Fast Computing of Dung Semantics in Acyclic Probabilistic Argumentation Frameworks

Stefano Bistarelli, Victor David, Pierre Monnin, Francesco Santini, Carlo Taticchi

Fast Computing of Dung Semantics in Acyclic Probabilistic Argumentation Frameworks

Stefano Bistarelli[1], **Victor David**[2*], **Pierre Monnin**[2], **Francesco Santini**[1], **Carlo Taticchi**[1]

[1]Department of Mathematics and Computer Science University of Perugia, Italy
[2]Université Côte d'Azur, Inria, CNRS, I3S, Sophia-Antipolis, France
firstname.lastname@{unipg.it,inria.fr}

Abstract

This paper presents fast and exact methods for computing the probability of an argument's acceptance using Dung's semantics in the Constellation paradigm of Abstract Argumentation. For (directed) *Singly-Connected Graphs* (*SCG*s), the problem can now be solved in linearithmic time instead of being exponential in the number of attacks, as reported in the literature. Moreover, in the more general case of *Directed Acyclic Graphs* (*DAG*s), we provide an algorithm whose time complexity is linearithmic in the product of the out-degree of *dependent arguments*, i.e., arguments reaching the argument considered for acceptance through multiple paths in the graph. We theoretically show that this complexity is lower than the lower-bound of the (exact) Constellation method, which is also supported by empirical results. We also compare our approach on DAGs with the (approximate) Monte-Carlo method, which is stopped when our approach obtains the exact results. Within this time constraint, Monte-Carlo still outputs significant errors, underlying the fast computation of our approach.

1 Introduction

The pioneering article in the field of Abstract Argumentation comes from P.M. Dung (Dung 1995), who defined the notion of an *Abstract Argumentation Framework* (*AF*). AFs can be seen as directed graphs where the nodes are arguments, and the edges represent conflict relations (called attacks) between two arguments. Since Dung, many extensions have been proposed, e.g., the addition of a support relation (Cohen et al. 2014), the addition of a similarity relation (Amgoud and David 2018, 2021), or the addition of weights (Bistarelli, Rossi, and Santini 2018).

In this paper, we consider *Probabilistic AF*s (or *PrAFs*) and specifically the *Constellation* paradigm (Li, Oren, and Norman 2011): probability values determine the likelihood of both arguments and attacks to be part of an AF, thus generating different possible AFs (i.e., "worlds") with a different existence probability each. Hence, the uncertainty lies in the topology of PrAFs.[1] To study an argument's acceptance probability, we need to compute this value in all the possible worlds induced by a PrAF and then aggregate these values. However, the number of induced AFs is generally exponential (Li, Oren, and Norman 2011), which undermines possible large-scale applications.

The motivations for our work originate from (Fazzinga, Flesca, and Parisi 2015; Fazzinga, Flesca, and Furfaro 2018), where the complexity of computing the acceptance probability of an argument using grounded semantics, which is also the focus of our paper, has been proven to be $FP^{\#P}$-$complete$. To overcome such a high complexity, the work in (Sun and Liao 2015) suggests some restrictions: if the probability of each argument is 0 or 1, then computing the acceptance probability is polynomial in time in the case of grounded semantics. If the probability is ternary, that is, 0, 0.5, or 1, then the same acceptance probability is P-hard. The authors of (Nofal, Atkinson, and Dunne 2021) propose a fast algorithm to compute the grounded extension for classic AFs, showing recent interest in a strictly related field.

This paper focuses on AFs in the shape of trees and, in general, on acyclic graphs, which have traditionally been significant in Argumentation. For example, several important debate platforms such as *Kialo.com* organize the discussion as a tree. A tree is well suited to represent specific forms of argumentation: multi-party debates can be modeled through debate trees in which agents reply to previously presented arguments (Bolton et al. 2020). Trees can support the fundamental turn-based mechanism in human dialogues (Amgoud, Parsons, and Maudet 2000). Such trees can be associated with probabilities automatically obtained from relation classification (Schindler 2020).

Moreover, AFs have also recently been used to explain (acyclic) neural networks (Potyka 2021). The works in (Toni, Rago, and Čyras 2023; Toni et al. 2023) merge probabilities and *Assumption-Based Argumentation*, the relationships between assumptions, claims, and counterarguments are naturally represented as trees.

The main goal of this work is to fastly compute the acceptance probability of an argument in the grounded extension of a PrAF by using local propagation, i.e., according to the acceptance probability of its direct attackers and the probability of incoming attacks. First, we do it very efficiently for directed graphs that follow the topology of *Singly-Connected Graph*s (*SCG*s), or Singly-Connected Networks, or "polytrees" (Chow and Liu 1968; Kim and Pearl 1983;

*corresponding author

[1]On the other hand, the epistemic paradigm (Hunter 2013) uses probability theory to represent degrees of belief in arguments.

Henrion 1988; Thomas, Howie, and Smith 2005).

An **SCG** is defined as a tree that may have multiple roots, where a node can have multiple parents, and there is exactly one unique path between two nodes.

In this case, we show that our propagation procedure is linearithmic (i.e., $\mathcal{O}(n\,logn)$) in the number of arguments. Next, we also deal with the more general case of *Directed Acyclic Graphs* (*DAGs*). In general, unlike SCGs, we may have to deal with at least two distinct paths between certain pairs of vertices; this is due to *dependent arguments*, i.e., arguments with two or more outgoing attacks reaching via a path the argument being considered for acceptance. This notion affects the complexity of the proposed algorithm, whose upper bound is linearitmic in the product of the out-degree of such dependent arguments. We also empirically estimate the asymptotic behavior in terms of time complexity on a selected testbed made of 200,000 arguments contained in 4,000 random DAGs.

On the same benchmark, the experiments show that our approach performs much better than computing all the induced worlds, as accomplished by the Constellation approach instead; as a reminder, both these methods are exact. A second round of tests shows that by stopping the computation at the same time as our algorithm finds a solution on a given DAG, the (approximate) Monte-Carlo approach used in (Li, Oren, and Norman 2011) returns a significant approximation error (around 38% on the average).

