Sampling random spanning arborescences in graphs with low conductance

Statistics and Probability Letters 226 (2025) 110481

Contents lists available at ScienceDirect

Statistics and Probability Letters

journal homepage: www.elsevier.com/locate/stapro

Sampling random spanning arborescences in graphs with low conductance

Vittorio Zampinetti [a,b,*], Harald Melin [a], Jens Lagergren [a]

[a] Royal Institute of Technology (KTH), Stockholm, Sweden
[b] Politecnico di Torino, Italy

ARTICLE INFO

Keywords:
Random tree sampling
Bayesian inference
Random walk
Wilson's algorithm

ABSTRACT

Sampling random spanning arborescences in directed graphs is critical for applications in network analysis, optimization, and machine learning. While many state-of-the-art methods perform well on graphs with high conductance, they often fail or generalize poorly on low-conductance graphs. Inspired by Wilson's algorithm, we propose a novel sampling approach that overcomes this limitation by using dynamic programming to compute random walk probabilities. This avoids both inefficient walk simulations and numerically unstable Laplacian determinant calculations. Our method demonstrates superior efficiency and sampling quality in simulations, and is the only one to handle low-conductance graphs effectively.

1. Introduction

Generating random rooted spanning trees in directed weighted graphs, here referred to as *random spanning arborescences*, is a fundamental problem in graph theory (Lyons and Peres, 2017), and the applications in computer science and machine learning range from network analysis (Avena et al., 2018) to natural language processing (Koo et al., 2007) and phylogenetics (Melin et al., 2024; Koptagel et al., 2022).

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ be a weighted directed graph on a set $\mathcal{V}$ with n nodes where each arc $(i, j) \in \mathcal{E}$ has a non-negative real-valued weight w_{ij}. A spanning arborescence $\mathcal{T}$ of $\mathcal{G}$ is a directed tree rooted at some node $r \in \mathcal{V}$ in which all arcs point towards the root. A *random* spanning arborescence $\mathcal{T}$ is drawn from a distribution over all such trees where the probability of sampling $\mathcal{T}$ is proportional to its weight. Specifically, the probability of sampling a spanning arborescence $\mathcal{T}$ is given by

$$\mathbb{P}(\mathcal{T}) = \frac{1}{Z}\mathcal{W}(\mathcal{T}), \quad \mathcal{W}(\mathcal{T}) = \prod_{(i,j)\in\mathcal{T}} w_{ij},$$

where $\mathcal{W}(\mathcal{T})$ is the weight of $\mathcal{T}$, and Z is a normalization constant.

There are two main approaches to this problem. One is to use the classical Matrix-Tree Theorem (Kirchhoff, 1847; Tutte, 1948) which states that the normalization constant Z, i.e., the total weight of spanning trees in a directed graph, is equal to the determinant of the Laplacian of the weight matrix. Kulkarni (1990) and Colbourn et al. (1996) gave algorithms for using the MTT to sample spanning trees in directed graphs which run in $O(n^4)$ and $O(n^3)$ time respectively. Computing the determinant of a matrix is, however, known to be numerically unstable when the matrix is ill-conditioned (i.e., has low condition number) (Trefethen and Bau, 1997). The other approach is to sample spanning trees by simulating random walks on the graph and the algorithm by Wilson (1996) is the most well-known and widely used. Random walk based methods run in cover time, i.e., the expected time for a random walk

* Corresponding author at: Politecnico di Torino, Italy.
E-mail addresses: vittorio.zampinetti@polito.it (V. Zampinetti), haraldme@kth.se (H. Melin), jensl@kth.se (J. Lagergren).

https://doi.org/10.1016/j.spl.2025.110481
Received 24 April 2025; Received in revised form 1 June 2025; Accepted 2 June 2025
Available online 16 June 2025

to visit all the nodes in the graph, and thus benefit from high graph conductance. The *conductance* of a directed graph $\mathcal{G}$ can be defined as follows

$$\Phi(\mathcal{G}) = \min_{C \subset \mathcal{V}} \frac{\sum_{u \in C, v \in \mathcal{V} \setminus C} w_{uv}}{\sum_{u,v \in C} w_{uv}}.$$

Here, the minimum is taken over all nonempty, proper subsets $C \subset \mathcal{V}$ to avoid trivial partitions of the graph. In graphs with low conductance, the cover time cannot be bounded by a polynomial function of n and random walks become inefficient.

