CSC 2240H : Graphs, Matrices, and Optimization December 23, 2022

Generating Uniform Spanning Trees

Haonan Duan, Yangjun Ruan

In this lecture, we introduce two randomized algorithms for generating spanning trees uniformly. Given a connected, undirected and unweighted graph $G(V, E)$ with n vertices and m edges, the goal is to generate a spanning tree T of G chosen **uniformly** at random among all spanning trees of G. Recall that a spanning tree T of an undirected graph G is a subgraph that is a tree which includes all vertices of G.

1 The first algorithm: random walk

A random walk on G is a Markov chain over all vertices with the transition matrix:

$$P(w|v) = \begin{cases} \frac{1}{r_v} & \text{if (v, w) is an edge.} \\ 0 & \text{o.w.} \end{cases}$$

where r_v is the degree of the node v.

Algorithm I [Ald90] [Bro89]

Simulate a simple random walk on the graph G starting at an arbitrary vertex s until every vertex is visited. For each vertex $i \in V - s$ collect the edge $\{j, i\}$ that corresponds to the first entrance to vertex i. Let T be this collection of edges. Output T.

Our first algorithm is that a random walk can simulate a uniform spanning tree by selecting the first edge that enters the node. The pseudocode is shown above. The running time of Algorithm I is $O(mn)$. It's clear that T is a spanning tree because it contains $|V| - 1$ edges and no loops. We shall prove that it is indeed uniformly distributed.

Before discussing the algorithm, it will be useful to introduce *arborescences* defined as below:

Definition 1.1. (Arborescence) For a given $s \in G$, an arborescence T *rooted at* s is a directed spanning tree of G where all vertices in $G \setminus \{s\}$ have *exactly one* incoming arc.

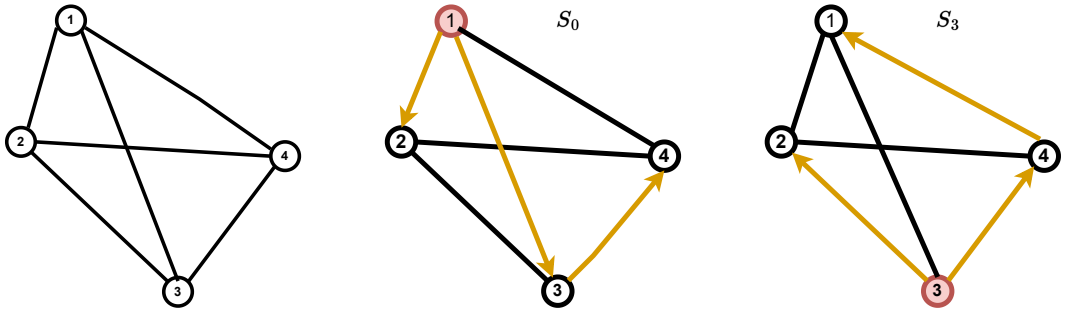
It is easy to check that there is a one-to-one correspondence between spanning trees of G and arborescences rooted at s: given any spanning tree, there is a unique way of directing its edges to make it an arborescence rooted at s; conversely, given any arborescence rooted at s, one can obtain a spanning tree by simply disregarding the direction of the edges. Therefore, if we get a procedure that generates arborescences, it also gives a procedure that generates spanning trees.

Indeed, random walk simulates a chain of arborescences along the way.

Definition 1.2. (Arborescence chain). Let $(X_j; -\infty < j < \infty)$ be the Markov chain of a random walk on G. We can define a Markov chain of arborescences $(S_j; -\infty < j < \infty)$. Considering

$(X_j, X_{j+1}, \cdots)$, we choose X_j to be the root of S_j and select the first edge entering other vertices to be in the tree. In other words, the tree S_j is rooted at X_j with edges $(X_{T_v^j - 1}, X_{T_v^j}); v \neq X_j$, where $T_v^j = \min\{k \leq j; X_k = v\}$. An example is given in Figure 1.

0	1	2	3	4	5	6	$7 \cdots$
1	2	1	3	4	1	3	$2 \cdots$



(a) G (b) S_0 (c) S_3

Figure 1: An example of an arborescene chain. The table above is a random walk over graph G. Then by Definition 1.2, $S_0 = \{[1,2],[1,3],[3,4]\}$ and $S_3 = \{[3,4],[4,1],[3,2]\}$

Proposition 1.3. *$(S_j; -\infty < j < \infty)$ is an irreducible and reversible Markov chain.*

Proof. Irreducibility and reversibility are directly from the assumption that G is connected and unweighted. □

Theorem 1.4. *Let $N(G)$ be the number of spanning trees t of G. Then $P(T = t) = 1/N(G)$ for each t produced by Algorithm 1.*

We will first prove a simpler case when the graph is r-regular.