To summarize, the main results of this research are:

- In the case of SCGs, a new linearithmic complexity upper-bound to compute the probability of acceptance of an argument in all the Dung's semantics.[2]
- In the general case of DAGs, a fast algorithm that:
  - outperforms the other known exact approach in the literature (i.e., the Constellation approach) both in terms of formal complexity and experimental runtime;
  - outputs exact solutions in a time for which the Monte-Carlo approach still obtains significant errors.

The rest of the paper is organized as follows: Section 2 summarizes the necessary information about PrAFs and SCGs/DAGs. Section 3 introduces an algorithm to solve the problem in SCGs, i.e., `Fast_SCG`, together with its correctness and complexity results. Section 4 extends the idea to a more general algorithm, i.e., `Fast_DAG`, which works with DAGs. We discuss its correctness and time complexity and compare exact and approximate methods in the literature. We conclude with final thoughts in Section 5.

2 Background

Dung's Argumentation Frameworks. Following (Dung 1995), an argumentation framework (AF) is a pair $\langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a set of elements called arguments and $\mathcal{R}$ is a binary relation on $\mathcal{A}$, called the direct attack relation. For $a, b \in \mathcal{A}$, if $(a, b) \in \mathcal{R}$, then we say that a directly attacks b and that a is a direct attacker of b; If there exists a path from a to b with a length greater than 1, then a is considered an indirect attacker of b. In the rest of the paper, when the context is clear, we will use "attack" instead of "direct attack". If for $a \in \mathcal{A}$ there is no $b \in \mathcal{A}$ with $(b, a) \in \mathcal{R}$, then a is unattacked. We denote the set of (direct) attackers of a as $\texttt{Att}^-(a) = \{b \in \mathcal{A} \mid (b, a) \in \mathcal{R}\}$ and the set of (direct) outgoing attacks of a as $\texttt{Att}^+(a) = \{b \in \mathcal{A} \mid (a, b) \in \mathcal{R}\}$. For a set of arguments $E \subseteq \mathcal{A}$ and an argument $a \in \mathcal{A}$, E defends a if $\forall (b, a) \in \mathcal{R}$, $\exists c \in E$ such that $(c, b) \in \mathcal{R}$. We say that a is defended if for each last argument (unattacked) b_n for each path to a (i.e., $\{(b_n, b_{n-1}), \ldots, (b_1, a)\}$), all the b_n arguments defend a, i.e., n is even.

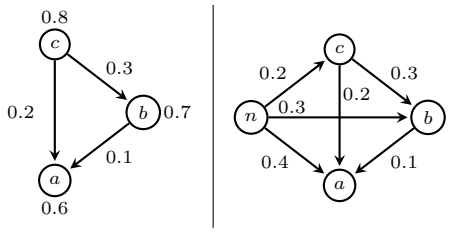

Figure 1: A PrAF (left) and its normal form (right) given in (Mantadelis and Bistarelli 2020).

An AF represents conflicting information, with reasoning performed using argumentation semantics. The semantics define acceptable arguments within an AF, and a collectively acceptable set of arguments, called an extension, serves as the reasoning outcome. Various semantics have been developed; see (Charwat et al. 2015) for detailed overviews. In this work, we consider the well-known grounded semantics whose extension can be constructed as $\texttt{gr} = \bigcup_{i \geq 0} G_i$, where G_0 is the set of all unattacked arguments, and $\forall i \geq 0$, $G_{i+1} = G_i \cup \{a \mid a \text{ defended by } G_i\}$. For any $\langle \mathcal{A}, \mathcal{R} \rangle$, the grounded extension `gr` always exists and is unique.

Probabilistic Frameworks and Constellation. There exist different ways to extend the classic AF with probability into a probabilistic argumentation framework (PrAF). For example, we can label arguments and/or attacks with a probability. In (Mantadelis and Bistarelli 2020), the authors proposed a way to transform any PrAF having probability on arguments and attacks to a PrAF with probability only on attacks (or only on arguments) using a normal form of probabilistic attacks (or normal form of probabilistic arguments). They showed that all these forms are equivalent: they obtain the same probabilistic distribution on their extensions. As illustrated in Figure 1, the transformation consists of adding a new argument n that attacks all the arguments so that the probability of the attack is equal to 1 - the previous probability of the argument. In the following, we consider PrAFs with only probability values on attacks.

Definition 1. A probabilistic argumentation framework (**PrAF**) is a tuple $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ such that: $\mathcal{A} \subseteq_f \texttt{Arg}$[3], $\mathcal{R} \subseteq_f \mathcal{A} \times \mathcal{A}$, $P_R : \mathcal{R} \rightarrow]0, 1]$.

Now, let us look at the computation of the probability of arguments using the Constellation method (Hunter 2012).

[2] The proposed algorithms focus on the grounded semantics, but all Dung's semantics coincide on DAGs/SCGs (Dung 1995).

[3] $\mathcal{A} \subseteq_f \texttt{Arg}$ stands for: $\mathcal{A}$ is a finite subset of all arguments.

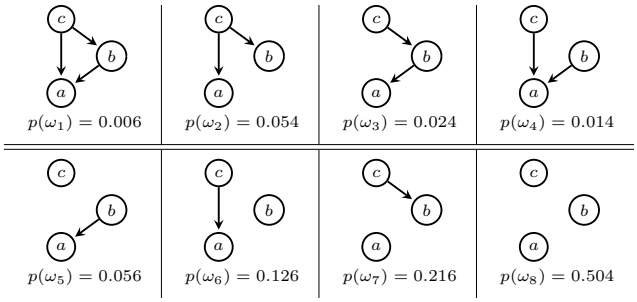

Figure 2: Graph constellation of $\mathbf{G}_1$.

We call a "graph constellation" the set of all possible worlds; the probability of a world is calculated by multiplying the probabilities of present relations and the inverse probabilities of absent relations.