Recent applications of machine learning have prompted a demand for sampling arborescences algorithms to be used as estimators of the posterior distribution in Bayesian inference methods such as variational inference (Melin et al., 2024) or of gradients in deep learning (Paulus et al., 2021). However, graphs that arise from data typically have no guarantees of high conductance. Notably, if the graph has low conductance, by Cheeger's inequality (Cheeger, 1970), the Laplacian matrix is ill-conditioned and the determinant of the Laplacian matrix cannot be computed accurately. Determinant based methods will therefore fail, leaving no viable alternative. Several algorithms exist for the undirected case (Aldous, 1990; Madry et al., 2014; Durfee et al., 2017), but they do not extend to the directed setting. The problem of sampling spanning arborescences from low-conductance graphs has not been given sufficient attention in the literature, and the existing methods do not seem to be able to handle this case.

In this paper, we propose a novel method inspired by Wilson's random walk algorithm. Instead of constructing an arborescence by simulating a random walk, we devise a dynamic programming algorithm to efficiently compute and update the probabilities of all random walk outcomes. Our approach addresses the limitations of existing methods, resulting in a sampling technique which allows to efficiently draw samples of random spanning arborescences even in the presence of low conductance.

In Section 2, we introduce relevant graph theory concepts and describe Wilson's algorithm for sampling random spanning arborescences. In Section 3 we present our dynamic programming approach, which we call *Castaway*, and elaborate on how to use it for sampling random arborescences in low-conductance graphs. Then, in Section 4, we show that our method compares well with the state-of-the-art in terms of time, accuracy and feasibility, by reporting the results of experiments on simulated graphs. Finally, we conclude the paper with a discussion on the potential applications of our method and future work. The code for reproducing the experiments is available at `github.com/Lagergren-Lab/treesampling`.

2. Related methods

The algorithm by Wilson (1996) provides a method for sampling random spanning arborescences in directed graphs by simulating random walks.

Definition 1 (*Random Walk on Graph*)**.** Given a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, a *random walk* on $\mathcal{G}$ is a Markov chain over the nodes where the probability of transitioning to node v from node u is given by the normalized weight $\tilde{w}_{uv} = w_{uv} / \sum_{v'} w_{uv'}$.

The realization of the random walk, simply called a *walk*, is a sequence of nodes $v_1, \dots, v_k$ such that v_1 is the starting node and v_k is the last node visited. The probability of a random walk from v_1 to v_k is given by the product of the transition probabilities along the walk. If the walk visits a node more than once, the path is said to contain a *loop*.

Definition 2 (*Loop-Erased Random Walk*)**.** A random walk where all loops are erased. The loop-erased random walk generates a sequence of distinct nodes $v_1, \dots, v_k$ (a *path*) where each arc (v_i, v_{i+1}) is the last arc leaving v_i in the random walk.

Wilson's algorithm. Given a fixed root $r \in \mathcal{V}$, the algorithm initialize the tree to contain only the root node. Then a loop-erased random walks starts from $v_1 \neq r$, picked from any of the nodes not in the tree. The walk proceeds until it transitions to the root. The path thereby generated from v_1 to r is added to the tree. Another loop-erased random walk is started from the next node v_1' not in the tree and terminates when a node in the tree is visited. The path so obtained is appended to the tree. The process continues until all nodes have been added to the tree. The algorithm is guaranteed to sample a random spanning tree with probability proportional to the product of the weights of its arcs. However, the following argument shows that the algorithm may take a long time to visit all nodes in the graph. For example, in a rooted directed graph with a single root node r and only two other nodes v_1 and v_2 that are strongly connected to each other and weakly connected to r, the random walk starting from v_1 will cycle between v_1 and v_2 for a time inversely proportional to the conductance of the graph before visiting the root and therefore terminating the walk.