Proof. Let G be an r-regular graph. Let $(X_j; -\infty < j < \infty)$ and $(S_j; -\infty < j < \infty)$ be the Markov chain of the nodes and arborescences introduced previously.

Consider the transition matrix Q for S_j in reversed time:

$$Q(t, t') = P(S_{-1} = t' | S_0 = t)$$

Then

- Given t, there are exactly r trees t' such that $Q(t, t') = \frac{1}{r}$ and $Q(t, t') = 0$ for other trees.
- Given t', there are exactly r trees t such that $Q(t, t') = \frac{1}{r}$ and $Q(t, t') = 0$ for other trees.

This is because X_{-1} has $\frac{1}{r}$ probability to be each of the r neighbours of v. Each possibility leads to S_{-1} being some tree t', and these also are the only possibilities. A similar argument can be made going from S_{-1} to S_0. Therefore, Q is doubly-stochastic, which further implies that the stationary distribution of S_j is uniform. □

To extend the above proof to non-regular graphs, we first make the observation that each tree t has $r(v)$ neighbours, where v is the root of the tree. This implies that we can swap r in the two bullet points above with the degree of the roots. It then follows that the stationary distribution of t is proportional to the degree of its root. Thus, all directed spanning trees rooted at the same node are uniform. Since our trees are undirected and unrooted, all T in Algorithm 1 are uniform.

2 Faster Generation of Spanning Trees

In this section, we will introduce a faster algorithm for generating *approximately uniform* random spanning trees. In particular, we focus on the generation of $\delta-$random spanning trees:

Definition 2.1 ($\delta-$random spanning trees)**.** A randomized algorithm A that generates $\delta-$random spanning trees outputs a random spanning tree T with probability $p(T)$ that is $\delta-$*far from uniform*, i.e.,

$$\frac{1-\delta}{|\mathcal{T}(G)|} \leq p(T) \leq \frac{1+\delta}{|\mathcal{T}(G)|}$$

where $\mathcal{T}(G)$ is the set of spanning trees of G.

With the same reasoning as before, if we get a procedure that generates $\delta-$random arborescences, it also gives a procedure that generates $\delta-$random spanning trees.

[KM09] introduces a faster generation algorithm which generates $\delta-$random spanning trees in expected time of $\tilde{O}(m\sqrt{n}\log(1/\delta))$. For brevity of illustration, we will focus on a simplified algorithm that gives an expected running time of $\tilde{O}(m^2/\sqrt{n}\log(1/\delta))$, which shares the same spirit.

The key idea of the algorithm is the observation that the random walk algorithm may spend a long time walking in regions that have already been covered. Indeed, the random walk algorithm has a running time of $O(mn)$, while only a tiny fraction $O(n)$ is used for recovering an arborescence. Therefore, the algorithm seeks to obtain a shortcut that cuts out the random walk corresponding to visiting already explored parts of G. The essential steps are to first decompose the graph into small subgraphs that can be quickly covered, and then shortcut the walk inside each subgraph *if it is already covered*.

2.1 Decompose the graph

First, we define the (ϕ, γ)-Decomposition that will permit the implementation of the fast generation algorithm. Let $(D_1, \ldots, D_k, S, C)$ denote a partition of G, where D_i are disjoint subgraphs, $S = V(G) \setminus \bigcup_i V(D_i)$ is the set of remaining vertices, and $C = E(G) \setminus \bigcup_i E(D_i)$ is the set of edges not entirely contained inside one of D_i. For a given D_i, let $C(D_i)$ be the subset of C incident to D_i and $U(D_i)$ be the set of vertices of D_i incident to an edge from C.

Definition 2.2 ((ϕ, γ)-decomposition)**.** $(D_1, \ldots, D_k, S, C)$ is a (ϕ, γ)-decomposition if:

1. $|C| \leq \phi|E(G)|$
2. $\forall i$, the diameter $\gamma(D_i) \leq \gamma$
3. $\forall i$, $|C(D_i)| \leq |E(D_i)|$

Note that the first condition ensures that the subgraphs $D_1, \ldots, D_k$ contain most (all but a ϕ fraction of) edges in G. Intuitively, the second condition ensures that each subgraph could be covered relatively quickly, using the fact that the cover time of an unweighted graph G' with diameter $\gamma(G')$ is at most $O(|E(G')|\gamma(G'))$ [Ale+79].

