In this paper, we address the above issues and provide complexity results as well as implementations for bipolar ABA, and therefore indirectly for various BAFs. Specifically, we analyse the (non-empty) existence, verification, (credulous and sceptical) acceptance and enumeration complexity problems in bipolar ABA under the semantics capturing various BAFs. We establish that bipolar ABA is equally as complex as abstract argumentation (AA) \cite{Dra86}. We then give algorithms for extension enumeration in bipolar ABA, which effectively capture solutions to other complexity problems too. We describe an implementation of bipolar ABA as well as various BAFs and complement it with a scalability evaluation, showing that our system is fit for practical deployment.

The paper is organised as follows. Section 2 provides background on argumentation and complexity theory, as well as existing complexity results for AA. We give the complexity results for bipolar ABA in Section 3. In Section 4 we advance new algorithms for implementing bipolar argumentation. We describe the software system implementing these algorithms in Section 5, alongside evaluating the system’s scalability practically. We review related work in Section 6 and discuss conclusions and future work in Section 7.
2. Elements of Complexity

We assume knowledge of fundamental time and space complexity classes, as well as the concepts of hardness and completeness [31]. Thus, we here recap the complexity problems studied in argumentation, as well as established results for AFs.

2.2.1 Enumeration. We first give (the less standard) enumeration problems and related complexity classes following [24]. An enumeration problem is a pair $(L, \text{Sol})$ such that $L \subseteq \Sigma^*$ (for an alphabet $\Sigma$ containing at least two symbols) and $\text{Sol} : \Sigma^* \rightarrow 2^L$ is a function such that for all $x \in \Sigma^*$, we have that the set of solutions $\text{Sol}(x)$ is finite, and $\text{Sol}(x) = \emptyset$ iff $x \notin L$. An enumeration algorithm $A$ for an enumeration problem $P = (L, \text{Sol})$ outputs, on input $x$, exactly the elements from $\text{Sol}(x)$ without duplicates. For enumeration algorithms, we use the RAM model of computation [24].

The complexity classes OutputP and nOP are defined thus. Let $P = (L, \text{Sol})$ be an enumeration problem. $P \in \text{OutputP}$ if there exists an enumeration algorithm $\tilde{A}$ for $P$ and some $m \in \mathbb{N}$, such that on every input $x$, algorithm $\tilde{A}$ terminates in time $O(|x|^m + |\text{Sol}(x)|^m)$. Problems not in OutputP constitute the class nOP.

The following decision problem — \textsc{Manysol}(P): Given $x \in L$ and a positive integer $m$ in unary notation, is $|\text{Sol}(x)| \geq m^2$ — is strongly related to the enumeration problem $P = (L, \text{Sol})$. If \textsc{Manysol}(P) $\notin \text{P}$, then $P \notin \text{OutputP}$. We will use \textsc{Manysol} in our analysis of the enumeration problem in bipolar ABA.

2.2.2 Problems of Interest. We now state the problems we are interested in. In the following, $F$ stands for a bipolar ABA framework and $\sigma \in \{\text{adm, prf, set-stb}\}$ denotes a semantics, where $\text{adm, prf}$ and $\text{set-stb}$ abbreviate admissible, preferred and set-stable, respectively.

1. \textbf{Existence ($\text{Ex}_F^\sigma$):} Does $F$ admit a $\sigma$ extension?

2. \textbf{Non-Empty Existence ($\text{NE}_F^\sigma$):} Does $F$ admit a non-empty $\sigma$ extension?

3. \textbf{Verification ($\text{VER}_F^\sigma(A)$):} Given $A \subseteq A$, is $A$ a $\sigma$ extension of $F$?

4. \textbf{Credulous Acceptance ($\text{CA}_F^\sigma(a)$):} Given $a \in L$, is there a $\sigma$ extension $A$ of $F$ such that $A \vdash^R a$ for some $A' \subseteq A$ and $R \subseteq \mathcal{R}$?

5. \textbf{Sceptical Acceptance ($\text{SA}_F^\sigma(a)$):} Given $a \in L$, is it the case that for every $\sigma$ extension $A$ of $F$ it holds that $A' \vdash^R a$ for some $A' \subseteq A$ and $R \subseteq \mathcal{R}$?

6. \textbf{Extension Enumeration ($\text{EE}_F^\sigma$):} Return all $\sigma$ extensions of $F$.

The above complexity problems admit natural counterparts in BAFs (as well as AFs). In fact, the only difference is in the credulous and sceptical acceptance problems, for which instead of asking for deductions as in bipolar ABA, one asks for containment in extensions in B(ABA), see e.g. [15, 16, 19]. As various BAFs are captured in bipolar ABA via a polynomial mapping [12], our complexity results for bipolar ABA in this paper will cover various BAFs too.

