Fast Computing of the Grounded Extension in Acyclic Probabilistic Argumentation Frameworks

Stefano Bistarelli, **Victor David**, **Francesco Santini**, **Carlo Taticchi**

Department of Mathematics and Computer Science University of Perugia, Italy

firstname.name@unipg.it

Abstract

We propose a faster algorithm to compute the (exact) probability of acceptance of an argument in the grounded extension when using the constellations approach. We show that for singly connected graphs, the problem can be solved in linearithmic time instead of exponential. Moreover, in the case of acyclic graphs, the problem is exponential in the number of dependent arguments instead of exponential in the number of attacks, thus significantly improving the performance in the practice. Our approach is then compared against i) computing the whole constellations (an exact method), and ii) a Monte Carlo approach (an approximate method).

1 Introduction

The pioneering article in the field of abstract argumentation comes from P.M. Dung (Dung 1995), who defined the notion of 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; Amgoud and David 2021) the addition of weights (Bistarelli, Rossi, and Santini 2018), or support relations (Amgoud and Ben-Naim 2018).

In this paper, we focus on probabilistic AFs (or *PrAF*) and more in particular on the *constellations* approach (Li, Oren, and Norman 2011): probability values determine the likelihood of both arguments and attacks to be part of an AF, thus generating different frameworks with a different existence probability. Hence, the uncertainty resides in the topology of the considered PrAF.[1] Thus, when we want to study the acceptance probability of an argument, we need to compute this value in all the possible AFs induced from a PrAF, and then aggregate. However, the number of induced AFs is in general exponential (Li, Oren, and Norman 2011), consequently reducing the research attractiveness in this field for practical reasons. This paper is the first step to showing that it is possible to optimize the computation of the acceptance probability without having to build the entire constellations.

[1]The epistemic approach (Hunter 2013) uses instead probability theory to represent degrees of belief in arguments, given a fixed AF.

Citing some related work, in (Fazzinga, Flesca, and Parisi 2015), the complexity of computing the acceptance probability of an argument has been done for different semantics and the result for the grounded is $FP^{\#P}\text{-}complete$. Having such a high complexity, the work in (Sun and Liao 2015) proposed some restrictions on the value of the probability to improve the complexity. If the probability is binary 0 or 1, then the probability of acceptance is polynomial in time in the case of grounded semantics. If the probability is ternary 0, 0.5, or 1, then the acceptance probability with the grounded is P-hard. Finally, in (Nofal, Atkinson, and Dunne 2021) a new fast algorithm to compute the ground extension has been proposed for classic AF. It could be interesting to study how this algorithm can be extended to PrAF.

In this paper, the key idea is to compute the acceptance probability of an argument in the ground 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. We do it very efficiently for AFs that follow the topology of *Singly Connected Network*s (*SCN*s) or "polytrees" (Chow and Liu 1968; Kim and Pearl 1983; Henrion 1988; Thomas, Howie, and Smith 2005). In these (acyclic) graphs, a node may have several parents but at most one underlying path exists between any pair of nodes. In this case, the propagation procedure is linearithmic (*O(n) log(n)*) in the number of arguments. However, we can also apply propagation to *Multi Connected Networks* (*MCN*s) (Henrion 1988), i.e., acyclic graphs in general. Even if such an exact approach is exponential in the number of dependent arguments (when more than one path exists between a pair of nodes), on random graphs we show that it still performs much better than computing all the induced AFs, and, by stopping the computation at the same time, the Monte Carlo approach used in (Li, Oren, and Norman 2011) often returns a significant approximation error.

All supplementary materials (codes, data sets, proofs) are available at this link: https://github.com/Vict0r-David/UNIPG/tree/main/Proba_Arg.

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 attack relation. For $a, b \in$

$\mathcal{A}$, if $(a,b) \in \mathcal{R}$, then we say that a attacks b and that a is an attacker of b. If for $a \in \mathcal{A}$ there is no $b \in \mathcal{A}$ with $(b,a) \in \mathcal{R}$, then a is unattacked. 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. Let the set of attackers of a denoted by $\mathtt{Att}(a) = \{b \in \mathcal{A} \mid (b,a) \in \mathcal{R}\}$.