3. Methods

We now describe our algorithm Castaway for sampling random spanning arborescences. We give an overview of the algorithm and then describe the details of the computation of the probabilities. Subsequently, we discuss instability issues for graphs with low conductance and how to resolve them.

3.1. Algorithm overview

Let $(v_1, \dots, v_k)$ be a walk on $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ right before reaching a node in the current tree $\mathcal{S} = (\mathcal{V}_S, \mathcal{E}_S)$ in an iteration of Wilson's algorithm. We are interested in the first and last nodes v_1, v_k. The probability of walking on the set of nodes $\mathcal{U} = \mathcal{V} \setminus \mathcal{V}_S$ from v_1 to v_k is given by the sum of the probabilities of all walks with the same endpoints. This quantity, that we denote by $P_{\mathcal{U}}(v_1, v_k)$, can be computed efficiently with dynamic programming. Let $\mathcal{O}$ be the *out set*, that is the set of nodes which, once reached, terminate the random walk, and initialize it to $\mathcal{O} = \mathcal{V}_S$. The probability that v_k is the *exit node*, that is the last node visited in the walk before reaching $\mathcal{O}$, is given by

$$q_{v_k}(v_1) = P_{\mathcal{U}}(v_1, v_k) \sum_{v_o \in \mathcal{O}} \tilde{w}_{v_k v_o} . \tag{1}$$

Then, for any node $v_o \in \mathcal{O}$, the probability of ending the walk in v_o and therefore adding the arc (v_k, v_o) is $\tilde{w}_{v_k v_o}$. After adding the arc (v_k, v_o) to the tree, v_k is removed from the set $\mathcal{U}$. This step fixes the last arc in the loop-erased random walk started from v_1 and, in order to further draw arcs backwards in the path, we set $\mathcal{O} = \{v_k\}$ as the new out set. The table $P_{\mathcal{U}}$ is updated by subtracting the probability of all walks that go through v_k. The algorithm continues by drawing the node $v_{k-1} \in \mathcal{U}$ proportionally to $P_{\mathcal{U}}(v_1, v_{k-1})\tilde{w}_{v_{k-1} v_k}$ and adding the arc (v_{k-1}, v_k) to the tree. This is repeated until the start node v_1 joins the tree. Then as in Wilson's algorithm, the next start node v_1' is picked from the set of nodes not in the tree $\mathcal{U}$ and the process is repeated. The algorithm ends when all nodes are added to the tree. We provide the pseudocode in Appendix B in the supplementary material online.

Probability table. The table is built by iteratively adding nodes to the set $\mathcal{U} = \{v_1, \dots, v_m\}$ from those not in the tree. Let $\mathcal{U}^{(k)}$ be the set of k nodes in the graph that have been added to $\mathcal{U}$. Initially, $\mathcal{U}^{(1)} = \{v_1\}$ and $P_{\mathcal{U}^{(1)}}(v_1, v_1) = 1$. We define $P_{\mathcal{U}^{(k)}}(v_i, v_j)$ recursively:

$$P_{\mathcal{U}^{(k)}}(v_i, v_j) = P_{\mathcal{U}^{(k-1)}}(v_i, v_j) + R^{\text{out}}_{\mathcal{U}^{(k-1)}}(v_i, v_k) \times \rho_{\mathcal{U}^{(k)}}(v_k) \times R^{\text{in}}_{\mathcal{U}^{(k-1)}}(v_k, v_j) , \tag{2}$$