Existing complexity results for AFs are summarised in Table 1 (stable stands for stable); see [15, 16] for surveys of these results.

| sem | Ex | NE | VER | CA | SA | EE |
|-----|----|----|-----|----|----|----|
| adm | Trivial (Y) | NP-c | P | NP-c | Trivial (N) | nOP |
| prf | Trivial (Y) | NP-c | coNP-c | NP-c | T^P | nOP |
| set-stb | NP-c | NP-c | P | NP-c | coNP-c | nOP |

3 COMPLEXITY RESULTS

In this section, we prove new complexity results for the complexity problems in bipolar ABA. Table 2 summarises our results.

| sem | Ex | NE | VER | CA | SA | EE |
|-----|----|----|-----|----|----|----|
| adm | Trivial (Y) | NP-c | P | NP-c | Trivial (N) | nOP |
| prf | Trivial (Y) | NP-c | coNP-c | NP-c | T^P | nOP |
| set-stb | NP-c | NP-c | P | NP-c | coNP-c | nOP |

These results show that the problems for bipolar ABA frameworks belong to precisely the same complexity classes as their corresponding problems for AFs. As a consequence, the same results apply to the various BAFs investigated in [12].

We first present prerequisite results needed for all of the problems, then we study verification, before moving on to existence, acceptance, and enumeration problems. Note that because there exists a polynomial time mapping between AFs and bipolar ABA frameworks [12], all the computational problems for bipolar ABA are at least as hard as their AF counterparts.

Throughout, unless stated otherwise, we assume as given a fixed but otherwise arbitrary bipolar ABA framework $F = (L, \mathcal{R}, A^-)$.

3.1 Prerequisite Results

The derivability problem for ABA frameworks is as follows.
Proposition 3.1. \( \text{DER}^F(A, a) \) is NL-complete (thus in P).

Proof. The following algorithm operates in logspace and nondeterministically solves the \( \text{DER}^F(A, a) \) problem: (1) Create a variable \( \beta \). (2) Set \( \beta \) equal to an arbitrary element of \( A \). If \( \beta = a \), output ‘yes’. Otherwise continue. (3) Initiate a counter \( k = 0 \). (4) Pick an arbitrary rule \( R \in R \) s.t. \( \{ \beta \} \) is the body of \( R \). If no such rule exists, output ‘no’. Otherwise continue. (5) If the head of \( R \) is equal to \( a \), output ‘yes’. Otherwise continue. (6) Set \( \beta \) equal to the head of \( R \). Increment \( k \) by 1. If \( k \geq |A| \), output ‘no’. Otherwise return to step 4. Note that this algorithm operates in logspace since the space usage of counter \( k \leq \log(|R|) \).

Hardness. We provide a (logspace) reduction from Reachability, the canonical NL-complete problem [31].

Reachability (RCH(G, s, t)): Given a directed graph \( G \) and vertices \( s \) and \( t \) of \( G \), is there a path from \( s \) to \( t \) in \( G \)?

The mapping below transforms a directed graph \( G \) into a language \( \mathcal{L} \) and a set of bipolar ABA rules \( R \):

\[
\mathcal{L} = A = \{ x : x \text{ is a node of } G \},
\]

\[ R = \{ y \leftarrow x : x \text{ and } y \text{ are nodes of } G \text{ and there exists an edge from } x \text{ to } y \text{ in } G \}, \]

\[ \triangledown = \alpha \text{ for } \alpha \in A. \]

This is a logspace transformation since we only need two counters to track the node and edge being considered at any point. Moreover, there is a path from \( s \) to \( t \) in \( G \) iff \( s = t \) or there is a chain of rules \( R \subseteq R \) of the form \( t \leftarrow y_n \leftarrow \ldots \leftarrow y_2 \leftarrow y_1 \leftarrow s \). This is precisely the condition in which \( \text{DER}^F((s), t) \) would output ‘yes’. Thus \( \text{RCH}(G, s, t) \) is logspace reducible to \( \text{DER}^F(A, a) \). This means that \( \text{DER}^F(A, a) \) is NL-hard.

We now analyse the fundamental properties of conflict-freeness and closure pertaining to all semantics considered in this paper.

Conflict-Freeness (CF\( F(A) \)): Given \( A \subseteq A \), is \( A \) conflict-free in \( F \)?

Closure (CL\( F(A) \) ): Given \( A \subseteq A \), is \( A \) closed in \( F \)?

Proposition 3.2. CF\( F(A) \) and CL\( F(A) \) are in P.

Proof. We present P-time algorithms for both problems:

CF\( F(A) \) : For each \( \alpha \in A \), use an NL oracle for \( \text{DER}^F(A, \overline{\alpha}) \) to check if \( A \vdash_{\text{ABA}} \{ \alpha \} \). If it does, output ‘no’. Else, output ‘yes’.

CL\( F(A) \) : For each \( \alpha \notin A \), use an NL oracle for \( \text{DER}^F(A, \overline{\alpha}) \) to check if \( A \vdash_{\text{ABA}} \{ \alpha \} \) for some \( R \subseteq R \). If it does, output ‘no’. Otherwise, output ‘yes’.

We will use the following result, which says that in a bipolar ABA framework, no sentence is deductible without a set.

Lemma 3.3. There is no deduction in \( F \) of the form \( \emptyset \vdash F \) for any \( \emptyset \in \mathcal{L} \) and \( R \subseteq R \).

Proof. Bipolar ABA frameworks do not contain any facts (i.e. rules of the form \( \alpha \leftarrow \top \)). Hence, it is impossible to have \( \top \) as the child of any node in a bipolar ABA deduction. As a result, a deduction of the form \( \emptyset \vdash F \) does not exist.

