# Kelner-Madry's $\widetilde{O}\left(m\sqrt{n}\log 1/\delta\right)$ Algorithm for Generating Random Spanning Trees*

Abbas Mehrabian[†]

University of Waterloo

April 4, 2011

## 1 Introduction

In FOCS 2009, Kelner and Madry presented an $\widetilde{O}(|E|\sqrt{|V|}\log 1/\delta)$ algorithm for generating $\delta$-random spanning trees of a graph $G = (V, E)$. Their result is presented in a simpler and hopefully more clear way here. To make in easier to understand, we will focus on the main ideas, and not the details. For example, to make things simpler, we will restrict ourselves to spanning trees, whereas to get the algorithm with the mentioned running time, one has to use directed rooted trees. We will use the arXiv version of their paper [5].

Given an undirected graph $G = (V, E)$, we want to randomly generate a spanning tree that is chosen uniformly at random from the set of spanning trees of $G$. This problem is interesting mainly from theoretical point of view. Recently, random spanning trees have also been used to build efficient sparsifiers.

In the following, $n$ and $m$ denote the number of vertices and edges of $G$, respectively. We hide the $\log n$ factors by using the $\widetilde{O}$ notation.

There are two general approaches to this problem. The first one is based on a theorem of Kirchhoff, which reduces the problem of counting the number of spanning trees to the evaluation of a determinant. After a sequence of papers, this approach resulted in an

*This is the report of my presentation on March 29th, for the C&O algorithms reading group and the course CO 750: Randomized Algorithms, Winter 2011, University of Waterloo.

[†]`amehrabi@uwaterloo.ca`

algorithm by Colbourn, Myrvold, and Nuefeld [4], whose running time is the amount of time necessary to multiply two $n \times n$ matrix, the best known bound for which is $O(n^{2.376})$.

The second approach, which is the one taken here, is by using random walks. Consider a person standing at a vertex of the graph in the beginning. In each step, she chooses one of her neighbours, uniformly at random, and goes there. She continues doing this forever. This is called a *random walk* on graph $G$. One can find a random spanning tree by using the following theorem, proved independently by Broder [2] and Aldous [1].

**Theorem 1.** *Suppose that you simulate a random walk in $G$ starting from an arbitrary vertex $s$ until all vertices have been visited. For each $v \in V \setminus \{s\}$, let $e_v$ be the edge through which $v$ was visited for the first time. Then $T = \{e_v : v \in V \setminus \{s\}\}$ is a uniformly random spanning tree of $G$.*

This theorem immediately gives an algorithm for generating a random spanning tree: just simulate a random walk, storing the edges $e_v$ for all vertices $v$, until all vertices are visited. The expected running time of this algorithm is asymptotically equal to the expected number of steps the random walk takes before every vertex of $G$ is visited. The latter quantity is called the *cover time* of $G$, and is $\widetilde{O}(mn)$. Thus we get an $\widetilde{O}(mn)$ algorithm for generating a random spanning tree. There are graphs whose cover time is indeed $\widetilde{\Theta}(mn)$, so the bound on the running time is tight.

Here we give an algorithm with expected running time $\widetilde{O}(m\sqrt{n} \log 1/\delta)$ for generating a $\delta$-random spanning tree. This means that if $G$ has a total of $t$ spanning trees, and $T$ is an arbitrary spanning tree of $G$, then the probability that the algorithm generates $T$ is between $(1-\delta)/t$ and $(1+\delta)/t$. For sparse graphs, this is faster than the algorithm of Colbourn et al., which has running time $O(n^{2.376})$

To get an idea about how can we improve upon the basic simulation algorithm, let us consider a typical situation in which the cover time can be large. Suppose that the graph $G$ has two large cliques $A$ and $B$ with a long path connecting them. If the random walk starts at $A$, it is likely that it covers the whole $A$ quickly, and then to cover $B$, it needs to take the long path to $B$. It is possible that before reaching $B$, the walk reverses its direction and returns to $A$, and spend quite a lot of time in $A$ again. However, in this second visit, it does not give us any new information about the spanning tree we are going to build, because it is not visiting any vertex in $A$ for the first time. This may happen again and again, and the walk may visit $A$ (and move around) many times before eventually reaching $B$ and covering it. If we could somehow shortcut the walk's trajectory when it is visiting $A$ for the $j$-th time, $j > 1$, then we could decrease the total simulation time.