Definition 2. Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be a PrAF and $\omega = \langle \mathcal{A}', \mathcal{R}' \rangle$ be a world denoted by $\omega \sqsubseteq \mathbf{G}$ iff $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{R}' \subseteq (\mathcal{A}' \times \mathcal{A}') \cap \mathcal{R}$. The **probability of the world** ω, is denoted by $p(\omega) = \Big(\prod_{att \in \mathcal{R}'} P_R(att) \Big) \times \Big(\prod_{att \in \mathcal{R} \setminus \mathcal{R}'} \big(1 - P_R(att)\big) \Big)$.

Example 1. The constellation of $\mathbf{G}_1 = \langle \{a, b, c\}, \{(c, b), (c, a), (b, a)\}, P_R \rangle$ s. t. $P_R((c, b)) = 0.3$, $P_R((c, a)) = 0.2$, and $P_R((b, a)) = 0.1$, with the probability of each world is represented in Figure 2.

The sum of the probability of all worlds is equal to 1 (Hunter 2013), i.e., for any PrAF $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$, $\sum_{\omega \sqsubseteq \mathbf{G}} p(\omega) = 1$. The Constellation method computes the probability of acceptance of a set of arguments by examining the acceptance of these arguments in each world.

Definition 3. Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be a PrAF, for any $X \subseteq \mathcal{A}$ and $\mathcal{S}$ an extension-based semantics, we denote the **probability of skeptical acceptance** by $P^{\mathcal{S}}(X, \mathbf{G}) = \sum_{\omega \sqsubseteq \mathbf{G}} p(\omega) \times \mathtt{In}^{\mathcal{S}}(\omega, X)$, where $\mathtt{In}^{\mathcal{S}}(\omega, X) = 1$ if X is a subset of each extension of $\mathcal{S}$ in ω, otherwise equal to 0. In the following, we will omit the graph parameter when it is clear, i.e., we will use $P^{\mathcal{S}}(X)$.

If we redefine $\mathtt{In}^{\mathcal{S}}$ as "if X is a subset of at least one extension of $\mathcal{S}$", the probability of credulous acceptance is computed. Here, we focus on the skeptical case.

Moreover, this paper focuses only on extension-based semantics and trees or acyclic graphs. It is important to recall that on DAGs, all the Dung semantics return the same extension, i.e., the grounded, complete, preferred, stable (see Theorem 30 in (Dung 1995)) and semi-stable (see Theorem 3 in (Caminada 2006)). Therefore, in the rest of the paper, we will denote by $P(x)$ the probability of acceptance of an argument x for all the Dung semantics.

3 Linearithmic Computation in SCG PrAFs

From the title of this section onwards, we call an "SCG PrAF" any PrAF that satisfies the topology of an SCG.

In (Bistarelli et al. 2022), a function is proposed to compute the probability of an argument under grounded semantics for *uncontroversial* acyclic PrAFs ((Dung 1995), i.e., arguments cannot attack and defend the same argument).

Definition 4. For any PrAF $\langle \mathcal{A}, \mathcal{R}, P_R \rangle$, $\mathtt{Fast}^{\mathtt{gr}} : \mathcal{A} \rightarrow [0, 1]$ computes the probability of any argument to be accepted w.r.t. the grounded semantics, such that $\mathtt{Fast}^{\mathtt{gr}}(a) =$

$$\begin{cases} 1 & \text{if } \mathtt{Att}^{-}(a) = \emptyset \\ \prod\limits_{b \in \mathtt{Att}^{-}(a)} 1 - \big(\mathtt{Fast}^{\mathtt{gr}}(b) \times P_R(b, a)\big) & \text{otherwise} \end{cases}$$

An argument is in the grounded extension if it is defended. Trivially, an unattacked argument is always defended, so its acceptance probability is 1. For an argument a attacked by b, $\mathtt{Fast}^{\mathtt{gr}}(b) \times P_R(b, a)$ computes the joint probability of b being acceptable and attack (b, a) existing. Thus, $1 - \mathtt{Fast}^{\mathtt{gr}}(b) \times P_R(b, a)$ gives the probability that either b is unacceptable or attack (b, a) does not exist. Finally, the product of this computation for each attack ensures that a is defended, i.e., none of its attackers actually attacks a.

Example 2. The skeptical acceptance probability of the arguments of $\mathbf{G}_1$ in the grounded semantics are: $P(a) = \mathtt{Fast}^{\mathtt{gr}}(a) = 0.024 + 0.216 + 0.504 = 0.744$, $P(b) = \mathtt{Fast}^{\mathtt{gr}}(b) = 0.014 + 0.056 + 0.126 + 0.504 = 0.7$, and $P(c) = \mathtt{Fast}^{\mathtt{gr}}(c) = 1$.

However, an orderly random single application of the formula to each argument does not ensure that the computed value equals the probability of an argument. That is why we present the new optimized algorithm `Fast_SCG`, relying on a specific ordering of computations to ensure its correctness.

Fast SCG algorithms in detail. `Fast_SCG` computes the probability of an input argument a in a PrAF with the steps presented in Algorithm 1 and Algorithm 2.

Computation order (l. 1 / Algorithm 2). `Fast_SCG` propagates the probabilities from the roots (i.e., unattacked arguments) to the computed argument a. We use the `order` function to determine the order of the computation. This function traverses the graph backward from a to the roots and saves the maximum distance of each reachable argument to a in the $dico\_max\_dist$ variable. This corresponds to finding the longest path in a directed acyclic graph and is linear in the number of edges (Eppstein 1998). The `order` function also sorts the $dico\_max\_dist$ variable to order the arguments in decreasing maximum distance w.r.t. a, with a worst-case complexity in $\mathcal{O}(n\,log(n))$, where n is the number of arguments (e.g., Heapsort (Schaffer and Sedgewick 1993)).

Probability propagation (l. 2–7). We iteratively consider arguments from roots to a by browsing the ordered $dico\_max\_dist$ (l. 3), thus propagating probabilities in the correct order. According to Definition 4, the probability that an argument is acceptable is equal to the conjunction of the probability that its attackers will not attack it, i.e., the probability of the complement of the edge from the attacker to exist and the attacker to be acceptable (l. 6).

We show next that `Fast_SCG` characterizes P for any SCG PrAF and not any uncontroversial acyclic PrAF as discussed in (Bistarelli et al. 2022).