3.2 Verification

We now analyse the complexity of the verification problem under the admissible, preferred and set-stable semantics. In order to prove the results for admissible semantics, we first introduce the notion of minimal attacks in ABA.

Definition 3.4. \( A \subseteq A \) minimally attacks \( B \subseteq A \), denoted by \( A \vdash_{\text{min-ABA}} B \), if \( A \vdash_{\text{ABA}} B \) and there is no \( A' \subseteq A \) s.t. \( A' \vdash_{\text{ABA}} B \).

Lemma 3.5. All minimal attacks are of the form \( \{ \alpha \} \vdash_{\text{min-ABA}} B \) where \( \alpha \in A \) and \( B \subseteq A \).

Proof. Assume there are \( A \subseteq A \) and \( B \subseteq A \) s.t. \( A \vdash_{\text{min-ABA}} B \) and \( |A| \neq 1 \). Then we have two cases: (1) \( |A| = 0 \). Lemma 3.3 implies that \( A \vdash_{\text{ABA}} B \). This contradicts \( A \vdash_{\text{min-ABA}} B \). (2) \( |A| > 1 \): In order for \( A \vdash_{\text{ABA}} B \) there must exist \( \alpha \in A \) and \( \beta \in B \) where either \( \alpha = \beta \) or there exists a chain of rules \( \gamma \leftarrow \gamma_n \leftarrow \ldots \leftarrow \gamma_2 \leftarrow \gamma_1 \leftarrow \alpha \). In both cases we have \( \{ \alpha \} \vdash_{\text{ABA}} B \). However, \( \{ \alpha \} \subset A \) so we have a contradiction to the definition of minimal attacks. In any event, \( A = \{ \alpha \} \) where \( \alpha \in A \) as required.

Proposition 3.6. \( \text{VER}^F_{\text{adm}}(A) \) is in P.

Proof. (1) Use a P oracle for \( \text{CF}^F(A) \) to check if \( A \) is conflict-free. If it is not, output ‘no’. Otherwise continue. (2) Use a P oracle for \( \text{CL}^F(A) \) to check if \( A \) is closed. If it is not output ‘no’. Otherwise continue. (3) For each assumption \( \beta \notin A \), call an NL oracle for \( \text{DER}^F(A, \overline{\beta}) \) |\( A | \) times (once for each \( \alpha \in A \)) to check if \( \{ \beta \} \vdash_{\text{ABA}} A \). If it does and \( A \vdash_{\text{ABA}} \{ \beta \} \) output ‘no’. Otherwise continue. (4) Output ‘yes’.

Note that step 3 is sufficient to check that \( A \) defends itself. This follows from the fact that if \( B \vdash_{\text{ABA}} A \) then \( \{ \beta \} \vdash_{\text{ABA}} A \) for some \( \beta \in B \) (Lemma 3.5). From the definition of attacks, it follows that if \( A \vdash_{\text{ABA}} \{ \beta \} \) then \( A \vdash_{\text{ABA}} B \) as well. Moreover, since step 1 of the algorithm checks that \( A \) is conflict-free, we know that \( \beta \notin A \). So it suffices to prove that \( A \) defends itself against singleton sets of assumptions which are not contained within it.

Proposition 3.7. \( \text{VER}^F_{\text{adm}}(A) \) is coNP-Complete.

Proof. Membership comes from the following non-deterministic, P-time algorithm, adapted from [13], which solves the \( \text{coVER}^F_{\text{adm}}(A) \) problem: (1) Use a P oracle for \( \text{CF}^F(A) \) to check if \( A \) is admissible. If it is not, output ‘yes’. Otherwise continue. (2) Guess an assumption set \( A' \supset A \). (3) Use a P oracle for \( \text{VER}^F_{\text{adm}}(A') \) to check if \( A' \) is admissible. If it is, output ‘yes’. Otherwise output ‘no’.

Proposition 3.8. \( \text{VER}^F_{\text{set-stb}}(A) \) is in P.

Proof. We present the following P-time algorithm: (1) Use a P oracle for \( \text{CF}^F(A) \) to check if \( A \) is conflict-free. If it is not, output ‘no’. Else continue. (2) Use a P oracle for \( \text{CL}^F(A) \) to check if \( A \) is closed. If it is not, output ‘no’. Else continue. (3) For each \( \alpha \notin A \), calculate \( \text{Cl}(\{ \alpha \}) \) by calling an NL oracle for \( \text{DER}^F(A) \) |\( A | \) times, and then check if \( A \vdash_{\text{ABA}} \text{Cl}(\{ \alpha \}) \) using an additional \( |\text{Cl}(\{ \alpha \})| \) oracle calls. If it does not, output ‘no’. Otherwise, output ‘yes’.

□
3.3 Existence and Acceptance

Before proving the remainder of our results we make the following observations.

Proposition 3.9. $\text{EX}_{adm}^\mathcal{F}$, $\text{NE}_{adm}^\mathcal{F}$ and $\text{CA}_{adm}^\mathcal{F}$ are respectively equivalent to $\text{EX}_{prf}^\mathcal{F}$, $\text{NE}_{prf}^\mathcal{F}$ and $\text{CA}_{prf}^\mathcal{F}$.

Proof. Follows from the fact that every preferred extension is admissible and every admissible extension is a subset of some preferred assumption set [13, Prop1].