In general, let $D$ be a highly connected subgraph in $G$ (that will be covered quickly). Let $e_1, \ldots, e_k$ be the edges with exactly one endpoint in $D$. Suppose that the walk has covered $D$. The next time it enters $D$, its actual trajectory inside $D$ is not important (it does not reveal any of the $e_v$'s in Theorem 1), and the only important thing is, through which edge does it exit $D$. Of course this is a random walk, and it does not exit through a deterministic edge, so let us say it exits through $e_i$ with probability $p_i$. If we could compute the probabilities $p_1, \ldots, p_k$, then we could omit the simulation of the walk inside $D$, and when the walk enters $D$ (for the $j$-th time, $j > 1$), choose a random edge $e \in \{e_1, \ldots, e_k\}$ according to the known probability distribution, and let the walk exit $D$ through the edge $e_i$.

We will present an $\widetilde{O}(m^2 \log(1/\delta)/\sqrt{n})$ algorithm first. We will describe how to decompose $G$ into highly connected subgraphs in Section 2, explain how to compute the shortcutting probabilities in Section 3, and show how to bound the expected simulation time in Section 4. Finally, in Section 5, we will describe briefly how the running time can be improved to $\widetilde{O}(m\sqrt{n}\log(1/\delta))$.

## 2 Building Strong Decompositions

Let $G = (V, E)$ be a graph, $S \subseteq V$ and $D_1, \ldots, D_k$ be the connected components of $G - S$. Then $(S, D_1, \ldots, D_k)$ is a *strong* $(\phi, \gamma)$*-decomposition* if the following hold.

(1) $|C| \le \phi|E|$, where $C = E \setminus (\cup E(D_i))$.

(2) $|P(S)| \le \phi|V|$, where $P(S)$ is the set of vertices in $S$ that have a neighbour out of $S$.

(3) For $i = 1, \ldots, k$, the diameter of $D_i$ is at most $\gamma$.

(4) For $i = 1, \ldots, k$, $|\partial(D_i)| \le |E(D_i)|$, where $\partial(D_i)$ is the set of edges with exactly one endpoint in $D_i$, and $E(D_i)$ is the set of edges with both endpoints in $D_i$.

If we have such a decomposition, then $D_1, \ldots, D_k$ are the highly connected subgraphs discussed in the introduction. Next we show such a decomposition can be found quickly.

**Lemma 2.** *For any graph $G = (V, E)$ and $\phi$ small enough, a strong $(2\phi, O(\log|E|/\phi))$-decomposition of $G$ can be found in linear time.*

*Proof.* The algorithm uses the ball-growing technique of Leighton and Rao [7]. We will need a few definitions first. For a vertex $v$ and a nonnegative integer $j$, let

- $B(v, j)$ be the set of vertices whose distance from $v$ is at most $j$.
- $R(v, j)$ be the set of vertices whose distance from $v$ is exactly $j$.
- $R^+(v, j) = E(B(v, j + 1)) \backslash E(B(v, j))$.
- $R^-(v, j) = E(B(v, j)) \backslash E(B(v, j - 1))$.
- $t = \phi/(2 - 2\phi)$.

Now, consider the following algorithm.

```
while G is nonempty do
   choose an arbitrary vertex v and set j = 0
   while |R (v, j+1)| > t |V(B(v,j))|
      OR |R+(v, j+1)| > t |E(B(v,j))|
      OR |R-(v, j+1)| > t |E(B(v,j))| do
         j = j + 1
   Suppose that you stop at j=j(i)
   Add R(v, j(i)+1) to S and the ball B(v, j(i)) as a new component Di
   Delete S, Di, and all their incident edges
```

It is not hard to see that the algorithm can be implemented to run in linear time. Next we show that if $\phi$ is small enough, then the resulting decomposition is a strong $(2\phi, O(\log |E|/\phi))$-decomposition.

(1) Fix some $i$ and $j = j(i)$. When we build the component $D_i$, we set $D_i = B(v, j)$. We also added $R(v, j + 1)$ to $S$, so $R^-(v, j + 1) \cup R^+(v, j + 1)$ is added to $C$. The number of edges added to $C$ is

$$|R^-(v, j+1) \cup R^+(v, j+1)| \leq |R^-(v, j+1)| + |R^+(v, j+1)| \leq 2t|E(B(v, j))| = 2t|E(D_i)|.$$

If we write this inequality for all $i$ and add them up, we get $|C| \leq 2t(|E| - |C|)$, which gives $|C| \leq 2t|E|/(1 + 2t) = \phi|E|$.

(2) Fix some $i$ and $j = j(i)$. When we build the component $D_i$, we set $D_i = B(v, j)$. We also added $R(v, j+1)$ to $S$. The number of vertices added to $S$ is $R(v, j+1) \leq t|V(D_i)|$. If we write this inequality for all $i$ and add them up, we get $|S| \leq t(|V| - |S|)$, which gives $|S| \leq t|V|/(1+t) < \phi|V|$.