An AF provides means to represent conflicting information. Reasoning with that information is done by means of argumentation semantics. A semantics provides a characterization of acceptable arguments in an AF. A set of acceptable arguments according to a semantics is called an extension and is taken as a reasoning outcome. Many semantics have been proposed, see e.g. (Charwat et al. 2015) for overviews. In this work, we will consider the very well-established grounded semantics: the grounded extension of $\langle \mathcal{A}, \mathcal{R} \rangle$ can be constructed as $\mathtt{gr} = \bigcup_{i \geq 0} G_i$, where G_0 is the set of all unattacked arguments, and $\forall i \geq 0$, G_{i+1} is the set of all arguments that G_i defends. For any $\langle \mathcal{A}, \mathcal{R} \rangle$, the grounded extension $\mathtt{gr}$ always exists and is unique.

Probabilistic Frameworks and Constellations. 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 PrAF with probability only on attacks (or only on arguments) thanks to the probabilistic attack normal forms (or probabilistic argument normal form). They showed that all these forms are equivalent, i.e. same probabilistic distribution on their extensions.

Definition 1 (PrAF)**.** *A probabilistic argumentation framework (PrAF) is a tuple $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ such that: $\mathcal{A} \subseteq_f \mathtt{Arg}$ [2], $\mathcal{R} \subseteq_f \mathcal{A} \times \mathcal{A}$, $P_R : \mathcal{R} \rightarrow]0,1]$.*

We call SCN PrAF every PrAF respecting the constraints of a Singly Connected Network.

The constellation of a graph is composed of all its possible subgraphs (worlds), and we compute the probability of one subgraph as follows.

Definition 2 (Probability of a world)**.** *Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ and $\omega = \langle \mathcal{A}', \mathcal{R}', P_R \rangle$ be probabilistic argumentation graphs such that $\omega \sqsubseteq AF$ [3]. The probability of subgraph ω, denoted $p(\omega) = \Big(\prod_{att \in \mathcal{R}'} P_R(att)\Big) \times \Big(\prod_{att \in \mathcal{R} \setminus \mathcal{R}'} (1 - P_R(att))\Big)$*

Example 1. *Let see the SCN PrAF $\mathbf{G} = \langle \{a,b,c,d\}, \{(a,b),(c,d),(d,b)\}, P_R \rangle$ such that $P_R((a,b)) = 0.4$, $P_R((c,d)) = 0.7$, $P_R((d,b)) = 0.2$.*

Recall that it was shown in (Hunter 2013) that the sum of the probability of any subgraph is equal to 1. Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be a PrAF, then $\sum_{\omega \sqsubseteq AF} p(\omega) = 1$.

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

[3] The notation $\omega \sqsubseteq AF$ stands for: $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{R}' = \{(a,b) \in \mathcal{R} \mid a \in \mathcal{A}' \text{ and } b \in \mathcal{A}'\}$: ω is a subgraph of an AF.

Let us recall now how to compute the probability of an argument or a set of arguments belonging to the extensions of extension-based semantics.

Definition 3 (Acceptance Probability)**.** *Let $\mathbf{G} = \langle \mathcal{A}, \mathcal{R}, P_R \rangle$ be a PrAF, $X \subseteq \mathcal{A}$ and $\mathcal{S}$ an extension-based semantics, we denote by $P^{\mathcal{S}}(X) = \sum_{\omega \sqsubseteq AF} p(\omega) \times \mathtt{In}^{\mathcal{S}}(\omega, X)$, 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 a previous workshop paper (Bistarelli et al. 2022) we discussed the possibility of computing the probabilities based on the grounded semantics by using the next function.

Definition 4 ($\mathtt{Fast}^{\mathtt{gr}}$)**.** *Let $\mathtt{Fast}^{\mathtt{gr}}$ be the function from any PrAF $\langle \mathcal{A}, \mathcal{R}, P_R \rangle$ which computes the acceptance probability of any argument to be in the grounded extension ($\mathcal{A} \rightarrow [0,1]$), s.t. $\mathtt{Fast}^{\mathtt{gr}}(a) =$*

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

Example 1 (Continued). The acceptance probabilities of the arguments in the grounded semantics are: $P^{\mathtt{gr}}(a) = 1$, $P^{\mathtt{gr}}(b) = \mathtt{Fast}^{\mathtt{gr}}(b) = 0.084 + 0.336 + 0.144 = 0.564$, $P^{\mathtt{gr}}(c) = \mathtt{Fast}^{\mathtt{gr}}(c) = 1$, $P^{\mathtt{gr}}(d) = \mathtt{Fast}^{\mathtt{gr}}(d) = 0.024 + 0.096 + 0.036 + 0.144 = 0.3$.