where

$$\begin{aligned} c R^{\text{out}}_{\mathcal{U}^{(k-1)}}(v_i, v_k) &= \sum_{v \in \mathcal{U}^{(k-1)}} P_{\mathcal{U}^{(k-1)}}(v_i, v)\, \tilde{w}_{v v_k} , \\ R^{\text{in}}_{\mathcal{U}^{(k-1)}}(v_k, v_j) &= \sum_{v \in \mathcal{U}^{(k-1)}} \tilde{w}_{v_k v}\, P_{\mathcal{U}^{(k-1)}}(v, v_j) \text{ and} \\ \rho_{\mathcal{U}^{(k)}}(v_k) &= \left(1 - \sum_{u \in \mathcal{U}^{(k-1)}} \tilde{w}_{v_k u}\, R^{\text{out}}_{\mathcal{U}^{(k-1)}}(u, v_k) \right)^{-1} \end{aligned} \tag{3}$$

are the probabilities of walking out of $\mathcal{U}^{(k-1)}$, walking in $\mathcal{U}^{(k-1)}$, and, from v_k, returning to itself through $\mathcal{U}^{(k)}$ respectively. Full derivation of the formula is given in Appendix A in the supplementary material online.

3.2. Resolving instability

The algorithm may fail to build the probability table due to numerical instability. In particular, when a node v_k is added to the set of nodes, the return probability $\rho_{\mathcal{U}^{(k)}}(v_k)$ may numerically evaluate to 1, causing the geometric series in Eq. (3) to diverge. This phenomenon arises because a random walk restricted to $\mathcal{U}$ visits v_k with such high frequency that limited machine precision leads to numerical overflow. To characterize this behaviour, we identify a set of nodes – hereafter referred to as *component* – in which the ratio between the total weight of outgoing arcs and the total weight of internal arcs is vanishingly small. This structure corresponds to a region of low conductance. Furthermore, a high return probability at v_k often implies that such node receives high-weighted arcs even from outside the component. The presence of one or more such nodes in the graph increase the likelihood of near-linear dependencies in the Laplacian matrix, which in turn complicates the computation of the determinant. We fix to this issue by handling nodes that lead to such instability in a special way. We call these nodes *crasher nodes*. By excluding the crasher nodes from the set $\mathcal{U}$, the probability table can be computed without numerical issues. The probability of attaching a node v_k to the tree is approximated with

$$q_{v_k}(v_1) = \sum_{u \in \mathcal{U}} P_{\mathcal{U}}(v_1, u) \sum_{c \in \mathcal{X}} \tilde{w}_{uc} R^{\text{in}}_{\mathcal{U}}(c, v_k) \sum_{v_o \in \mathcal{O}} \tilde{w}_{v_k v_o} , \tag{4}$$

where $\mathcal{X}$ is the set of crasher nodes in the graph. The formula relies on the reasonable assumption that any random walk that leaves a component more than once gives a negligible contribution to the total probability of the walks. Edge cases such as v_k being a crasher node or v_1 being a crasher node are further discussed in Appendix C in the supplementary material online. The original algorithm is extended to handle this case by identifying crasher nodes as follows: if the return probability of a node becomes infinite, we add that node to the set of crasher nodes and adopt the new formula for sampling exit nodes. This fix brings the complexity of the algorithm up to $O(n^4 K)$ which is still polynomial in the number of nodes n and K being the number of crasher nodes.

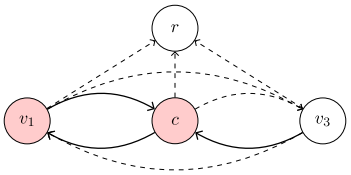
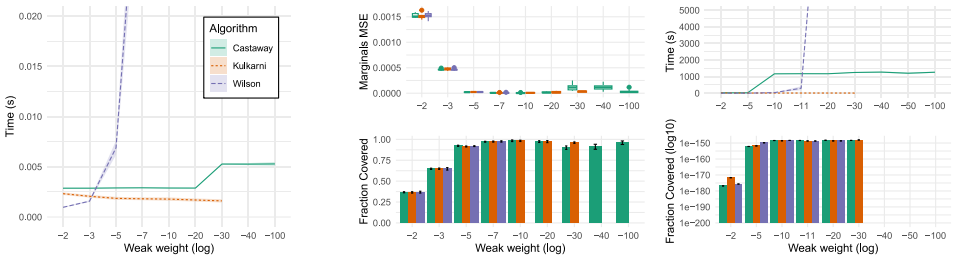


Fig. 1. Example of a graph with four nodes and one crasher component (red). Dashed arcs represent ε-weighted arcs, while solid arcs have unit weights. The crasher c is visited many times with high probability by a random walk from any node inside and outside the component, e.g., v_1 and v_3 respectively.