(3) Fix some $i$ and $j = j(i)$. We claim that the diameter of $D_i$ is at most

$$6\,\frac{1 + \ln|E|}{\ln(1+t)} = O\left(\frac{\ln|E|}{-\ln(1-\phi)}\right) = O(\log|E|/\phi),$$

where we have used $-\ln(1-\phi) = \Theta(\phi)$ for $\phi$ small enough. Assume that the diameter is larger than this. So $j > 3(1+\ln|E|)/\ln(1+t)$, thus a particular one of the conditions in the second while has been triggered more than $(1+\ln|E|)/\ln(1+t)$ times. Let $a = (1+\ln|E|)/\ln(1+t)$. If it was the first condition, then since $B(v, 0) = 1$, the ball $B(v, j)$ would have more than $(1+t)^a \geq |E|$ vertices, which is not possible. If it was the second (or the third) one, then since $|E(B(v, 1))| \geq 1$, the ball $B(v, j)$ would have more than $(1+t)^{a-1} \geq |E|$ edges, which is impossible as well.

(4) For all $i$ with $|\partial(D_i)| > |E(D_i)|$, add $V(D_i)$ to $S$. The size of $C$ becomes at most twice, because for each $i$, we add $|E(D_i)|$ new edges to $C$, and $C$ already contained $|\partial(D_i)|$ edges that connect it with $D_i$. Moreover, the size of $P(S)$ do not change, because the vertices added to $S$ do not have a neighbour out of $S$. The diameters of other $D_i$'s do not change. Hence the result is a strong $(2\phi, O(\log|E|/\phi))$-decomposition. $\square$

## 3 Computing the Shortcutting Probabilities

Fix a $D = D_i$. For $v \in D$ and $e \in \partial D$, let $P_v(e)$ be the probability that a random walk starting at $v$, exits $D$ through $e$. Assuming these probabilities are known, we shortcut our random walk as follows. Before the walk has visited all vertices of $D$, we do not do any shortcutting. Suppose that the walk has covered $D$. Now, assume that in some step the walk enters $D$ through some vertex $v \in D$. In the next step, instead of following the usual definition of a random walk, we choose an edge $e \in \partial D$ according to the probability distribution imposed by $\{P_v(e) : e \in \partial D\}$. Suppose that $e = uu'$ with $u \in V(D)$. Then the "modified" random walk jumps from $v$ to $u$ in the next step, and traverses the edge

$uu'$ in the next step. The rest is the same as the basic simulation algorithm: we continue until all vertices of $G$ have been visited.

We will show how to compute approximate values for $P_v(e)$ quickly, using the known connection between random walks and electrical networks, and a recently developed efficient linear system solver.

**Lemma 3.** *Given a strong* $(\phi, \gamma)$*-decomposition of* $G$*, we can compute multiplicative* $(1+\epsilon)$*-approximations of all values* $P_v(e)$ *in time* $\widetilde{O}(\phi m^2 \log(1/\epsilon))$*.*

*Proof.* Fix a $D = D_i$ and $e = uu' \in \partial D$ with $u \in V(D)$. Build a multigraph $D'$ from $D$ as follows. Add vertices $u'$ and $u^*$ to $D$, where $u^*$ is a dummy vertex. Also add the edge $uu'$. For every $ww' \in \partial(D) \setminus \{e\}$ with $w \in V(D)$, add an edge $wu^*$. (Notice that we might have $w' = u'$.) Observe that for every $v \in V(D)$, $P_v(e)$ is precisely the probability that a random walk in $D'$, started at $v$, hits $u'$ before $u^*$.

Now, if we treat $D'$ as an electrical circuit with unit resistance on each edge, in which we impose voltage +1 at $u'$ and 0 at $u^*$, then the voltage achieved at $v$ is equal to $P_v(e)$ (see, e.g., [8]). The formulas describing this circuit can be written as a linear system with the voltages at each vertex being its variables, and Kirchhoff's current law at each vertex being its equations. We put two external current generators so that an external current of 1 is going into vertex $u'$, and an external current of -1 is going into vertex $u^*$. This system can be written in the form $Ax = b$, where $A$ is a symmetric, diagonally dominant $|V(D')| \times |V(D')|$ matrix and has $|V(D')| + |E(D')|$ nonzero elements. Thus multiplicative $(1+\epsilon)$-approximations for the voltages can be found in time $\widetilde{O}(|E(D')| \log(1/\epsilon))$ using the linear system solver of Spielman and Teng [6].