Proposition 3.10. $\text{EX}_{set-st}^\mathcal{F}$ is equivalent to $\text{NE}_{set-st}^\mathcal{F}$.

Proof. Assume $\emptyset$ is set-stable in $\mathcal{F}$. As $\mathcal{A} \neq \emptyset$, there is $\alpha \in \mathcal{A}$ s.t. $\emptyset \not\sim_{\text{ABA}} \text{Cl}((\alpha))$. But this contradicts Lemma 3.3. Thus, $\emptyset$ is never set-stable in $\mathcal{F}$, and so existence of a set-stable extension is equivalent to the existence of a non-empty set-stable extension.

3.3.1 Existence. We now consider (non-empty) existence.

Proposition 3.11. $\text{EX}_{adm}^\mathcal{F}$ and $\text{EX}_{prf}^\mathcal{F}$ are constant, with answer ‘yes’.

Proof. $\emptyset$ is conflict-free, defends itself, and, by Lemma 3.3, is closed. Hence, $\emptyset$ is admissible, which establishes the claim for $\text{EX}_{adm}^\mathcal{F}$. The claim for $\text{EX}_{prf}^\mathcal{F}$ then follows from Proposition 3.9.

Now we switch our attention to the non-emptiness problem.

Proposition 3.12. $\text{NE}_{adm}^\mathcal{F}$, $\text{NE}_{prf}^\mathcal{F}$, $\text{NE}_{set-st}^\mathcal{F}$ and $\text{EX}_{set-st}^\mathcal{F}$ are NP-Complete.

Proof. The following non-deterministic, $P$-time algorithm proves membership for admissible and set-stable semantics. The results for $\text{NE}_{adm}^\mathcal{F}$ and $\text{EX}_{set-st}^\mathcal{F}$ follow from Propositions 3.9 and 3.10.

1. Guess an assumption set $\mathcal{A} \subseteq \mathcal{A}$. (2) Use a $P$ oracle for $\text{VER}_{adm}^\mathcal{F}(\mathcal{A})$ (or $\text{VER}_{set-st}^\mathcal{F}(\mathcal{A})$) to check if $\mathcal{A}$ is admissible (or set-stable) extension. If not, output ‘no’. Otherwise Output ‘yes’. □

3.3.2 Credulous and Sceptical Acceptance. We now turn to acceptance problems.

Proposition 3.13. $\text{CA}_{adm}^\mathcal{F}(\alpha)$, $\text{CA}_{prf}^\mathcal{F}(\alpha)$, and $\text{CA}_{set-st}^\mathcal{F}(\alpha)$ are NP-complete, $\text{SA}_{adm}^\mathcal{F}(\alpha)$ is coNP-complete, and $\text{SA}_{prf}^\mathcal{F}(\alpha)$ is $\Sigma_2^p$-complete.

Proof. Membership uses the following algorithm, adapted from [13], solving $\text{CA}_{adm}^\mathcal{F}(\alpha)$ and $\text{coSA}_{adm}^\mathcal{F}(\alpha)$, and our previous results for $\text{VER}_{adm}^\mathcal{F}(\alpha)$. The result for $\text{CA}_{prf}^\mathcal{F}(\alpha)$ follows from Proposition 3.9.

1. Guess $\mathcal{A} \subseteq \mathcal{A}$. (2) Use a $\text{VER}_{adm}^\mathcal{F}(\mathcal{A})$ oracle to check if $\mathcal{A}$ is a $\sigma$ extension. If it is not, output ‘no’. Otherwise continue. (3) Use an NL oracle for $\text{DER}_{adm}^\mathcal{F}$ to check that the formula under consideration is derivable (or not derivable for $\text{coSA}_{adm}^\mathcal{F}(\alpha)$) from $\mathcal{A}$ and $\mathcal{R}$. If it is not, output ‘no’. Otherwise, output ‘yes’. □

Sceptical acceptance under admissible semantics is trivial.

Proposition 3.14. $\text{SA}_{adm}^\mathcal{F}(\alpha)$ is constant, with answer ‘no’.

Proof. $\emptyset$ is admissible (as in the proof of Proposition 3.11), so any sentence derivable from all admissible extensions is derivable from $\emptyset$. However, by Lemma 3.3, no such sentence exists. □

We are left to address the enumeration problem.

3.4 Extension Enumeration

We here establish the complexity of EE in bipolar ABA using the MANYSPORT problem (see Section 2.2).

Proposition 3.15. $\text{EE}_{adm}^\mathcal{F}$, $\text{EE}_{prf}^\mathcal{F}$ and $\text{EE}_{set-st}^\mathcal{F}$ are in $\text{np}$, assuming $P \neq \text{NP}$.

Proof. MANYSPORT($\text{EE}_{adm}^\mathcal{F}$) is NP-hard for $\sigma \in \{\text{adm, prf, set-st}\}$ [24]. Because AFS can be mapped into flat bipolar ABA in $P$-time [12] and since stable and set-stable semantics coincide for flat ABA, MANYSPORT($\text{EE}_{adm}^\mathcal{F}$) is NP-hard for $\sigma \in \{\text{adm, prf, set-st}\}$. □

This completes the complexity analysis of bipolar ABA as summarised in Table 2.