Theorem 1 (`Fast_SCG` correctness)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an SCG PrAF, then $\forall a \in \mathcal{A}$, $\mathtt{Fast\_SCG}(a) = P(a)$.

We show further that $\mathtt{Fast\_SCG}(a)$ is linearithmic in the number n of all direct and indirect attackers of an argument a belonging to a PrAF $\mathbf{G}$.

Algorithm 1: $\texttt{Fast\_SCG}(a, PrAF)$

$dico\_max\_dist = \texttt{order}(a, PrAF)$;
$prob = \{\}$;
for $k \in dico\_max\_dist$ **do**
    $prob[k] = 1$;
    for $att \in \texttt{Att}^-(k)$ **do**
        $prob[k] = prob[k] \times (1 - prob[att] \times P_R(att, k))$;
return $prob[a]$;

Algorithm 2: $\texttt{order}(a, PrAF)$

for $k \in \mathcal{A}$ **do**
    $dico\_lvl[k] = 0$;
$queue = \{a\}$;
while $queue \neq \emptyset$ **do**
    $queue2 = \{\}$;
    for $k \in queue$ **do**
        for $att \in \texttt{Att}^-(k)$ **do**
            if $dico\_lvl[att] \leq dico\_lvl[k]$ **then**
                $dico\_lvl[att] = dico\_lvl[k] + 1$;
                $queue2.add(att)$;
    $queue = queue2$;
$dico\_max\_dist = \texttt{heapSort}(dico\_lvl)$;
return $dico\_max\_dist$;

Theorem 2 (Complexity of Fast_SCG)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an SCG PrAF, $\forall a \in \mathcal{A}$, the time complexity of $\texttt{Fast\_SCG}(a)$ is linearithmic in the number n of all direct and indirect attackers of a, i.e., $\mathcal{O}(n \log n)$.

DAG PrAFs are any acyclic PrAFs. Especially, contrary to SCG PrAFs, DAG PrAFs may contain *dependent arguments and attacks*, that are formally defined below.

Definition 5. Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be a PrAF and $a, b \in \mathcal{A}$. We say that b is a **dependent argument** of a, denoted by $\texttt{dep}(a, b)$, iff at least two direct outgoing attacks from b have a path to a. The set of dependent arguments of an argument a in a PrAF $\mathbf{G}$ is denoted by $\texttt{Dep}(a, \mathbf{G}) = \{b \in \mathcal{A} \mid \texttt{dep}(a, b)\}$.

Direct outgoing attacks from b that reach a (from a path) are called **dependent attacks**. The set of dependent attacks of an argument b, on an argument a in a PrAF $\mathbf{G}$, is denoted by $\texttt{DepAtt}(b, a, \mathbf{G}) = \{(b, i) \in \mathcal{R} \mid \exists$ a path from b to a through $i\}$.

Example 3. Consider the DAG PrAF depicted in Figure 3. Argument a depends on argument c as two outgoing attacks from c reach a. Note that this sub-structure between a and c is the minimal structure to create a dependent argument. Argument a also depends on arguments d and f. Hence, $\texttt{Dep}(a, \mathbf{G}_2) = \{c, d, f\}$. Moreover, $\texttt{DepAtt}(c, a, \mathbf{G}_2) = \{(c, a), (c, b)\}$, $\texttt{DepAtt}(d, a, \mathbf{G}_2) = \{(d, c), (d, b)\}$ and $\texttt{DepAtt}(f, a, \mathbf{G}_2) = \{(f, e), (f, d)\}$. Note that (f, g) is not a dependent attack, given that it doesn't reach a.

Figure 1 shows how to transform a PrAF with probabilities on arguments and attacks to probabilities on attacks only by adding an argument that attacks all arguments. This implies that this argument has several paths with the other arguments, which no longer satisfy the definition of an SCG (but that of a DAG). However, in Theorem 3, we show that the Fast_SCG algorithm also works for these transformed PrAF SCGs (from fully weighted to semi-weighted), which are special cases of DAGs such that their dependent arguments are not attacked.

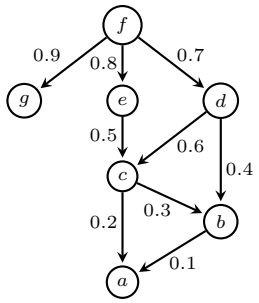

Figure 3: Acyclic Multiply-Connected Graph PrAF $\mathbf{G}_2$.

Theorem 3 (Fast_SCG correctness on transformed SCGs)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an DAG PrAF. If $a \in \mathcal{A}$ s.t. $\forall b \in \texttt{Dep}(a, \mathbf{G})$, $\texttt{Att}^-(b) = \emptyset$, then $\texttt{Fast\_SCG}(a) = P(a)$.

4 Fast Computation in Acyclic PrAFs

In general, applying Fast_SCG to DAG PrAFs would lead to values that differ from the actual probabilities computed by Constellation. Intuitively, among the differences, the Fast_SCG value of an argument a would involve terms in which the probability of each of its dependent arguments d is elevated at most to the power of the number of dependent attacks of d to a (i.e., $|\texttt{DepAtt}(d, a, \mathbf{G})|$). For example, consider only the sub-structure formed by arguments a, b, c, and d and the two paths from d to a removing edge (c, b) in Figure 3. Given that d has two dependent attacks (i.e., $\{(d, b), (d, c)\}$), we will get at most a power of 2 on the symbolic variable[4] of d. Hence, Fast_SCG would compute prob[a] $= (1 - P_R(c, a)(1 - P_R(d, c)\text{prob[d]}))) \times (1 - P_R(b, a)(1 - P_R(d, b)\text{prob[d]})$, leading to a term with prob[d]^2. In probability, it can be shown that, $P(a) = P([\overline{(b, a)} \cup \overline{b}] \bigcap [\overline{(c, a)} \cup \overline{c}]) = P([\overline{(b, a)} \cup ((d, b) \cap d)] \bigcap [\overline{(c, a)} \cup ((d, c) \cap d)])$. Developing the expression, we obtain a term $(d, b) \cap d \cap (d, c) \cap d$ where the idempotence of $\cap$ leads to $(d, b) \cap (d, c) \cap d$, i.e., no $P(d)^2$ appears in the final expression. Thus, $\texttt{Fast\_SCG}(a)$ and $P(a)$ differ when $P(d) \neq 1$, i.e., when d is attacked.