(a) Running time (b) Sampling accuracy (c) Larger graph ($n = 100$)

Fig. 2. (a) Average running time over 10 differently weighted graphs plotted as a function of $\log(\varepsilon)$. Shaded regions show the standard deviation of the estimates. The lineplot for Kulkarni is incomplete due to failure in computing the determinant in low-conductance graphs. (b) Boxplots (upper plot) show the MSE for marginal distribution of arcs against ground truth and bars (lower plot) show the fraction of probability mass covered by the sampled trees. Error bars indicate one standard deviation from the estimates. (c) Running time and fraction of probability mass covered by the sampled trees for graphs with $n = 100$ nodes.

4. Experiments

To evaluate our method we run experiments on simulated graphs. We assess the performance of the algorithm compared to Wilson's algorithm and algorithm A8 in Kulkarni (1990) (hereafter referred to as *Kulkarni*), in terms of running time and distribution of the sampled trees. Although Kulkarni's algorithm is not the fastest determinant-based method – e.g., Colbourn's runs in $O(n^3)$ as noted in the Introduction – it is more numerically stable, and the runtime difference is irrelevant for the purpose of our investigation. In order to highlight the weaknesses of the existing methods, we generate graphs that have low conductance. We address the unstable case of the *crasher nodes* as described in Section 3.2 and in order to do so, we simulate graphs with a certain structure. These graphs have K components, each of which contains $\lfloor n/(K+1) \rfloor$ nodes that have arcs with unit weights between each other and ε-weighted arcs to nodes outside their component, with $\varepsilon > 0$ being a parameter of the simulation. Each component contains a crasher node. The remaining $n - \lfloor n/(K+1) \rfloor$ nodes are further divided such that one node is set as the root, with no incoming arcs (zero weight) and ε-weighted outgoing arcs, and the rest of the nodes have ε-weighted arcs to all other nodes except the crasher nodes, which receive unit weight arcs. A schematic representation of the graph structure is shown in Fig. 1.

We generate graphs with $n = 7$ nodes divided into $K = 2$ crasher components with two nodes each, one root node and another two-node non-crasher component. This size is small enough to allow for a complete enumeration of the spanning arborescences, which we use to compute the ground truth and benchmark the algorithms. The ε-weight, or *weak weight*, takes values such that $\log(\varepsilon) \in \{-2, -3, -5, -7, -10, -20, -40, -100\}$, and for each value, we generate 10 perturbed matrices of weights. We run the three algorithms on $N = 1000$ samples and compare three metrics: running time, mean squared error (MSE) of the marginal distribution of arcs and fraction of probability mass covered by the sampled trees. To improve the accuracy of all algorithms, the sampled trees are re-weighted, meaning that their frequency is replaced by the tree weight divided by the total weight of the sampled trees. The Castaway algorithm is executed in its $O(n^3)$ version until it fails to compute the probability table due to numerical overflow. Past that point, we run the algorithm with the instability fix described in Section 3.2. Fig. 2 summarizes the results of the experiments. In particular, running times are plotted in Fig. 2(a) and show that Castaway is the only method that allows to sample spanning arborescences with weaker weights in a reasonable time, while still being able to sample the correct distribution, as shown in Fig. 2(b). Castaway is thus proven to outperform the other two methods since, as the weak weight decreases, Wilson's algorithm rapidly becomes impractical, and for $\log(\varepsilon) \leq -40$ Kulkarni's algorithm fails to sample arborescences due to errors in the determinant computation.