4 ALGORITHMS

We have shown that many of the standard problems for bipolar ABA are non-tractable. As such, practical algorithms for solving them must make use of advanced techniques and heuristics. We now propose such algorithms for the $\text{EE}^\mathcal{F}$ problem. Note that, effectively, $\text{EE}^\mathcal{F}$ answers the other standard problems too. Having all the $\sigma$ extensions of $\mathcal{F}$ one can establish (non-empty) existence immediately, verification by checking membership in the set of enumerated extensions, and (credulous and sceptical) acceptance by using the efficient algorithm for derivation (cf. Proposition 3.1).

The algorithms in this section make use of a backtracking strategy. They recursively traverse a binary tree from left to right, where the root node is the empty set and the tree forks to a left (or right) node by including (or excluding) an assumption. If the current node represents a valid extension, it is added to the solution set. Backtracking occurs whenever the procedure is going down a path which will never lead to a correct solution, at this point, it moves back up the tree and takes a different path instead.

4.1 Enumeration of Preferred Extensions

We first give a basic algorithm for enumerating preferred extensions that conveys the main ideas. In what follows a labelling is a total mapping $\text{Lab} : \mathcal{A} \rightarrow \{\text{IN}, \text{OUT}, \text{UNDEC}, \text{BLANK}, \text{MUST\_OUT}\}$.

4.1.1 Basic Algorithm. We define some labelings which correspond to the different states of our algorithm while traversing the binary tree. These definitions and the algorithms following them are inspired by the corresponding work for AFS, particularly [28].

Definition 4.1. A labelling $\text{Lab}$ of $\mathcal{F}$ is:

• the initial labelling of $\mathcal{F}$ iff $\text{Lab} = \{\langle \alpha, \text{BLANK} \rangle : \alpha \in \mathcal{A} \setminus S \} \cup \{\langle \beta, \text{UNDEC} \rangle : \beta \in S \}$ where $S \subseteq \mathcal{A}$ is the set of all $y \in \mathcal{A}$ s.t. $\{y\} \not\sim_{\text{min-ABA}} \text{Cl}((\gamma))$.
• a terminal labelling of $\mathcal{F}$ iff for each $\alpha \in \mathcal{A}$, $\text{Lab}((\alpha)) \neq \text{BLANK}$.
• a hopeless labelling of $\mathcal{F}$ iff there exists an $\alpha \in \text{MUST\_OUT}$ s.t. for all $\beta \in \mathcal{A}$, if $\langle \beta, \text{UNDEC} \rangle$ then $\text{Lab}((\beta)) \not\in \{\text{OUT, UNDEC}\}$.
• an admissible labelling of $\mathcal{F}$ iff $\text{Lab}$ is a terminal labelling of $\mathcal{F}$ and $\text{MUST\_OUT} = \emptyset$.
• a preferred labelling of $\mathcal{F}$ iff $\text{Lab}$ is an admissible labelling of $\mathcal{F}$ and $\{x : \text{Lab}((x)) = \text{IN}\}$ is maximal (w.r.t. $\subseteq$) among all admissible labelings of $\mathcal{F}$. 

In the above, the initial labelling corresponds to the root of the binary tree. Terminal labelings correspond to leaf nodes of the tree. If there are no MUST_OUT assumptions in a terminal labelling, then we have an admissible labelling. Preferred labelings are those which are maximally admissible. Finally, hopeless labelings are those which are guaranteed to not reach an admissible labeling.

Next, we define two procedures of our algorithm, which correspond to taking the left or right path down our binary tree.

**Definition 4.2.** Let Lab be a labelling of $\mathcal{F}$, and $\alpha \in \mathcal{A}$.

- The **left-transition** of Lab to the new labelling Lab' using $\alpha$ is defined by: (1) Lab' ← Lab. (2) For each $\beta \in \text{Cl}(\{\alpha\})$, Lab'($\beta$) ← IN. (3) For each $\gamma \in \mathcal{A}$, if $\alpha \not{\sim}_{\text{min}\text{-ABA}} \text{Cl}(\{\gamma\})$, Lab'($\gamma$) ← OUT. (4) For each $\delta \in \mathcal{A}$, with Lab($\delta$) ≠ OUT, if $\delta \not{\sim}_{\text{min}\text{-ABA}} \text{Cl}(\{\alpha\})$, Lab'($\delta$) ← MUST_OUT.

- The **right-transition** of Lab to the new labelling Lab' using $\alpha$ is defined by: (1) Lab' ← Lab. (2) For each $\beta \in \mathcal{A}$, if $\alpha \in \text{Cl}(\{\beta\})$, Lab'($\beta$) ← UNDEC.

A left transition starts by labelling all assumptions in the closure of some target assumption as IN. We then label any assumptions whose closure is minimally attacked by the target assumption as OUT. After that, we can add those assumptions which minimally attack the closure of the target assumptions to MUST_OUT. In a right-transition, we label all assumptions whose closure contains the target assumption as UNDEC.

Algorithm 1 enumerates all the preferred extensions of $\mathcal{F}$.

**Proposition 4.3.** Algorithm 1 solves the EE$_{\text{prf}}$ problem.

**Proof outline.** Completeness. Algorithm 1 builds every closed, conflict-free subset of $\mathcal{F}$. This is guaranteed by our definitions of initial labelling, left-transition and right-transition.

