Mathematical Analysis of Wilson's Algorithm for Uniform Spanning Trees

Abstract

We present a mathematical analysis of Wilson's algorithm, which generates uniform spanning trees using loop-erased random walks. We derive the hitting probabilities, expected path lengths, and convergence properties, with particular focus on grid graphs for maze generation. Our results demonstrate why the first path typically exhibits $\Theta(n \log n)$ complexity while subsequent paths converge significantly faster.

1 Preliminaries

1.1 Definitions

Let $G = (V, E)$ be an undirected graph where:

$$V = \{(x, y) \mid 0 \leq x < w, 0 \leq y < h\} \tag{1}$$
$$E = \{((x, y), (x', y')) \mid |x - x'| + |y - y'| = 1\} \tag{2}$$

A spanning tree $T = (V, E')$ satisfies:

$$V(T) = V(G) \tag{3}$$
$$|E'| = |V| - 1 \tag{4}$$
$$T \text{ is connected} \tag{5}$$

A uniform spanning tree (UST) is drawn from the uniform distribution over all spanning trees of G.

1.2 Loop-Erased Random Walk

For a random walk $(X_0, X_1, X_2, \ldots)$ with $X_0 = v$, its loop-erased version $(Y_0, Y_1, \ldots)$ is:

$$Y_0 = X_0 = v \tag{6}$$
$$\sigma(i) = \max\{j < i : X_j \neq X_i\} \tag{7}$$
$$Y_i = X_{\sigma^i(0)} \tag{8}$$

2 Wilson's Algorithm

2.1 Algorithm Formulation

Given a graph $G = (V, E)$:

Algorithm 1 Wilson's Algorithm

```
T ← {v_0} for arbitrary v_0 ∈ V
U ← V \ {v_0}
while U ≠ ∅ do
    Select u ∈ U
    P ← LERW(u, T)   ▷ Loop-erased path from u to T
    T ← T ∪ P
    U ← U \ P
end while
return T
```

2.2 Properties

Theorem 1 (Uniformity)**.** *Wilson's algorithm generates a spanning tree T with probability:*

$$\mathbb{P}(T) = \frac{1}{|ST(G)|} \tag{9}$$

where $ST(G)$ is the set of all spanning trees of G.

Lemma 2. *For any unvisited vertex u and any spanning tree T_u of the subgraph induced by $T \cup \{u\}$:*

$$\mathbb{P}(LERW(u, T) = T_u \setminus T) = \frac{1}{d(u)} \tag{10}$$

where $d(u)$ is the degree of u.

3 Mathematical Analysis

3.1 Hitting Probabilities

For a vertex $v_i \notin T$ and $T \subseteq V$, the hitting probability is:

$$h(v_i, T) = \mathbb{P}(\exists t : X_t \in T \mid X_0 = v_i) = 1 \tag{11}$$

For the first path ($|T| = 1$), the expected hitting time is:

$$\mathbb{E}[H_{v_i,T}] = \frac{|E|}{|V|} \cdot R_{\text{eff}}(v_i, T) \tag{12}$$

where R_{eff} is the effective resistance.

3.2 First Path Analysis

Theorem 3. *For a grid graph with $n = w \times h$ vertices, the expected number of steps for the first path is:*

$$\mathbb{E}[\textit{first path steps}] = \Theta(n \log n) \tag{13}$$

Sketch. For a single target vertex v_0 in a 2D grid:

$$R_{\text{eff}}(v_i, v_0) \approx \frac{1}{\pi} \ln d(v_i, v_0) + O(1) \tag{14}$$

$$\mathbb{E}[H_{v_i,v_0}] \approx \frac{2|E|}{|V|\pi} \ln d(v_i, v_0) \tag{15}$$

Integrating over all possible distances in a grid:

$$\mathbb{E}[\text{steps}] \approx \frac{2|E|}{|V|\pi} \int_1^{\sqrt{n}} \ln r \cdot 2\pi r \, dr = \Theta(n \log n) \tag{16}$$

□

3.3 Loop Erasure Effect

For a random walk of length L, the expected length after loop erasure is:

$$\mathbb{E}[\text{LERW length}] \approx \sqrt{L} \tag{17}$$

Therefore, for the first path with $L = \Theta(n \log n)$:

$$\mathbb{E}[\text{first path LERW length}] = \Theta(\sqrt{n \log n}) \tag{18}$$

3.4 Convergence Rates

As the algorithm progresses, $|T|$ increases, and hitting probabilities change. For $|T| = k$:

$$\mathbb{P}(\text{hit } T \text{ within } m \text{ steps}) = 1 - \left(1 - \frac{k}{n}\right)^m \tag{19}$$

This yields expected path lengths:

$$\mathbb{E}[\text{first path}] = \Theta(n \log n) \tag{20}$$

$$\mathbb{E}[\text{middle paths}] = \Theta(n^{1/2} \log n) \tag{21}$$

$$\mathbb{E}[\text{last paths}] = \Theta(\log n) \tag{22}$$

Corollary 4. *The total expected runtime of Wilson's algorithm on a grid graph is dominated by the first few paths, giving $\Theta(n \log n)$ overall complexity.*

4 Conclusion

Wilson's algorithm provides a uniformly distributed spanning tree through loop-erased random walks. The first path dominates the runtime with $\Theta(n \log n)$ complexity, while subsequent paths exhibit progressively faster convergence. This mathematical structure explains both the algorithm's correctness and its observed performance characteristics in maze generation applications.

A Implementation

The full OCaml implementation of Wilson's algorithm described in this paper is available in our GitHub repository:

`https://github.com/username/maze-generation`

The repository contains:

- `wilson.ml` - Implementation of Wilson's algorithm
- `maze.ml` - Core maze data structures
- `animate.ml` - Visualization components for the algorithm
- Benchmark tools and test suites