That is why we propose the Fast_DAG algorithm that computes the probability of arguments in acyclic PrAFs. Specifically, our algorithm defines a new multiplicative operator $\otimes$ that is idempotent for the probability of dependent arguments as the $\cap$ operator in the realm of probabilities. That is, for a dependent argument k, $P(k) \otimes P(k) = P(k)$. To do so, we compute the symbolic expressions of probabilities that involve dependent arguments, remove powers, and reduce the expressions iteratively for each dependent argu-

[4] We refer to a symbolic variable to clarify that the power is not applied to the value of the variable.

ment before computing the final value. Our Python implementation[5] leverages the SageMath library (The Sage Developers 2024) for symbolic computations.

Fast DAG algorithms in details. `Fast_DAG` computes the probability of an input argument a in a PrAF with the steps presented in Algorithm 3 and Algorithm 4. The algorithm and its different steps are illustrated in Example 4.

Computation order (l. 1). Similarly to `Fast_SCG`, `Fast_DAG` propagates probabilities from the roots to the computed argument a, and accordingly, we use the `order` function to determine the order of computation.

Determining dependent arguments (l. 2 and Algorithm 4). We determine the set of dependent arguments of a, since they require symbolic computations to remove powers before computing the actual values. To do so, we compute $back\_att$, the set of reachable arguments by traversing the graph backward from a to the roots, including a. Dependent arguments (Dep) are arguments in $back\_att$ that have at least 2 out-going attacks reaching a, which is computed by intersecting the set of arguments they directly attack ($\texttt{Att}^+(b)$) with the set of arguments reachable backward from a ($back\_att$). Note that we exclude unattacked arguments ($|\texttt{Att}^-(b)| > 0$) as their values is always 1, which does not require symbolic computations.

Probability propagation (l. 3–11). We iteratively consider arguments from roots to a by browsing the ordered $dico\_max\_dist$ (l. 4). Similarly to `Fast_SCG`, the probability of an argument to be acceptable is equal to the conjunction of the probability of its attackers not to be acceptable, i.e., the probability of the complement of the edge from the attacker to exist and the attacker being acceptable (l. 8 and l. 10). The main difference is as follows: if an attacker is a dependent argument, we use its symbol (l. 8) instead of its probability (l. 10). This leads to probabilities of arguments to be symbolic expressions containing both symbols possibly and floats instead of an actual value. The final probability is calculated directly when attackers are neither dependent arguments nor symbolic expressions. When using $prob[att]$ (l. 10), such probabilities and symbolic expressions propagate along the computations. Hence, at the end of the for loop (l. 6), probabilities of arguments in $prob$ can be either actual floats or symbolic expressions. Note that, when an attacker is a dependent argument, its symbol is used, potentially masking its own symbolic expression (see computation of b^* in Example 4, where symbol c is used, masking the symbolic expression c^* that depends on d).

Handling dependent arguments (l. 12–16). If a has no dependent argument, then no symbolic expressions are involved. Its probability has thus already been computed and can be returned directly (l. 18). Otherwise ($|Dep| > 0$, l. 12), symbolic expressions need to be handled. To do so, `Fast_DAG` progressively expands the symbolic expression of a, with expressions of dependent arguments from closest to furthest (l. 13, reversing $dico\_max\_dist$). As mentioned above, this is because close dependent arguments are masking symbolic expressions coming from further parts of the DAG PrAF. For each dependent argument k, we first expand the symbolic expression of a, remove the potential powers on k, reduce the obtained expressions by grouping similar terms together, and finally replace k with its symbolic expression or value. This removal of powers corresponds to using a multiplicative operator $\otimes$ that is idempotent w.r.t. the probability of each dependent argument, i.e., $P(k) \otimes P(k) = P(k)$.

[5] https://github.com/Vict0r-David/Fast_DAG

Algorithm 3: `Fast_DAG`$(a, PrAF)$

```
1:  dico_max_dist = order(a, PrAF) ;
2:  Dep = dependant_arg(a, PrAF);
3:  prob = {} ;
4:  for k ∈ dico_max_dist do
5:      k* = 1;
6:      for att ∈ Att⁻(k) do
7:          if att ∈ Dep then
8:              k* = k* × (1 − symbol(att) × P_R(att, k)) ;
9:          else
10:             k* = k* × (1 − prob[att] × P_R(att, k));
11:     prob[k] = k* ;
12: if |Dep| > 0 then
13:     for k ∈ dico_max_dist⁻¹ do
14:         if k ∈ Dep then
15:             prob[a].expand&deletePower();
16:             prob[a].reduce&replace(k, prob[k]);
17: return prob[a];
```

Algorithm 4: `dependant_arg`$(a, PrAF)$

```
1: back_att = backwards(a, PrAF);
2: Dep = set();
3: for b ∈ back_att do
4:     if |Att⁺(b) ∩ back_att| > 1 and |Att⁻(b)| > 0 then
5:         Dep.add(b) ;
6: return Dep;
```

Example 4. We compute the probability of a in the DAG PrAF $\mathbf{G}_2$ (Figure 3) step by step.

$\texttt{order}(a, \mathbf{G}_2) = \{f : 4, e : 3, d : 3, c : 2, b : 1, a : 0\}$

$\texttt{dependant\_arg}(a, \mathbf{G}_2) = \{c, d\}$

Note that g is not in $\texttt{order}(a, \mathbf{G}_2)$ because it cannot be reached backward, and f, which is a dependent argument, is not considered in $\texttt{dependant\_arg}(a, \mathbf{G}_2)$ since we ignore dependent arguments that are not attacked.