**Soundness.** We need to show that the generated sets are maximal and admissible. Maximality is ensured by line 4 together with the fact that maximal sets are constructed first (by performing left-transitions before right-transitions). For admissibility we need to show that the sets in $E$ are closed, conflict-free and defend themselves. Closure is guaranteed because as soon as a new assumption is labelled IN, so is every element in its closure. Conflict-freeness is guaranteed since any assumption attacked by the set of IN assumptions is immediately labelled OUT. Defence is guaranteed by our usage of the MUST_OUT label and hopeless labelings.

Figure 1 shows an example of Algorithm 1 calculating the preferred extensions of a bipolar ABA framework. The algorithm starts with the initial labelling and forks to the left and right by performing the appropriate transition procedure (represented by left and right arrows in the figure). Leaf nodes are identified as either admissible or preferred extensions (although only preferred ones are saved). Moreover, the figure shows how the notion of hopeless labellings reduces the search-space of the algorithm.

1. **Improved Algorithm.** We now discuss several improvements of the basic algorithm given above, similarly to [28].

Algorithm 1 can be improved by introducing influential assumptions. The idea is to select the most influential assumption for a left-transition to reach a terminal or hopeless labelling faster.
We next give algorithms for enumerating admissible and set-stable extensions of $F$.  

**Algorithm 2: Enumerate_Preferred($F$, Lab, E)**

**input:** $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \rightarrow)$ is a bipolar ABA framework.
Lab : $\mathcal{A} \to \{\text{IN}, \text{OUT}, \text{UNDEC}, \text{BLANK}, \text{MUST\_OUT}\} \subseteq 2^\mathcal{A}$

**output:** Lab : $\mathcal{A} \to \{\text{IN}, \text{OUT}, \text{UNDEC}, \text{BLANK}, \text{MUST\_OUT}\} \subseteq 2^\mathcal{A}$

1. Propagate Lab;
2. if Lab is a hopeless labelling then return;
3. while Lab is not a terminal labelling do
   4. Select a new assumption $\alpha \in \mathcal{A}$ s.t. $\alpha$ is influential;
   5. Get a new labelling called Lab’ by applying the left-transition of Lab using $\alpha$;
   6. if Lab’ is not a hopeless labelling then
      7. Call Enumerate_Preferred($F$, Lab’, E);
   8. Update Lab by applying the right-transition of Lab using $\alpha$;
   9. if Lab is a hopeless labelling then return;
10. if Lab is an admissible labelling then
    11. if $\{x : \text{Lab}(x) = \text{IN}\}$ is not a subset of any set in $E$ then
        $E \leftarrow E \cup \{\{x : \text{Lab}(x) = \text{IN}\}\}$.

**Proposition 4.6.** Algorithm 2 solves the EE$_{prf}$ problem.

**Proof outline.** We show that none of the changes introduced in Algorithm 2 compromise soundness or completeness. (1) Selecting the most influential assumption does not compromise left and right transitions, because by definition this assumption will be labelled BLANK. (2) Changing the right transition to be performed as a while loop doesn’t change the order of operations. (3) Checking for hopeless labellings does not have any side effects, so doing it more often will not either. (4) Labelling propagation excludes only admissible labellings which are not preferred. □

### 4.2 Enumeration of Admissible and Set-Stable Extensions

We next give algorithms for enumerating admissible and set-stable extensions of $F$.  

**4.2.1 Enumeration of Admissible Extensions.** Algorithm 2 can be adapted to find admissible extensions. To achieve this, we first need to drop the maximality check. Moreover, the labelling propagation step needs to be removed. Indeed, if we do not remove it, then there is a risk that some admissible sets will be overlooked since these sets do not necessarily contain every assumption that they defend. This modification is achieved in Algorithm 3.

**Algorithm 3: Enumerate_Admissible($F$, Lab, E)**

**input:** $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \rightarrow)$ is a bipolar ABA framework.
Lab : $\mathcal{A} \to \{\text{IN}, \text{OUT}, \text{UNDEC}, \text{BLANK}, \text{MUST\_OUT}\} \subseteq 2^\mathcal{A}$

**output:** Lab : $\mathcal{A} \to \{\text{IN}, \text{OUT}, \text{UNDEC}, \text{BLANK}, \text{MUST\_OUT}\} \subseteq 2^\mathcal{A}$

1. if Lab is a hopeless labelling then return;
2. while Lab is not a terminal labelling do
   3. Select a new assumption $\alpha \in \mathcal{A}$ s.t. $\alpha$ is influential;
   4. Get a new Lab called Lab’ by applying the left-transition of Lab using $\alpha$;
   5. if Lab’ is not a hopeless labelling then
      6. Call Enumerate_Admissible($F$, Lab’, E);
   7. Update Lab by applying the right-transition of Lab using $\alpha$;
   8. if Lab is a hopeless labelling then return;
   9. if Lab is an admissible labelling then
      10. $E \leftarrow E \cup \{\{x : \text{Lab}(x) = \text{IN}\}\}$

Therefore, we have the following result:

**Proposition 4.7.** Algorithm 3 solves the EE$_{adm}$ problem.

**4.2.2 Enumeration of Set-Stable Extensions.** Algorithm 2 can also be adapted to find set-stable extensions. By definition any set-stable extension attacks the closure of all assumption sets it does not contain. Thus, the UNDEC label is no longer useful since any assumption which would have been labelled UNDEC in the case of preferred semantics should now be labelled MUST\_OUT. We change some of our definitions accordingly.