We should solve the linear systems for all $e \in C$, so the total running time is

$$\widetilde{O}\left(|C| \sum |E(D_i')| \log 1/\epsilon\right) = \widetilde{O}\left(|C| \sum |E(D_i)| \log 1/\epsilon\right) = \widetilde{O}\left((\phi m) m \log 1/\epsilon\right),$$

where we have used property (1) of strong decompositions $|C| \leq \phi m$, and also property (4) of strong decompositions to get $|E(D_i')| = O(|E(D_i)|)$. □

## 4 Bounding the Expected Simulation Time

In the previous two sections, we explained how our simulation works. In this section we bound the expected simulation time. The steps in our modified random walk are of three type:

1. The steps in which we traverse an edge of $C$.
2. The steps in which we traverse an edge of $D_i$, while $D_i$ has not been covered yet.
3. The steps in which we "jump" from a vertex $v \in D_i$ to a vertex $u \in D_i$ (so that in the next step we traverse some edge $uu' \in \partial D$), after $D_i$ has already been covered.

Notice that $C$ contains at most a $\phi$ fraction of the edges of $G$, by property (1) of strong decompositions. The basic random walk (without shortcuts) takes an expected number of $O(mn)$ steps before covering the whole graph, and so, intuitively, in at most a $\phi$ fraction of those steps, we would traverse an edge in $C$. (To make this precise, we need our starting vertex to be chosen according to the stationary distribution of the Markov chain associated with the random walk; but for simplicity we avoid this.) Thus, the expected number of steps of type 1 in our simulation is $O(\phi mn)$. Notice that every step of type 3 is followed by a step of type 1. Hence the expected number of steps of type 3 is $O(\phi mn)$ as well, and we just need to bound the expected number of type 2 steps we take.

For bounding the expected number of steps of type 2, fix a $D = D_i$. It is known [3] that the expected cover time of a graph $H$ with diameter $\ell$ is $O(\ell|E(H)|\log|V(H)|)$. Our walk is not really a random walk in $D$, since it may go out of $D$. However, using some random walk techniques, which we omit here (see the proof of Lemma 6 in Kelner-Madry's paper [5]), it can be shown that the expected number of steps taken inside $D$ before it is covered is $\widetilde{O}(|E(D)|\gamma(D))$. Here $\gamma(D)$ is the diameter of $D$, which is at most $\gamma$ by property (3) of strong decompositions. In the proof, the fact $|\partial D| \leq |E(D)|$ is used, implied by property (4) of strong decompositions (the necessity of such a condition makes sense, because we want to look at those steps of the random walk on $G$ that are taken inside $D$, and for this to be similar to a real random walk on $D$, we do not want it to go out of $D$ most of the time).

Consequently, the expected total number of steps in our walk, which asymptotically equals the running time of the simulation, is bounded by $O(\phi mn) + \widetilde{O}(m\gamma) + O(\phi mn) = \widetilde{O}(m\gamma + \phi mn)$.

Now, we present an $\widetilde{O}(m\log(1/\delta)/\sqrt{n})$ algorithm to generate a $\delta$-random spanning tree. Set $\phi = 1/\sqrt{n}$. By Lemma 2, a strong $(1/\sqrt{n}, \widetilde{O}(\sqrt{n}))$-decomposition of $G$ can be found in linear time. Then, by Lemma 3, $(1+\delta/mn)$-approximations of the probabilities $P_v(e)$ can be found in time $\widetilde{O}(m^2\log(1/\delta)/\sqrt{n})$. It can be verified (see Lemma 10 in [5]) that taking the value $\delta/mn$ for $\epsilon$ ensures that our approximate simulation is good enough, and the resulting tree is $\delta$-random. Then, we can run the simulation, whose expected running time is $\widetilde{O}(m\sqrt{n})$ by the discussion above. Therefore, the expected total running time of the algorithm is $\widetilde{O}(m^2 \log(1/\delta)/\sqrt{n})$ as desired. In the next section, we describe how to improve the running time to $\widetilde{O}(m\sqrt{n}\log(1/\delta))$.

## 5 Obtaining a Faster Algorithm

The bottleneck of the algorithm presented in the previous sections is finding the probabilities $P_v(e)$. Recall that in the shortcutting procedure, for each $i$, when we enter some $D_i$ (after we have already covered $D_i$), we choose a random edge $uu' \in \partial D$ with $u \in V(D_i)$ and jump to $u$ and traverse the edge $uu'$. What if instead of doing this, we jump directly to $u'$? This may cause problems in building our spanning tree, because it may be the first time that we are visiting $u'$, and if so, then the edge $e_{u'}$ in Theorem 1 is undefined.