We calculate (l. 4–11) the expression of each argument in $\texttt{order}(a, \mathbf{G}_2)$:[6]

$f^* = 1; \quad e^* = 0.2; \quad d^* = 0.3;$

$c^* = (1 - 0.5(0.2)) \times (1 - 0.6d) = 0.9 \times (1 - 0.6d);$

$b^* = (1 - 0.3c) \times (1 - 0.4d);$

$a^* = (1 - 0.2c) \times (1 - 0.1(1 - 0.4d)(1 - 0.3c));$

Then (l. 13–16), we develop the dependent arguments from nearest to furthest from a. Therefore, for c then d, `Fast_DAG` expands the expression of a^*, deletes the powers of the considered dependent argument, and replaces the symbolic variable with its expression. Starting with c:

[6] To ease notations, we use k^* to denote prob[k]

$a^* = 0.9 - 0.15c - 0.006c^2 + 0.04d - 0.02cd + 0.0024c^2d$
$a^* = 0.9 - 0.156c + 0.04d - 0.0176cd$
$a^* = 0.9 - 0.156(0.9(1 - 0.6d)) + 0.04d$
$\quad - 0.0176(0.9(1 - 0.6d))d$
Continuing with d: $a^* = 0.7596 + 0.1084d + 0.009504d^2$
$a^* = 0.7596 + 0.1084d + 0.009504d = 0.7949712$.

In the case of SCG PrAFs, we show that `Fast_DAG` can also compute the exact solution in linearithmic time, according to the number of arguments due to its ordering (as explained in Section 3) and linear for the computation of the probability according to the number of attackers.

Theorem 4 (Complexity of `Fast_DAG` in SCGs)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an SCG PrAF. For any argument $a \in \mathcal{A}$, the upper bound of $\texttt{Fast\_DAG}(a)$ is linearithmic in the number n of all direct and indirect attackers of a, i.e., $\mathcal{O}(n \log n)$.

In Figure 4, we illustrate the linearithmic complexity of `Fast_DAG` (see in red the linearithmic growth coefficients, i.e., $\frac{n \log n}{(n-1)\log(n-1)}$) on 16 classes of 100 SCGs. A class groups 100 SCGs with the same number of attacks, ranging from 500 to 2000 attacks. Note that for SGCs, the number of arguments equals the number of attacks+1 (due to the relevant SCG structure obtained by the `order` function). All the experiments in this paper were performed on an M2 Pro CPU - 2.42 GHz / 3.5 GHz with 32 GB RAM.

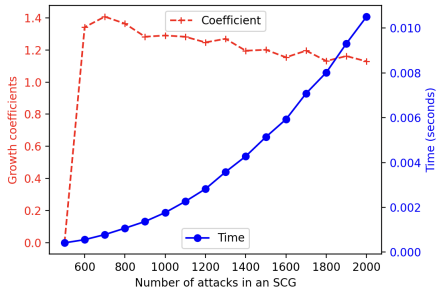

Figure 4: `Fast_DAG` average runtime for an argument by class of SCGs, and coefficients of growth between two consecutive classes.

In Theorem 5, we show that `Fast_DAG` characterizes the Constellation approach in the case of DAG PrAFs.

Theorem 5 (`Fast_DAG` correctness)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an acyclic PrAF, then $\forall a \in \mathcal{A}$, $\texttt{Fast\_DAG}(a) = P(a)$.

Intuitively, the correctness stems from the multiplicative operator $\otimes$ that is idempotent for the probability of dependent arguments. This corresponds to the idempotence of the intersection of an event with itself in the realm of probabilities.

Now, we theoretically and experimentally investigate the complexity of `Fast_DAG`. In the worst case, the complexity of `Fast_DAG` resides in the first expansion (l. 15 of Algorithm 3), where the biggest polynomials are present regarding the number of variables. These polynomials are multi-variate, with $|\texttt{Dep}(a, \mathbf{G})|$ variables (i.e., one variable per dependent argument) where each dependant argument can be elevated at most to the power of its number of outgoing attacks reaching a (i.e., $|\texttt{DepAtt}(a, \mathbf{G})|$). The expansion of such a polynomial multiplication has been shown (Frigo and Johnson 2005) to be of the same complexity as the multi-dimensional Fast Fourier Transform, which is $\mathcal{O}\Big(\prod_v \texttt{MaxP}(v) \log \prod_v \texttt{MaxP}(v)\Big)$, when the symbolic variable v of each dimension can be elevated at most to the power $\texttt{MaxP}(v)$.

From this complexity, we now show the upper bound of `Fast_DAG` according to the worst expansion case.

Theorem 6 (Upper bound of `Fast_DAG`)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an DAG PrAF, and $a \in \mathcal{A}$ be an argument. The upper bound of $\texttt{Fast\_DAG}(a, \mathbf{G})$ is defined by:
$\mathcal{O}(\prod_{d \in \texttt{Dep}(a)} x_d \log \prod_{d \in \texttt{Dep}(a)} x_d)$, s.t. $x_d = |\texttt{DepAtt}(d, a)| + 1$.

Theorem 7 shows that for any PrAF DAG, the `Fast_DAG` algorithm has a better time complexity than Constellation since the $\mathcal{O}$ of the former is smaller than the Ω of the latter, which is equal to $2^{|\mathcal{R}|}$ (i.e., the extension-based semantics is considered in $\Omega(1)$, which is a strong hypothesis).

Theorem 7 (`Fast_DAG` vs Constellation)**.** Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be an DAG PrAF, hence, for any $a \in \mathcal{A}$,

$$\mathcal{O}(\texttt{Fast\_DAG}(a, \mathbf{G})) < \Omega(P(a, \mathbf{G})).$$

Experimental setting. The `Fast_DAG` algorithm has been studied on 4,000 randomly generated DAGs containing 50 arguments and 75 attacks randomly distributed. This setting maximizes the number of arguments computed in an acceptable time. Random DAGs have a typology following Erdős–Rényi graphs (Erdös and Rényi 1959): each pair of arguments has the same probability of having an attack. Then we randomly select the desired number of attacks.

