# Combinatorics and Computation Notes

Marissa Masden

10/16/2019

## 1 Wilson's Algorithm

The fastest known algorithm to generate a spanning tree uniformly from all spanning trees of a given graph $G$ is *Wilson's Algorithm*, which is given by the following general steps:

**Wilson's Algorithm:**

1. Pick a root vertex $r$, not necessarily uniformly from $G$, and set $T_0 = \{r\}$.
2. Repeat the following:
   (a) Select a random vertex $v_i$ uniformly from vertices not in $T_i$.
   (b) Perform a *loop-erased random walk* (LERW) in $G$ until we find a path $P_i$ whose first vertex is $v_i$ and whose last vertex is in $T_i$.
   (c) Set $T_{i+1} = T_i \cup P_i$.
   (d) Do until the vertices of $T_k$ span the vertices of $G$.

While we have not yet defined a loop-erased random walk, it should be clear that this process generates a spanning tree of $G$. What is not clear is that this spanning tree should be uniform, which we must prove. We begin by more precisely defining a loop-erased random walk:

A **loop-erased random walk** in $G$ starting at $v$ is a walk $(v_0, v_1, v_2, ..., v_k, ...)$ with first vertex $v_0 = v$ which is generated by the following steps:

1. Choose $v_k$ uniformly from the neighbors of $v_{k-1}$ and add it to the end of the walk.
2. If $v_k$ is a vertex which is already in the walk as some $v_j$ for $j < k$, delete the vertices $v_{j+1}, ..., v_k$ from the walk, and resample the walk, starting at sampling for $v_{i+1}$.

Recall that in a finite connected graph the expected hitting time of every vertex is finite when following a random walk (in fact, random walks on finite graphs are positive recurrent). As a loop-erased random walk can be generated by removing subwalks from any standard random walk, this algorithm will terminate in finite time with probability 1, with total running time dependent on the maximum expected hitting time of any vertex in the graph, which we will not discuss here.

*Claim:* Wilson's algorithm selects a spanning tree uniformly from all spanning trees of $G$.

*Proof:* Consider placing a stack of infinitely many "cards" on each vertex of $G$ except the root. We "color" the $i$th card of each stack by color $i$. Let each card also have a neighboring vertex on it, chosen uniformly and independently from the neighbors of that stack's vertex. Think about each card as recording what layer it is in, and providing an edge between that vertex and one of its neighbors.

If the top cards on each stack define a tree in this manner, then we are done.

Otherwise, there are some cycles. Consider the algorithm given as follows:

1. Select a cycle $C$ given by some of the cards on the top of the stacks.
2. **Pop** the cycle $C$ by removing the cards corresponding to edges in $C$ from the top of their stacks.
3. Repeat until there are no cycles.

This algorithm is equivalent to Wilson's algorithm: Wilson's algorithm randomly finds cycles to pop and pops them. However, we still must verify that this new algorithm generates spanning trees uniformly. In particular, if we want this to work we need it to be irrelevant which cycle we choose to pop first.

*Lemma:* The resulting tree is independent of the order in which cycles are popped.

*Proof of Lemma:* Note each *colored* cycle is only popped once, even if a given cycle could be popped multiple times. Consider a colored cycle $C$ which is popped when it is color $k$. That is, we have popped the following cycles in order:

$$C_1, C_2, ..., C_k = C$$

Suppose that the cycle $\tilde{C}$ was popped instead of $C_1$. Can $C$ still be popped? We will see that it can be - we will construct a sequence of cycles which can be

popped leading us to $C$.

We consider two cases. In the easy case, $\tilde{C}$ is disjoint from $C_1, ..., C_k$. In that case, we may pop the cycles in this order: $\tilde{C}, C_1, ..., C_k$.

In the harder case, suppose that $\tilde{C}$ is not disjoint from $C_1, ..., C_k$. Let $C_i$ be the first cycle with a common vertex with $\tilde{C}$. We will see that $\tilde{C} = C_i$. In fact, if not, there is some vertex $w$ with different successors in the two cycles.

Since by our assumption $w$ is not in $C_1, ..., C_{i-1}$, then in fact its card has not yet been removed: it is the same color in $\tilde{C}$ and $C_i$. This is a contradiction: $w$'s card must tell us to go to the same place in each cycle, and cannot have different successors in $C_i$ and $\tilde{C}$.

Therefore, $\tilde{C} = C_i$ and furthermore $C_i$ is disjoint from $C_1, ..., C_{i-1}$, so we may choose to pop it first. The following order permits us to still choose to pop $C = C_k$:

$$\tilde{C}, C_1, ..., C_{i-1}, C_{i+1}, ..., C_k$$

Therefore, our lemma is proven. Lastly, we wish to see that the cycle-popping algorithm gives us trees uniformly. However, that was not discussed in this class period.