3 Fast Computation in Acyclic PrAF

We propose a method for computing the probability of an argument in a PrAF to be accepted with respect to the grounded semantics. In this paper, we restrict to acyclic (MCN) PrAF. Our approach is based on the $\mathtt{Fast}^{\mathtt{gr}}$ function of Def. 4. Recall that an argument is in the grounded extension if it is defended. Trivially, an unattacked argument is always defended, so its probability of being accepted is 1. On the other hand, when an argument a is attacked by b, $\mathtt{Fast}^{\mathtt{gr}}(b) \times P_R(b,a)$ computes the joint probability that argument b is acceptable and the attack (b,a) exists. Thus $1 - \mathtt{Fast}^{\mathtt{gr}}(b) \times P_R(b,a)$ gives the probability that either argument b is not acceptable or attack (b,a) does not exist. Finally, the product of this computation for each attack ensures that a is defended.

In the following, we present the two algorithms that constitute the proposed method. For more details and optimizations, the program implemented in Python is available on GitHub[4] and to deal with the symbolic computation we used the symPy library (Meurer et al. 2017).

The function $\mathtt{Fast\_MCN}^{gr}$ of **Algorithm 1** takes as input an argument "goal" that we want to compute its probability and a PrAF, to give as output the final probability of the "goal". The idea of this first algorithm is to apply the right order to the second algorithm to calculate step by step the final probability of each attacker until the goal. We first use (line 1) the function `order` which traverses the graph back through the attackers from the "goal" to the roots. This function saves the maximum distance between each argument and the "goal" (find the longest path in a directed acyclic graph is linear in time (Eppstein 1998)). Then the `order`

[4] https://github.com/Vict0r-David/UNIPG/tree/main/Proba_Arg.

Algorithm 1 `Fast_MCN`^{gr}(goal, PrAF)

```
1:  goal_lvl = order(goal,PrAF);
2:  for arg ∈ goal_lvl do
3:      prob[arg], dep, arg_lvl = DevFast(arg,PrAF,prob);
4:      if len(dep) > 0 then
5:          arg_lvl.keepDep&Reverse(dep);
6:      for a ∈ arg_lvl do
7:          a* = 1;
8:          for b ∈ Att(a) do
9:              a* = a* × (1 - symb(b) × P_R(b,a)) ;
10:         prob[arg].subs(a,a*);
11:         prob[arg] = expand&deletePow();
12: return prob[goal];
```

Algorithm 2 `DevFast`(goal, PrAF, final_prob)

```
1:  goal_lvl = order(goal,PrAF) ;
2:  dep = dependant_arg(goal,PrAF);
3:  prob = {}, symbol = False ;
4:  for a ∈ goal_lvl do
5:      a* = 1;
6:      for b ∈ Att(a) do
7:          if b ∈ dep then
8:              a* = a* × (1 - symb(b) × P_R(b,a)) ;
9:              symbol = True;
10:         else if len(prob[b].vars) > 0 then
11:             a* = a* × (1 - prob[b] × P_R(b,a));
12:         else
13:             a* = a* × (1 - final_prob[b] × P_R(b,a));
14:     prob[a] = a* ;
15: if symbol then
16:     prob[a].expand&deletePow();
17: return prob[goal], dep, goal_lvl;
```

function applies a sorting algorithm (e.g., Timsort (Auger et al. 2018) or Heapsort (Schaffer and Sedgewick 1993) with a worst-case complexity in $O(nlogn)$ where n is the number of arguments) to obtain the arguments in the decreasing order with respect to the maximum distance of the "goal". Then (line 3) we apply for each argument the second algorithm `DevFast` which calculates the expanded expression of the argument according to its dependent arguments. If the argument has some dependency (line 4) we use the function `keepDep&Reverse` (line 5), deleting every non-dependent argument, then reverse the order (increasing way). Then for each dependent argument (line 6), we develop its expression (line 9). Finally, after the development, we substitute the symbol of the dependent argument (line 10) and solve the problem by expanding the expression followed by the deleting power (line 11).