In these DAGs, arguments have a standard deviation of 2.1 for outgoing attacks, 1.4 for incoming attacks and for both an average of 1.5. Furthermore, on average per graph, an argument has 4.2 dependent arguments (standard deviation of 3.8), such that these dependent arguments have, on average, a minimum of 3.3 and a maximum of 6.8 outgoing dependent attacks (with both a standard deviation of 1.4), 16.73 arguments (max: 33) accessible backward, in trees with an average depth of 4.67 (max: 11) and an average width of 6.22 (max: 15). Large trees are also deep: some have 31 arguments, width = 15, and depth = 11. This shows that we generated a large variety of structures, avoiding only lines or sparse graphs.

Fast DAG vs Constellation. 99.84% of arguments are computed in less than 1 second. The worst case is around 10,000 seconds ($<$ 3 hours). On average, computing all arguments of a graph takes 10 seconds. In comparison, the Constellation method, with exponential complexity in the number of edges, was tested on graphs from 2 to 31 edges. The results show that execution time doubles with each additional edge, with graphs of 30 edges taking approximately 8,000 seconds. Extrapolating, a graph with 75 edges would require around 3×10^{17} seconds ($\sim$ 9 billion years).

Runtime analysis. The runtime of an argument a in $\mathbf{G}$ by `Fast_DAG` can be approximated (as we show experimentally in Figure 5) with the maximal number of terms `maxT` to deal

with, among all the expand operations (l. 15 of Algorithm 3) of dependent arguments. We define $\texttt{maxT}(a, \mathbf{G}) =$

$$\max_{d \in \texttt{Dep}(a,\mathbf{G})} \prod_{v \in \texttt{Symb}(a,expr,Dep,\mathbf{G})[d]} |\texttt{DepAtt}(v, a, \mathbf{G})| + 1$$

where $\texttt{Symb}(a, expr, Dep, \mathbf{G})[d]$, computed by Algorithm 5, is the set of symbolic variables present in the expand (l. 15 of Fast_DAG) for dependent argument d. The parameter $expr$ is the (potentially symbolic) probability expression before the resolution of dependent arguments (l. 12), and Dep is the output of $\texttt{dependant\_arg}(a, PrAF)$ (l. 2). The method $X.symbs()$ (l. 2 and l. 5 in Algorithm 5) returns the set of symbolic variables in the expression X.

Intuitively, each expansion's terms corresponds to the maximum number of terms in a multi-variate polynomial, where the (symbolic) variables are the "visible" dependent arguments at this step. Each dependent argument (i.e., variable) can be elevated to the power of at most its number of out-going attacks reaching a (i.e., $|\texttt{DepAtt}(a, \mathbf{G})|$). The "visibility" is because dependent arguments mask symbolic expressions (see computation of b^* in Example 4, where symbol c is used, masking its own symbolic expression that depends on d).

Figure 5 shows that the runtime increases linearithmically according to this maximum number of terms, i.e., experimentally we estimate the runtime of $\texttt{Fast\_DAG}(a, \mathbf{G})$ by:

$$\texttt{maxT}(a, \mathbf{G}) \log \texttt{maxT}(a, \mathbf{G}).$$

Note that this approximation is close to $\mathcal{O}$, with the difference that it does not take into account all the dependent arguments, but the maximum number of visible dependent arguments when expanding. Therefore, in practice, even if there is a huge overall number of dependent arguments, if they are all nested (e.g., in Example 4 if f were attacked, f would be a symbolic variable nested in c and d), the number of dependent arguments visible at each expansion will be low, and thus, the complexity will also be low.

Algorithm 5: $\texttt{Symb}(a, expr, Dep, PrAF)$

1: $dico\_max\_dist = \texttt{order}(a, PrAF)$;
2: $dico\_symb = \{\}$; $S = expr[a].symbs()$;
3: **for** $d \in dico\_max\_dist^{-1}$ **do**
4: **if** $d \in Dep$ **then**
5: $S.\texttt{del}(d)$; $S.\texttt{add}(expr[d].symbs())$;
6: $dico\_symb[d] = S$;
7: **return** $dico\_symb$;

In Figure 5, in addition to the cases predicted by the previous formula (linearithmic according to maxT), we also observe several computations with large maxTs but low runtimes. These results highlight optimizations in SageMath.

Fast DAG vs Monte-Carlo. Besides the exact Constellation method, there exists in the literature an approximate method using the Monte-Carlo concept (Li, Oren, and Norman 2011), which improves its approximation over time. We compare Fast_DAG with the results of this Monte-Carlo method with a time budget equal to the time taken by Fast_DAG. In Figure 6, we test on 1,000 graphs (i.e., 50,000 arguments) randomly selected from the previous 4,000 DAGs. Results show that the time needed by Fast_DAG to compute the probability of arguments is not enough for Monte-Carlo to approximate correctly. Furthermore, we can see that around 25% of the approximations (12,300) did not have time to run because Fast_DAG was too fast. Fast_DAG computes an argument's probability based on a subset of the graph (i.e., attackers in the backward direction as defined in order). If the sub-graph is small, a single Monte-Carlo iteration in a random world may not be feasible.

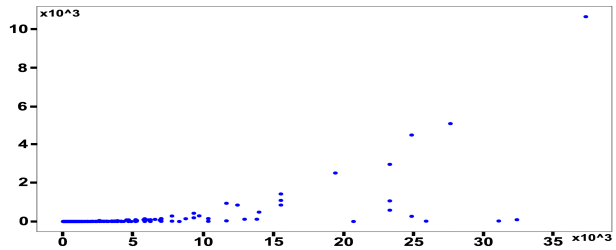

Figure 5: Time in seconds taken by Fast_DAG (y-axis) to compute arguments depending on the max number of terms (i.e., maxT) in their expression (x-axis). Values are of the order of 10^3.

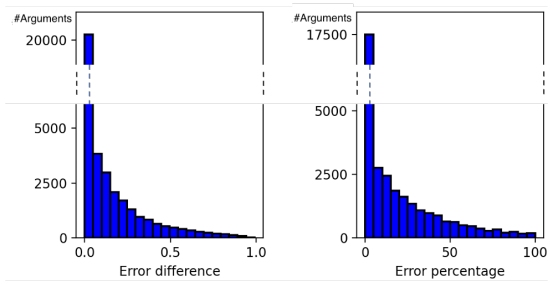

Figure 6: Histogram of errors of Monte-Carlo on Fast_DAG runtime. Average difference = 0.15; average error percentage = 38%.