For reasons that will become clear later, we consider a specific decomposition that can be quickly computed, shown by the following lemma:

Lemma 2.3 (Obtaining good $((\phi, \gamma)$-decompositions)**.** *For G and any $\phi = o(1)$, a $(\phi, \tilde{O}(1/\phi))$-decomposition of G can be computed in time $\tilde{O}(m)$.*

We omit the proof for brevity and refer interested readers to Lemma 13 and its proof in [KM09].

2.2 Decompose the walk

Then, we considet the random walk $X = (X_i)$ over the decomposed graph started from a vertex chosen by the stationary distribution G. Let τ be the cover time of G, i.e., the first time that the walk visits all the vertices of G. The result in [Ale+79] yields that $E[\tau] = O(mn)$ which is the expected time of traditional random walk.

We decompose the walk over decomposed G. Let Z and Z_i be the random variables corresponding to the number of times that X traverses edges from C and inside D_i respectively. By definition, we have that $\tau = \sum_i Z_i + Z$. The following lemma establishes the expectation of Z:

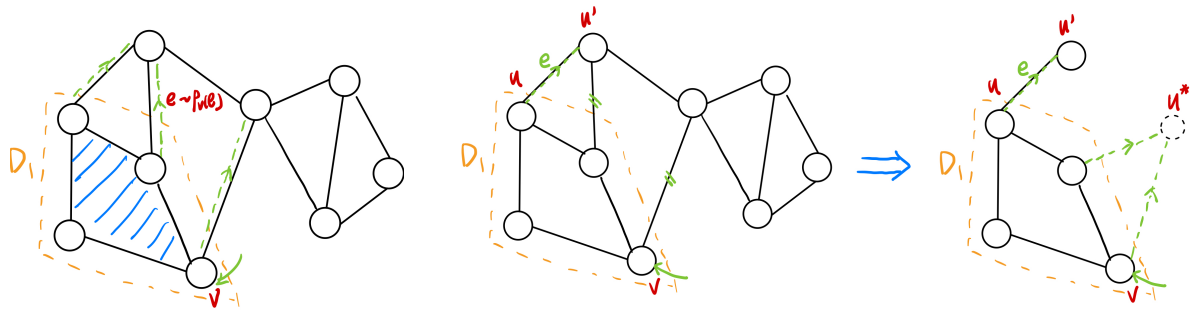
Fact 2.4. *The expected traversed time in C is $E(Z) = O(\phi mn)$.*

Proof. Since the random walk starts from the stationary distribution of G, the expected number of traversals of edges from C is just proportional to its size. Therefore, the assumption $|C| \leq \phi|E(G)|$ implies the above fact. □

As discussed before, we will be interested in the cover time for each subgraph. In particular, let Z_i^* be the random variable corresponding to the number of traversals inside D_i until X explores the whole subgraph D_i, we have that:

Lemma 2.5 (Expected cover time for subgraphs)**.** $E[Z_i^*] = \tilde{O}(|E(D_i)|\gamma(D_i))$.

Proof. Let us fix $D = D_i$. For a vertex $v \in V(D)$, let $d_G(v)$ be the degree of v in G and $d_D(v)$ be the degree of v in G. For $u, v \in U(D)$, let $p_{u,v}^D$ be the probability that a random walk in G that starts at u will reach v through a path that does not pass through any edge inside D. Consider a (weighted) graph D', which we obtain from D by adding, for each $u, v \in U(D)$, an edge (u, v) with weight $d_G(u)p_{u,v}^D$. We note that if we take our walk X and filter out of the vertices that are not from D, then the resulting "filtered" walk Y_D will just be a natural random walk in D'. As a result, it is easy to see that, in this case, $E[Z_i^*]$ can be upper-bounded by the expected time needed by a random walk in D', started at arbitrary vertex, to visit all of the vertices in D' and then reach some vertex in $U(D)$. Therefore, to bound $E[Z_i^*]$, it suffices to bound the cover time of D'. The cover time of undirected graph G' is at most $2 \log |V(G')| H_{max}(G')$, where $H_{max}(G')$ is the maximal hitting time. Following the results of [Ale+79], we have $H_{max}(G') \leq |E(G')|\gamma(G')$. □



(a) (b)

Figure 2: (a) The shortcut of trajectory inside D_1 Is implemented by $P_v(e)$ that characterizes the probability of X leaving D_1 through edge e after entering through vertex v; (b) The illustration for computing $P_v(e)$ by adding a dummy node u^* which connects to all other leaving edges.