We repeat the experiment on larger graphs ($n = 100$) with unchanged number of components and reduced sample size ($N = 100$) to keep the running time reasonable. The results are shown in Fig. 2(c). The increased size of the graph does not allow for a complete enumeration of the spanning arborescences, which is why the MSE is not computed. The fraction of total probability mass can only be computed when ε is large enough for the determinant computation to be stable. Nevertheless, the running time follows the same trend as in the smaller graphs experiment and the quality of the sampled trees are similar across the three algorithms.

5. Conclusion

We present a novel random spanning arborescence sampling algorithm for directed graphs. The method runs in polynomial time with n as the number of nodes in the graph and succeeds in sampling spanning trees even in the presence of weak connections, where existing methods fail. We demonstrate the effectiveness of our method through simulation experiments on graphs with weak weights and show that it is the only method able to sample spanning arborescences in a reasonable time. Future work may include the extension of the algorithm to sampling arborescences without a fixed root.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.spl.2025.110481.

Data availability

No data was used for the research described in the article.

References

Aldous, D.J., 1990. The random walk construction of uniform spanning trees and uniform labelled trees. SIAM J. Discrete Math. 3 (4), 450–465.
Avena, L., Castell, F., Gaudillière, A., Mélot, C., 2018. Random forests and networks analysis. J. Stat. Phys. 173, 985–1027.
Cheeger, J., 1970. A lower bound for the smallest eigenvalue of the Laplacian. In: C., Gunning R. (Ed.), Problems in Analysis: A Symposium in Honor of Salomon Bochner. Princeton Univ. Press, Princeton, NJ, pp. 195–199.
Colbourn, C.J., Myrvold, W.J., Neufeld, E., 1996. Two algorithms for unranking arborescences. J. Algorithms 20 (2), 268–281.
Durfee, D., Kyng, R., Peebles, J., Rao, A., Sachdeva, S., 2017. Sampling random spanning trees faster than matrix multiplication. In: Proc. 49th ACM SIGACT Symp. Theory Comput. pp. 730–742.
Kirchhoff, G., 1847. Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. Ann. Phys. 148 (12), 497–508.
Koo, T., Globerson, A., Carreras, X., Collins, M., 2007. Structured prediction models via the matrix-tree theorem. In: Proc. EMNLP-CoNLL. pp. 141–150.
Koptagel, H., Kviman, O., Melin, H., Safinianaini, N., Lagergren, J., 2022. VaiPhy: a variational inference based algorithm for phylogeny. Adv. Neural Inf. Process. Syst. 35, 14758–14770.
Kulkarni, V.G., 1990. Generating random combinatorial objects. J. Algorithms 11 (2), 185–207.
Lyons, R., Peres, Y., 2017. Probability on Trees and Networks. Cambridge University Press.
Madry, A., Straszak, D., Tarnawski, J., 2014. Fast generation of random spanning trees and the effective resistance metric. In: Proc. 26th ACM-SIAM Symp. Discrete Algorithms.
Melin, H., Zampinetti, V., McPherson, A., Lagergren, J., 2024. Victree: a variational inference method for clonal tree reconstruction. In: Proc. RECOMB 2024. Cambridge, MA, USA, pp. 429–433.
Paulus, M.B., Choi, D., Tarlow, D., Krause, A., Maddison, C.J., 2021. Gradient estimation with stochastic softmax tricks. arXiv preprint arXiv:2006.08063.
Trefethen, L.N., Bau, III, D., 1997. Numerical Linear Algebra. SIAM, Philadelphia, PA.
Tutte, W.T., 1948. The dissection of equilateral triangles into equilateral triangles. Math. Proc. Cambridge Philos. Soc. 44 (4), 463–482.
Wilson, D.B., 1996. Generating random spanning trees more quickly than the cover time. In: Proc. 28th ACM Symp. Theory Comput. pp. 296–303.