm 3 of [13] runs in just 7.8% of the time that is needed by ASPARTIX for grounded semantics. As the ICCMA'19 results indicate that μ-toksia for grounded semantics runs in 25.9% of the time needed by ASPARTIX for grounded semantics, it seems likely that Algorithm 3 of [13] would run faster than μ-toksia in a direct comparison. Given that our Algorithm 1 already outperforms Algorithm 3 of [13], it is likely that it will also outperform μ-toksia.

Our results indicate that the additional computational time required to construct the min-max numbering (that is, the difference in runtime between constructing both a strongly admissible labelling and its min-max numbering, and constructing a strongly admissible labelling only) is negligible.[12] In particular, commenting out the code in Algorithm 1 that constructs the min-max numbering (Algorithm 1″) yields a performance increase of less than 2%. Moreover, our approach to constructing the grounded labelling (Algorithm 1′), which also constructs the associated min-max numbering, even outperforms (by more than 3%) the fastest imperative algorithm for constructing the grounded extension/labelling in the literature that we are aware of (Algorithm 3 of [13]), even though the latter does not even construct the min-max numbering.

Overall, our findings indicate that Algorithm 1 is a good candidate for computing strong admissibility. As such, we would consider submitting it to ICCMA in case a track on strong admissibility is opened.

Acknowledgments

This work was supported by the BMK Endowed Professorship for Data-Driven Knowledge Generation: Climate Action (FFG project number 891607).

References

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

[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 revisited: 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.

[11] ICCMA'19 was the last ICCMA that had a track on grounded semantics.

[12] This may be related to the fact that Algorithm 1 constructs the strongly admissible labelling and its associated min-max number concurrently, rather than first constructing the strongly admissible labelling and then constructing its associated min-max numbering.

[5] M. Caminada, S. Harikrishnan, Tractable algorithms for strong admissibility, Argument & Computation (2024). In print.
[6] 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.
[7] 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.
[8] P. Baroni, M. Caminada, M. Giacomin, Abstract argumentation frameworks and their semantics, in: Handbook of Formal Argumentation, volume 1, College Publications, 2018, pp. 159–236.
[9] P. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artificial Intelligence 77 (1995) 321–357.
[10] 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.
[11] M. Caminada, D. Gabbay, A logical account of formal argumentation, Studia Logica 93 (2009) 109–145. Special issue: new ideas in argumentation theory.
[12] M. Caminada, G. Pigozzi, On judgment aggregation in abstract argumentation, Autonomous Agents and Multi-Agent Systems 22 (2011) 64–102.
[13] S. Nofal, K. Atkinson, P. Dunne, Computing grounded extensions of abstract argumentation frameworks, The Computer Journal 64 (2021) 54–63.
[14] A. Niskanen, M. Järvisalo, μ-toksia: An efficient abstract argumentation reasoner, in: Proceedings of the 17th International Conference of Knowledge Representation and Reasoning (KR 2020), 2020, pp. 800–804.
[15] W. Dvořák, A. Rapberger, J. Wallner, S. Woltran, Aspartix-v19 - an answer-set programming based system for abstract argumentation, in: International Symposium on Foundations of Information and Knowledge Systems, 2020, pp. 79–89.