**Definition 4.8.** Let Lab be a labelling of $F$. Then:
- Lab is the initial set-stable labelling of $F$ iff Lab = $(\{\alpha, \text{BLANK}\} : \alpha \in \mathcal{A} \setminus S) \cup (\{\beta, \text{MUST\_OUT}\} : \beta \in S)$ where $S \subseteq \mathcal{A}$ is the set of all $y \in \mathcal{A}$ s.t. $\{y\} \sim_{\text{min}\_ABA} \text{Cl}(\{y\})$.
- Let $\alpha$ be an assumption in $\mathcal{A}$. Then the set-stable right-transition of Lab to the new labelling Lab’ using $\alpha$ is defined by actions:
  1. Lab’ $\leftarrow$ Lab. (1) For each $\delta \in \text{Cl}(\{\alpha\})$ with Lab(\delta) $\neq$ OUT, Lab’(\delta) $\leftarrow$ MUST\_OUT.
  2. Lab is a set-stable labelling of $F$ iff Lab is a terminal labelling of $F$ and MUST\_OUT $= \emptyset$.

The modifications are achieved in Algorithm 4. Thus, as with enumeration of admissible extensions, we have the following result:

**Proposition 4.9.** Algorithm 4 solves the EE$_{set-stab}$ problem.

We note that while conceptually similar algorithms exist for enumerating extensions of AFs (see e.g., [8, 27, 28]), adapting them to be used for bipolar ABA frameworks is not a trivial task. Specifically,
We now describe the control flow of our system as depicted graphically in Figure 2. In order to test the scalability of our system, we generated 405 bipolar ABA frameworks of increasing size. To do this we adapted an existing benchmark generator from [10], originally used to create flat ABA frameworks, and ensured that bipolar ABA frameworks are generated instead.

We input a tuple of parameters $(N_i, N_a, N_{rh}, N_{rph})$ to the generator in order to create our frameworks. The parameters are defined as follows: (1) $N_i$ is the total number of sentences in the framework, i.e., $|\mathcal{L}|$. (2) $N_a$ is the number of assumptions, i.e., $|\mathcal{A}|$, given as a percentage of the number of sentences. (3) $N_{rh}$ is the number of distinct sentences to be used as rule heads, given as an integer. (4) $N_{rph}$ is the number of rules per distinct rule head, given as an interval $[\text{min}, \text{max}]$, where min and max are integers.

The specific parameters used were $(N_i, 37\%, N_a/2, [2, N_a/8])$ with the value of $N_i$ starting at 16, and increasing by 8 between subsequent frameworks. The largest framework consisted of 3248 sentences, 1202 assumptions and 174,365 rules.

We measured the elapsed time between inputting a framework and outputting its extensions, under the admissible, preferred and set-stable semantics, for all generated frameworks. The elapsed times were very similar for all three semantics. Figure 3 shows the time taken to calculate extensions for each framework, averaged over the three semantics. These experiments were run on a home machine, with 16GB of memory and a 2.9GHZ, 2 core CPU.

The results show that even for the largest frameworks considered, our algorithms calculate extensions in under 25 seconds. We do see the performance begin to deteriorate as the size of the frameworks increase. This is expected since the backtracking method we rely on operates in $O(2^n)$ in the worst case. Overall, these results demonstrate the feasibility of our algorithms as a means of generating extensions of large argumentation frameworks.

All in all, we have presented a scalable system for computing and enumerating all extensions of bipolar ABA frameworks under the semantics considered in this paper. Consequently, the system computes and enumerates extensions of various formulations of argumentation frameworks.

---

**Algorithm 4: Enumerate Set-stable**

**Input:** $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{E})$ is a bipolar ABA framework. 

$\text{Lab} : \mathcal{A} \rightarrow \{\text{IN, OUT, BLANK, MUST OUT}\}$

$E \subseteq 2^A$

**Output:** $\text{Lab} : \mathcal{A} \rightarrow \{\text{IN, OUT, BLANK, MUST OUT}\}$

1. Propagate $\text{Lab}$;
2. if $\text{Lab}$ is a hopeless labelling then return;
3. while $\text{Lab}$ is not a terminal labelling do
4. Select a new assumption $\alpha \in \mathcal{A}$ s.t. $\alpha$ is influential;
5. Get a new labelling called $\text{Lab}'$ by applying the left-transition of $\text{Lab}$ using $\alpha$;
6. if $\text{Lab}'$ is not a hopeless labelling then
7. Call Enumerate Set-stable($F, \text{Lab}', E$);
8. Update $\text{Lab}$ by applying the right-transition-set-stable of $\text{Lab}$ using $\alpha$;
9. if $\text{Lab}$ is a hopeless labelling then return;
10. if $\text{Lab}$ is a set-stable labelling then return;
11. $E \leftarrow E \cup \{\{x : \text{Lab}(x) = \text{IN}\}\}$
In [19], the authors studied the complexity of BAFs under deductive support as in [6]. Specifically, Fazzinga et al. analysed the verification problem VER and established that it is in P under admissible, stable, complete and grounded semantics, and in coNP under preferred semantics. We instead studied the complexity of bipolar ABA, and thus indirectly of BAFs not only under deductive support, but also under other interpretations of support and with diverging semantics, as captured in bipolar ABA [12]. In addition, we analysed all the complexity problems standard in argumentation, namely EX, NE, VER, CA, SA and EE. To our knowledge, these problems have not been investigated for BAFs, except for the work of Fazzinga et al. We restricted our study to the admissible, preferred and set-stable semantics of bipolar ABA used to capture various BAFs, but we will extend our analysis to other semantics in the future.