2.3 Shortcut the walk

Recall that the key idea of the algorithm is to shortcut the trajectory inside D_i after D_i has already been covered. More specifically, consider the walk after D_i is covered, this means that we already know for all edges $v \in V(D_i)$ which arc e_v should be added to the arborescence. Any further walk inside D_i provides no new information for constructing the arborescence, and could be shortcutted.

The main tool for implementing the shortcut is $P_v(e)$ that characterizes the probability of X leaving D_i *through edge* e *after entering through vertex* v, see Fig. 2a for an illustration. If we know $P_v(e)$ for all $v \in V(D_i)$ and all $e \in C(D_i)$, then we could essentially immediately choose the leaving edge e upon entering v without computing the explicit trajectory in D_i.

To formalize the intuition, let $\bar{X} = (\bar{X}_1, \ldots, \bar{X}_m)$ be a decomposition of the walk $X = (X_1, \ldots, X_\tau)$ into contiguous blocks $\bar{X}_j$ that are contained in D_{i_j} for $i_j \in \{0, \ldots, k\}$, where $D_0 = S$. We construct the shortcutted walk $\tilde{X} = (\tilde{X}_i)$ by processing $\bar{X}$ block by block: if D_{i_j} has not been covered or $i_j = 0$, we copy $\bar{X}_j$ to $\tilde{X}$, otherwise we copy only the first and last entries of the block.

With $P_e(v)$, the following lemma establishes that $\tilde{X}$ can be simulated efficiently:

Lemma 2.6 (Simulating $\tilde{X}$ with $P_v(e)$)**.** *Knowing $P_v(e)$ for all $e \in C(D_i), v \in V(D_i)$ and i, we can preprocess these values in $\tilde{O}(\phi mn)$ time and it allows simulation of l steps of $\tilde{X}$ in time $\tilde{O}(l)$.*

Proof. Simulating $\tilde{X}$ before D_i is covered is straightforward. The only thing to show is that the shortcutting of blocks can be simulated efficiently. To show this, we consider some i and $v \in U(D_i)$, and construct an array $A_v(m)$ where $A_v(m) = \sum_{1 \le j \le m} P_v(e_j)$ for $m \in \{1, \ldots, |C(D_i)|\}$ in $\tilde{O}(|C(D_i)|)$. So we can construct for all $v \in U(D_i)$ in $\tilde{O}(|C(D_i)||V(D_i)|)$. Summing over all D_i gives the above bound on preprocessing time. Furtheremore, this array enables choosing e with $P_v(e)$ with binary search in polylogarithmic time. □

From this lemma, we can show that we can find a random arborescence of G efficiently:

Lemma 2.7 (Finding a random arborescence)**.** *Given a (ϕ, γ)-decomposition of G and $P_v(e)$, we can find a random arborescence of G in expected time $\tilde{O}(m(\gamma + \phi n))$.*

Proof. We only need to compute the expected length of the shortcutted walk $\tilde{X}$, which is upper-bounded by the following quantity:

$$\sum_i \underbrace{E[Z_i^*]}_{\text{cover time of } D_i} + \underbrace{E[Z]}_{\text{traversals in } C} + \underbrace{2E[Z]}_{\text{shortcutted traversals}}$$

The first and the second terms are obvious, and the last term is due to the fact that the two vertices that remain in $\tilde{X}$ after shortcutting some block from X can be amortized into the number of traversals by X of some edges in C. By the second assumption in Definition 2.2, Fact 2.4, Lemma 2.5, we get that $\sum_i E[Z_i^*] + 3E[Z] = \tilde{O}(\sum_i |E(D_i)\gamma + \phi mn|) = \tilde{O}(m(\gamma + \phi n))$. By Lemma 2.6, the shortcutted work can be simulated in expected time $\tilde{O}(m(\gamma + \phi n))$. □

Then the remaining question is whether we can compute $P_e(v)$ efficiently, established by the following lemma:

Lemma 2.8 (Computing $P_v(e)$)**.** *Given a (ϕ, γ)-decomposition of G, we can compute multiplicative $(1+\varepsilon)$-approximations of $P_v(e)$ in time $\tilde{O}(\phi m^2 \log(1/\varepsilon))$*

Proof. Let us fix some $D = D_i$ and an edge $e = (u, u') \in C(D)$ with $u \in U(D)$. Consider a graph D' obtained by adding a dummy node u^* to D and then connecting all other leaving edges to u^*, i.e., for each $(w, w') \in C(D) \setminus \{e\}$, we add an edge (w, u^*) (note that w' can be equal to u'), see Fig. 2b for an illustration. Note that $P_v(e)$ is the probability that the walk started at v will *hit* u' *before* u^*, which can be quickly computed using *electrical flows*.