In the **Algorithm 2** `DevFast`, the detection of the dependent arguments (line 2), is done by looking at all paths back from the goal argument and if an argument belongs to at least two paths it is dependent. Then we proceed to a symbolic development starting from the unattacked arguments towards the argument goal (thanks to the function `order` line 1). For this iterative process we will develop the probability of each argument by replacing its attacker according to 3 situations: i) (lines 7-9) the attacker is a dependent argument then we keep the symbol; ii) (lines 10-11) the attacker is not dependant to the "goal" but it has dependent attacker then we use its development iii) (lines 12-13) otherwise the attacker is neither a dependent argument nor have dependency hence we can replace its symbol by its final probability (computed at the previous steps). When we find dependent arguments (i.e. having conjunction with itself) we need to solve the development (line 16), by expanding the expression and then deleting each power (resulting from the product with itself) in the symbol of the dependent argument.

Note that when the context of the graph is clear, it will be omitted in the call of the algorithms. In the following theorem, we show that this algorithm characterizes the constellation approach in the case of acyclic PrAFs.

Theorem 1. *If* $\mathbf{G}$ *be an acyclic graph (MCN) then* $\forall a \in \mathcal{A}$*,* $P^{\text{gr}}(a) = \texttt{Fast\_MCN}^{\text{gr}}(a)$.

We next compare experimentally our algorithm `Fast_MCN`^{gr} with the constellation approach on 150 random acyclic graphs with 50 arguments and between 10 to 90 attacks (10 graphs by category, see Fig. 2).

First of all, we can see that the constellation approach is exponential in the number of attacks (around 20) regardless of the graph structure. As for the algorithm `Fast_MCN`$^{\text{gr}}$, we become exponential later because we depend on the structure of the graph and not on the attacks. These graphs are randomly generated, and in some cases we get a significant number of dependent arguments: for example, in a random AF with 50 arguments and 90 attacks, the max dependency for an argument is 11, and some of these arguments are dependent in turn.

Furthermore, since the computation time depends on the structure of the attackers, for all arguments without dependencies or close to the roots, the computation time is extremely fast. Thus, we can see that the average time on all arguments is very low. In order to have a better indication of the complexity of the problem we will discuss the max time.

4 Linearithmic Computation in SCN PrAF

In the case of acyclic graphs (MCN), to compute the probability of an argument we need to go through all its attackers and compute their probability by also going through all their attackers. This is why, in the case of SCN graphs, the acyclic Fast algorithm is polynomial (in the number of n attackers, i.e. $O(\frac{n(n-1)}{2})$) because without dependent argument there is no expansion and each computation is linear in time.

However, if we know that we will work on SCN graphs we can compute very quickly the exact probability of an argument, just by one path doing an ordering local propagation. We present next the new algorithm `SCN_Fast`$^{\text{gr}}$, which is able to compute the probability of an argument to be accepted in the grounded extension.

We show next that `Fast_SCN`$^{\text{gr}}$ algorithm characterizes P^{gr} for any SCN PrAF.

Theorem 2. *If* $\mathbf{G}$ *be a SCN PrAF then* $\forall a \in \mathcal{A}$*,* $P^{\text{gr}}(a) = \texttt{Fast\_SCN}^{\text{gr}}(a)$.

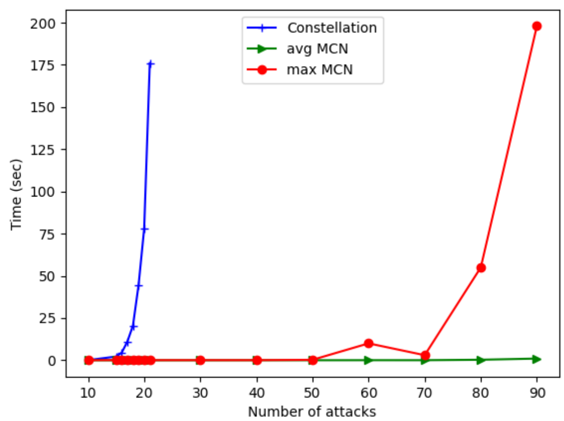

Figure 1: Comparison of run-times between the `Fast_MCN`^{gr} algorithm and the world's method (P^{gr}) in random acyclic graphs.

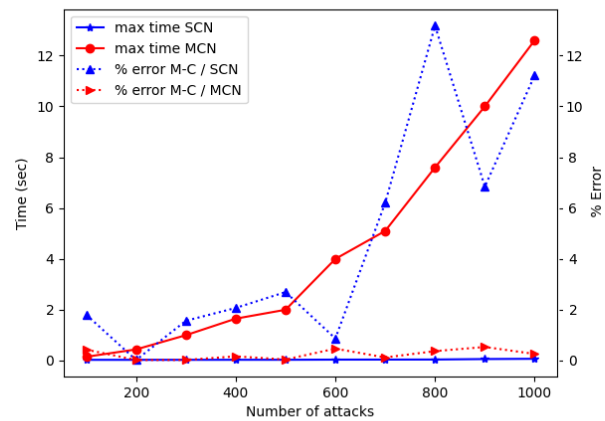

Figure 2: Run-times of `Fast_MCN`^{gr} vs `Fast_SCN`^{gr}, and % of errors of Monte-Carlo approach in these times (in random SCN graphs).