Complexity of ABA was investigated in [13]. Dimopoulos et al. studied general non-flat ABA with respect to the complexity of the derivation problem in the underlying deductive system of an ABA framework, as well as various instances of ABA, including the (flat) logic programming instance, called LP-ABA. Specifically, they established the generic upper bounds for VER, as well as both upper bounds and instance-specific lower bounds for CA and SA under admissible, preferred and stable semantics. We note that DER in LP-ABA belongs to P [13], and AFs can be mapped in P-time into LP-ABA [37]. Thus, the results proven in this paper apply to LP-ABA as well. In particular, results provided in Section 3 complement the original work of Dimopoulos et al. on LP-ABA by giving new lower bounds for EX, NE and VER problems.

The Tweety libraries [36] provide implementations of various argumentation formalisms including AFs and ABA. Tweety can enumerate extensions of ABA frameworks under five semantics, including admissible, preferred and stable, but not set-stable semantics. Tweety essentially takes a brute force approach. For example, to compute the preferred extensions, it first generates all possible sets of assumptions and checks which ones are admissible. It then iterates through all these and checks which are maximal. This is very slow, as witnessed e.g. in a framework with ten assumptions, where Tweety takes more than 5 minutes to calculate extensions. In contrast, we showed our algorithms to be efficient in situations with hundreds of assumptions (on the same hardware).

In [17], Egly et al. provide an implementation of deductive BAFs [6], but not of other approaches to BAFs. Their system reduces the problems to instances of answer set programming whereas our works by directly calculating extensions. There are also other implementations of structured argumentation formalisms (see [7] for a recent survey), and those relevant to ABA (e.g. [21–23]) are reviewed in [2]. Except for the Tweety libraries discussed above, to the best of our knowledge no other implementations of non-flat ABA in general, or bipolar ABA in particular, exist.

7 Conclusions and Future Work

In this paper, we established the computational complexity of six problems, namely (non-empty) existence, verification, (credulous and sceptical) acceptance and enumeration, for bipolar Assumption-Based Argumentation (ABA) under the admissible, preferred and set-stable semantics. Our results carry over to various Bipolar Argumentation Frameworks (BAFs) that are instances of bipolar ABA. We also provided novel algorithms for extension enumeration, consequently addressing the remaining problems, for bipolar ABA. Using these algorithms, we gave an implementation of bipolar ABA and various BAFs, and showed that it scales well. We have therefore provided solid theoretical foundations and realised an implementation underlying the practical deployment of bipolar argumentation.

In the future, we plan on extending our analysis to generalisations of bipolar ABA. We will explore whether empowering these frameworks with new capabilities, such as support for factual rules or rules with multiple elements in their body, will lead to an increase in complexity. Moreover, we plan to extend our labelling algorithms to work for all ABA frameworks. Such algorithms will find use in an even wider range of practical scenarios than those described in this paper, due to the higher expressive power of generic ABA.

Acknowledgements. The authors were supported by the EPSRC project EP/P029558/1 ROAD2H: Resource Optimisation, Argumentation, Decision Support and Knowledge Transfer to Create Value via Learning Health Systems.

Data access statement: All data created during this research is available at github.com/AminKaram/BipolarABASolver. For more information please contact Amin Karamlou at mak514@ic.ac.uk.

REFERENCES

[1] Leila Amgoud and Mathieu Serrurier. 2008. Agents that Argue and Explain Classifications. Autonomous Agents and Multi-Agent Systems 16, 2 (2008), 187–209. https://doi.org/10.1007/s10458-007-9025-6

[2] Ziyi Bao, Kristijonas Cýras, and Francesca Toni. 2017. ABAPlus: Attack Reversal in Abstract and Structured Argumentation with Preferences. In PRIMA 2017: Principles and Practice of Multi-Agent Systems - 20th International Conference (Lecture Notes in Computer Science). Bo An, Ana L. C. Bazzan, João Leite, Serena Villata, and Leendert van der Torre (Eds.). Springer, Nice, 420–437. https://doi.org/10.1007/978-3-319-69131-2_25

[3] Pietro Baroni, Serena Borsato, Antonio Rago, and Francesca Toni. 2018. The “Games of Argumentation” Web Platform. In Computational Models of Argument - Proceedings of COMMA 2018, Warsaw, Poland, 12-14 September 2018. 447–448. https://doi.org/10.3233/978-1-61499-906-5-447

[4] Andrei Bondarenko, Phan Minh Dung, Robert Kowalski, and Francesca Toni. 1997. An Abstract Argumentation/Thetoric Approach to Default Reasoning. Artificial Intelligence 93, 97 (1997), 63–101. https://doi.org/10.1016/S0004-3702(97)00015-5