Let us get into details. For a given $i$, some $v \in V(D_i)$, and $u \notin D_i$, define $Q_v(u)$ to be the probability that $u$ is the first vertex out of $D_i$ that is reached by a random walk in $G$ that starts at $v$. Note that $Q_v(u) = 0$ if $u \notin P(S)$, so we only need to consider $u \in P(S)$. We show that these probabilities can be computed more quickly than the values $P_v(e)$.

**Lemma 4.** *Multiplicative* $(1+\epsilon)$*-approximations for all values* $Q_v(u)$ *can be computed in time* $\widetilde{O}(\phi mn \log(1/\epsilon))$.

*Proof.* Fix $D = D_i$ and $u \in P(S)$. Let $D'$ be the multigraph obtained from $G[D \cup P(S)]$ by identifying all vertices in $P(S) \setminus \{u\}$, and naming the resulting vertex $u^*$. Note that $|E(D')| = |E(D)| + |\partial(D)| \leq 2|E(D)| = O(|E(D)|)$ by property (4) of strong decompositions. Then $Q_v(u)$ is precisely the probability that a random walk in $D'$ starting from $v$ hits $u$ before $u^*$. We can find multiplicative $(1+\epsilon)$-approximations for all such probabilities in time $\widetilde{O}(|E(D')|\log(1/\epsilon))$ by treating the graph as an electrical network and using the linear system solver of [6] (as in the proof of Lemma 3).

We need to do this for all $u \in P(S)$, so we can find multiplicative $(1+\epsilon)$-approximations for all values $Q_v(u)$ in time

$$\widetilde{O}(|P(S)| \sum |E(D_i)| \log(1/\epsilon)) = \widetilde{O}(\phi nm \log(1/\epsilon)),$$

where we have used the property (2) of strong decompositions $|P(S)| \leq \phi n$. □

Now, we present an $\widetilde{O}(m\sqrt{n}\log(1/\delta))$ algorithm for generating a $\delta$-random spanning tree. Set $\phi = 1/\sqrt{n}$. By Lemma 2, a strong $(1/\sqrt{n}, \widetilde{O}(\sqrt{n}))$-decomposition of $G$ can be found in linear time. Then, by Lemma 4, $(1 + \delta/n^5)$-approximations of the probabilities $Q_v(u)$ can be found in time $\widetilde{O}(m\sqrt{n}\log(1/\delta))$.

Now, we run a simulation of the random walk described in the beginning of this section. We simulate ($\delta$-approximately) this walk until it visits all vertices of $G$, which takes an expected number of $\widetilde{O}(m\sqrt{n})$ steps. Having done this, let $e_v$ be, as in Theorem 1, the first edge through which we have entered $v$ for the first time. Note that $e_v$ may be undefined for some vertices, but this can happen only if $v \in P(S)$. So, we need to define some of the $e_v$'s manually. To describe precisely how to do this, we need to work with directed rooted trees, which we have tried to avoid in this article. Thus, we will give a procedure which is the undirected analogous of what is really done, and the details can be found in the proof of Lemma 15 of [5]. Let $F = \{e_v : v \notin P(S) \cup \{s\}\}$. Then $F$ is a forest with $|P(S)|$ connected components. Contract each of its connected component into a single vertex. The remaining graph has $|P(S)|$ vertices, and $|P(S)| \leq \phi n$ by property (2) of strong decompositions. Now, using the algorithm of Colbourn et al. [4], build a random spanning tree $T$ of this graph in time $O((\phi n)^{2.376}) = O(n^{1.188})$. Return the tree $T \cup F$ as a $\delta$-random spanning tree of $G$. Therefore, the expected total running time of the algorithm is $\widetilde{O}(m\sqrt{n}\log(1/\delta))$.

## References

[1] D. J. Aldous, A random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics*, 1990.

[2] A. Broder, Generating random spanning trees. In *FOCS*, 1989.

[3] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *FOCS*, 1979.

[4] C. J. Colbourn, W. J. Myrvold, and E. Nuefeld, Two algorithms for unranking arborescences. *Journal of Algorithms*, 1996.

[5] J. A. Kelner, A. Madry, Faster generation of random spanning trees. arXiv:0908.1448v1[cs.DS], 2009.

[6] D. A. Spielman and S.-H. Teng, Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. arXiv:cs/0607105[cs.NA], 2006.

[7] T. Leighton and S. Rao, Multicommidity max-flow min-cut theorems and their usage in designing approximation algorithms. *Journal of the ACM*, 1999.

[8] L. Lovász, Random walks on graphs: A survey. http://www.cs.yale.edu/publications/techreports/tr1029.pdf.