5 Conclusion

Despite its scientific success and practical applications, the Constellation approach (Li, Oren, and Norman 2011) suffers from high complexity due to the exponential number of generated worlds. To address this, we propose to compute the acceptance probability of an argument with two new algorithms (Fast_DAG and Fast_SCG), which are able to give the same score as the Constellation method. In DAGs, the time of our algorithm is related to the structure of the graph. We are not exponential in the number of attacks (as it is for the Constellation method) but linearitmic in the product of the out-degree of dependent arguments, which we prove to be much better in theory and practice. Furthermore, for the specific cases of SCG/Tree PrAFs, we show that our computation time is only linearithmic in the number of attackers. Future work will investigate how to extend Fast_DAG to cyclic PrAF and compute a set of arguments.

Acknowledgments

This work was supported by the French government, managed by the Agence Nationale de la Recherche under the Plan d'Investissement France 2030, as part of the Initiative d'Excellence d'Université Côte d'Azur under the reference ANR-15-IDEX-01. The work was supported by European Union - Next Generation EU PNRR MUR PRIN - Project J53D23007220006 EPICA: "Empowering Public Interest Communication with Argumentation".

References

Amgoud, L.; and David, V. 2018. Measuring Similarity between Logical Arguments. In *Proc. of KR*, 98–107.

Amgoud, L.; and David, V. 2021. A General Setting for Gradual Semantics Dealing with Similarity. In *Proc. of AAAI*.

Amgoud, L.; Parsons, S.; and Maudet, N. 2000. Arguments, Dialogue, and Negotiation. In *ECAI*, 338–342.

Bistarelli, S.; David, V.; Santini, F.; and Taticchi, C. 2022. Computing Grounded Semantics of Uncontroversial Acyclic Constellation Probabilistic Argumentation in Linear Time.

Bistarelli, S.; Rossi, F.; and Santini, F. 2018. A novel weighted defence and its relaxation in abstract argumentation. *Int. J. Approx. Reason.*, 92: 66–86.

Bolton, E.; Calderwood, A.; Christensen, N.; Kafrouni, J.; and Drori, I. 2020. High Quality Real-Time Structured Debate Generation. *CoRR*, abs/2012.00209.

Caminada, M. 2006. Semi-Stable Semantics. In *Proceedings of the 1st International Conference on Computational Models of Argument, COMMA'06*, 121–130.

Charwat, G.; Dvořák, W.; Gaggl, S. A.; Wallner, J. P.; and Woltran, S. 2015. Methods for solving reasoning problems in abstract argumentation–a survey. *Artificial intelligence*, 28–63.

Chow, C.; and Liu, C. 1968. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3): 462–467.

Cohen, A.; Gottifredi, S.; García, A. J.; and Simari, G. R. 2014. A survey of different approaches to support in argumentation systems. *The Knowledge Engineering Review*, 29(5): 513–550.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Non-Monotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77: 321–357.

Eppstein, D. 1998. Finding the k shortest paths. *SIAM Journal on computing*, 28(2): 652–673.

Erdös, P.; and Rényi, A. 1959. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6: 290.

Fazzinga, B.; Flesca, S.; and Furfaro, F. 2018. Credulous and skeptical acceptability in probabilistic abstract argumentation: complexity results. *Intelligenza Artificiale*, 181–191.

Fazzinga, B.; Flesca, S.; and Parisi, F. 2015. On the complexity of probabilistic abstract argumentation frameworks. *ACM Transactions on Computational Logic (TOCL)*, 16(3): 1–39.

Frigo, M.; and Johnson, S. G. 2005. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2): 216–231.

Henrion, M. 1988. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Machine intelligence and pattern recognition*, volume 5, 149–163. Elsevier.

Hunter, A. 2012. Some foundations for probabilistic abstract argumentation. *Comma*, 2012: 117–128.

Hunter, A. 2013. A probabilistic approach to modelling uncertain logical arguments. *International Journal of Approximate Reasoning*, 54(1): 47–81.

Kim, J.; and Pearl, J. 1983. A computational model for causal and diagnostic reasoning in inference systems. In *International Joint Conference on Artificial Intelligence*, 0–0.

Li, H.; Oren, N.; and Norman, T. J. 2011. Probabilistic argumentation frameworks. In *International Workshop on Theorie and Applications of Formal Argumentation*, 1–16. Springer.

Mantadelis, T.; and Bistarelli, S. 2020. Probabilistic abstract argumentation frameworks, a possible world view. *International Journal of Approximate Reasoning*, 119: 204–219.

Nofal, S.; Atkinson, K.; and Dunne, P. E. 2021. Computing grounded extensions of abstract argumentation frameworks. *The Computer Journal*, 64(1): 54–63.

Potyka, N. 2021. Interpreting neural networks as quantitative argumentation frameworks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 6463–6470.

Schaffer, R.; and Sedgewick, R. 1993. The analysis of heapsort. *Journal of Algorithms*, 15(1): 76–100.

Schindler, C. 2020. Argumentative relation classification for argumentative dialogue systems.

Sun, X.; and Liao, B. 2015. Probabilistic argumentation, a small step for uncertainty, a giant step for complexity. In *Multi-Agent Systems and Agreement Technologies*. Springer.

The Sage Developers. 2024. *SageMath, the Sage Mathematics Software System (Version 10.4)*. `https://www.sagemath.org`.

Thomas, C. S.; Howie, C. A.; and Smith, L. S. 2005. A New Singly Connected Network Classifier based on Mutual Information. *Intelligent Data Analysis*, 9(2): 189–205.

Toni, F.; Potyka, N.; Ulbricht, M.; and Totis, P. 2023. Understanding ProbLog as Probabilistic Argumentation. *arXiv preprint arXiv:2308.15891*.

Toni, F.; Rago, A.; and Čyras, K. 2023. Forecasting with jury-based probabilistic argumentation. *Journal of Applied Non-Classical Logics*, 33(3-4): 224–243.