In particular (see, e.g., [Lov93]), we can treat D' as an electrical circuit where we impose voltage of 1 at u' and 0 at u^*, then the voltage achieved at v is exactly equal to $P_v(e)$. We can compute a $(1+\varepsilon)$-approximation in $\tilde{O}(|E(D')| \log 1/\varepsilon)$ using the linear solver in [ST14]. Computing for each $e \in C(D)$ and store the probabilities for all vertices v, the total running time can be bounded by $\tilde{O}(|C| \sum_i |E(D_i)| \log 1/\varepsilon) = \tilde{O}(\phi m^2 \log 1/\varepsilon)$, where we use the fact that $|E(D')| = |E(D)| + |C(D)| \leq 2|E(D)|$ by assumption 3 in Definition 2.2. □

Note that the above lemma only gives an approximation of $P_v(e)$. We need to show that it is sufficient for controlling the overall error and maintaining a good running time:

Lemma 2.9 (An approximate $P_v(e)$ is sufficient)**.** *Given a (ϕ, γ)-decomposition of G and multiplicative $(1+\varepsilon)$-approximation of $P_v(e)$, we can generate a $\delta-$random arborescence of G in expected time $\tilde{O}(m^2(\gamma + \phi n))$ as long as $\varepsilon \leq \delta/mn$.*

We omit the proof for brevity and refer interested readers to Lemma 10 and its proof in [KM09].

Putting all together, we have the theorem establishing the running time of the algorithm:

Theorem 2.10 (Total complexity)**.** *For any $\delta > 0$, we can generation a $\delta-$random spanning tree of G in expected time of $\tilde{O}(m/\sqrt{n} \log(1/\delta))$.*

Proof. Let $\phi = 1/n^{1/2}, \varepsilon = \delta/mn$, the total expected time can be decomposed as:

1. Get a $(1/n^{1/2}, \tilde{O}(n^{1/2}))$-decomposition: $\tilde{O}(m)$ (Lemma 2.3).
2. Compute estimate of $P_v(e)$: $\tilde{O}(m^2/\sqrt{n} \log(1/\delta))$ (Lemma 2.8).
3. Generate a δ-arborescence: $\tilde{O}(m\sqrt{n})$ (Lemma 2.7, Lemma 2.9).

Summinga all together, we have the total running time of $\tilde{O}(m^2/\sqrt{n}\log(1/\delta))$. □

We refer interested readers to Sec. 4 in [KM09] for an improved algorithm with $\tilde{O}(m\sqrt{n}\log(1/\delta))$, which is obtained by a stronger decomposition and a slightly different random walk.

References

[Ald90] David J Aldous. "The random walk construction of uniform spanning trees and uniform labelled trees". In: *SIAM Journal on Discrete Mathematics* 3.4 (1990), pp. 450–465 (cit. on p. 1).

[Ale+79] Romas Aleliunas, Richard M Karp, Richard J Lipton, László Lovász, and Charles Rackoff. "Random walks, universal traversal sequences, and the complexity of maze problems". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE Computer Society. 1979, pp. 218–223 (cit. on p. 4).

[Bro89] Andrei Z Broder. "Generating random spanning trees". In: *FOCS*. Vol. 89. 1989, pp. 442–447 (cit. on p. 1).

[KM09] Jonathan A Kelner and Aleksander Madry. "Faster generation of random spanning trees". In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE. 2009, pp. 13–21 (cit. on pp. 3, 4, 6, 7).

[Lov93] László Lovász. "Random walks on graphs". In: *Combinatorics, Paul erdos is eighty* 2.1-46 (1993), p. 4 (cit. on p. 6).

[ST14] Daniel A Spielman and Shang-Hua Teng. "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems". In: *SIAM Journal on Matrix Analysis and Applications* 35.3 (2014), pp. 835–885 (cit. on p. 6).