Algorithm 3 `Fast_SCN`^{gr}(goal, PrAF)

decreasing_lvl = `order`(goal,PrAF);
prob = {};
for a $\in$ decreasing_lvl **do**
  prob[a] = 1;
  for b $\in$ `Att`(a) **do**
    prob[a] = prob[a] $\times$ (1 - prob[b] $\times$ P_R(b,a));
return prob[goal];

We show further that this algorithm is linearithmic 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 3. *Let* $\mathbf{G}$ *be a SCN PrAF. For any argument* $a \in \mathcal{A}$*, the time complexity of* `Fast`$^{gr}(a)$ *is linearithmic on the number* n *of attackers (direct and indirect), i.e.* $O(nlogn)$*.*

Note that the intuition of `SCN_Fast`^{gr} cannot be used with MCN graphs in general, because in that case, we need to deal with dependent arguments. Indeed, if an argument attacks another one from several paths, it will be taken into account redundantly on each path by the conjunction (i.e. the product) of its attackers.

Let us take an example to illustrate this problem. Assume we add in the graph $\mathbf{G}$ of the example 1 an attack from d to a. Since the argument d has two paths, we will indirectly, in the calculation, conjunct this event with itself, as if it were a different argument, but it is not the case. To accept b, from the attack of a, we have a case where d is "In" (accepted), so it defends b against a; another case is that of the attack of d when it is "Out" (not accepted) and so b could be "In". Of course, d cannot be both "In" and "Out" at the same time, but since local propagation does not detect that the probability of a and b depend on d, the result is that the probability of b is evaluated as if we had two d arguments, one d_1 attacking a and another d_2 attacking b.

Finally, we compare experimentally on 9 random SCN graphs between 200 and 1000 arguments/attacks, the runtimes of our two new algorithms, and apply the approximate Monte-Carlo method (Li, Oren, and Norman 2011), to investigate what error rate this method produces using the execution time of our algorithms, see Fig. 2. We used only one graph and took the argument with a maximum time in order to compare the methods as well as possible. Indeed, if we make several executions and we average, it biases the comparison with the Monte-Carlo method.

We observe that the `Fast_MCN`^{gr} algorithm is polynomial and the `Fast_SCN`^{gr} is linear. When running the Monte-Carlo method, we can notice that due to the longer time produced by `Fast_MCN`^{gr}, the approximation is good. However, when applied to the time of the `Fast_SCN`^{gr}, we see that the larger the graph, the more the approximation deteriorates, as the method decreases in the number of iterations and therefore in accuracy. Note also a non-regular error curve when using the `Fast_SCN`^{gr} time. Since there are few iterations, we can have randomly a good or bad approximation.

5 Conclusion

The constellations approach suffers from a high complexity due to the exponential number of generated worlds. In order to tackle this, we propose to compute the acceptance probability of an argument with two new algorithms, which are able to give the same score. In acyclic graphs, the time of our algorithm is related to the structure of the graph, we are not bound by the number of attackers but by the number of dependent arguments which is better. For the specific case of SCNs, we show that we can work in linearithmic time with an exact and faster solution than the Monte-Carlo approach.

In future work, we will investigate how to extend this algorithm to cyclic PrAFs, then how to compute the probability of acceptance of a set of arguments, and finally how extend it to other semantics.

References

Amgoud, L., and Ben-Naim, J. 2018. Evaluation of arguments in weighted bipolar graphs. *International Journal of Approximate Reasoning* 99:39–55.

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*.

Auger, N.; Jugé, V.; Nicaud, C.; and Pivoteau, C. 2018. On the worst-case complexity of timsort. *arXiv preprint arXiv:1805.08612*.

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.

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* 220:2.

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.

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.

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

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.

Meurer, A.; Smith, C. P.; Paprocki, M.; Čertík, O.; Kirpichev, S. B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J. K.; Singh, S.; et al. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science* 3:e103.

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

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

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.

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.