Wilson's algorithm

Julien Zucchet

Selected Topics in Mathematics, period 4 2020-2021

Contents

1 Introduction

There are many possible ways to generate random spanning trees from a given graph: for instance, choose a vertex, start a random walk from this vertex, and when one meets a vertex for the first time add the edge which led to this vertex to the tree until every vertex has been visitied (it is the algorithm presented by Aldous in his article [2]). However, some work only for a certain type of graphs (for instance directed or undirected) and their performance in terms of execution time can vary a lot. Wilson, in its original article [5], published an algorithm working for both directed and undirected, and with an execution approximating the hitting time.

2 Definitions

Here are some definitions that will be used in this document.

2.1 Undirected graphs

Definition 1. *An undirected graph $G = (V, E)$ is:*

- ***simple*** *when it has no loop or multiple edges.*
- ***connected*** *when for every $x, y \in V$ there exists a path between x and y made of edges in E.*

Let $\mathcal{T} = (V', E')$ with $V' \subset V$ and $E' \subset E$. $\mathcal{T}$ is:

- ***a tree*** *when it is simple, connected, and has no loops.*
- ***a spanning tree*** *when it is a tree and $V' = V$.*

2.2 Directed graphs

The notion of tree can be transposed to a directed graph with the notion of arborescence: let $G = (V, E)$ be a directed graph.

Definition 2. *Let $\mathcal{A} = (V', E')$ be a subgraph of G, and fix $v \in V'$ a root. $\mathcal{A}$ is an* ***arborescence*** *if for all $w \in V' \backslash \{v\}$, there exists a unique directed path from w to v in $\mathcal{A}$, and it is a* ***spanning arborescence*** *if $V' = V$.*

3 Study of self-avoiding random walks

In this section will be presented a short study of self-avoiding walks which will be useful for the theoretical study of Wilson's algorithm. The references for this part are the lecture notes and the book of Lawler [4].

Fix $\overline{A} = A \cup \delta A$ a finite subset of $\mathbb{Z}^2$. A will be the domain for the random walk, and δA the boundary of the domain. Let $(X_n)_{n \in \mathbb{N}}$ be a Markov chain on $\overline{A}$, with a transition matrix P, and the states in δA are set absorbing. Let $x \in A$. Denote the set of walks starting from x and ending for the first time in δA by:

$$\mathcal{X}_A(x, \delta A) = \bigcup_{y \in \delta A} \mathcal{X}_A(x, y)$$

and denote the set of self-avoiding walks starting from x and ending for the first time in δA by:

$$\mathcal{W}_A(x, \delta A) = \bigcup_{y \in \delta A} \mathcal{W}_A(x, y)$$

where for $y \in \delta A$ $\mathcal{X}_A(x, y)$ is the set of walks starting from x and ending for the first time in y and $\mathcal{W}_A(x, y)$ the set of self-avoiding walks starting from x and ending for the first time in y.

Definition 3. *Let $x, y \in A$. Define the* ***Green's function*** *by:*

$$G_A(x,y) = \mathbb{E}_x\left[\sum_{n=0}^{+\infty} \delta_{X_n,y}\right]$$

where $\delta_{.,.}$ is the Kronecker symbol meaning that $\delta_{x,y} = 1$ if and only if $x = y$, otherwise $\delta_{x,y} = 0$. The Green's function represents the average number of visits to y when starting the chain from x. Note that if the probability to reach δA is strictly positive this number is finite almost surely.

Definition 4. *Let $\omega = [\omega_0, \dots, \omega_k]$ be a path of finite length in A. The (chronological)* ***loop-erasure*** *of ω $LE(\omega)$ is a path obtained by erasing the cycles in ω in the order they appear. More precisely, this operation can be described by the following steps:*

- *Compute the set $A = \{l \in [\![0,k]\!], \omega_l$ is used twice or more in $\omega\}$*
- *If $A \neq \emptyset$, compute $l = \min A$.*
- *Compute $n = \max\{i \in [\![l+1,k]\!], \omega_i = \omega_l\}$.*
- *Replace ω by $[\omega_0, \dots, \omega_l, \omega_{n+1}, \dots, \omega_k]$ if $n+1 \neq k$, otherwise replace ω by $[\omega_0, \dots, \omega_l]$.*
- *Repeat the steps until $A = \emptyset$.*

Definition 5. *The p measure $p(x,y)$ of a vertex $[x,y]$ is given by the weight induced by the transition matrix P on the vertex $[x,y]$. The p measure of a path $\omega = [\omega_0, \dots, \omega_k]$ is given by:*

$$p(\omega) = \prod_{i=0}^{k-1} p(\omega_i, \omega_{i+1})$$

Definition 6. *The* ***loop-erased random walk*** *from x to δA is the probability measure $\hat{p}$ on $\mathcal{W}_A(x, \delta A)$ given by*

$$\hat{p}(\eta) = \hat{p}_A(\eta) = \sum_{\omega \in \mathcal{X}(x,\delta A), LE(\omega) = \eta} p(\omega)$$

Proposition 1. *Let $\eta = [\eta_0, \dots, \eta_k] \in \mathcal{W}_A(x, \delta A)$ (described in terms of vertices) and denote by A_k the set $A \backslash \{\eta_0, \dots, \eta_{j-1}\}$. Then*

$$\hat{p}_A(\eta) = p(\eta) \prod_{j=0}^{k-1} G_{A_j}(\eta_j, \eta_j)$$

Proof. Let $\omega = [\omega_0, \dots, \omega_m] \in \mathcal{X}_A(x, \delta A)$ such that $LE(\omega) = \eta$. Then ω can be uniquely described by the following decomposition:

$$\omega = l_0 \oplus [\eta_0, \eta_1] \oplus l_1 \oplus ... \oplus [\eta_{k-1}, \eta_k]$$

where $[\eta_i, \eta_{i+1}]$ is the edge between the vertices η_i, $\oplus$ is the concatenation operator and l_i are *maximal* loops on η_i. The maximality of the loops means that it is the longest loop going from η_i to η_i in ω. In more formal terms, l_i can be written $[\omega_a, \dots, \omega_b]$ where $a = \min\{j \in [\![0,m]\!], \omega_j = \eta_i\}$ and $b = \max\{j \in [\![0,m]\!], \omega_j = \eta_i\}$. The uniqueness comes from the maximality of the loops. For a given l_i, the p measure of possible loops is given by $G_{A_i}(\eta_i, \eta_i)$, where $A_i = A \backslash \{\eta_0, \dots, \eta_{i-1}\}$. So the loop-erased measure of η is:

$$\begin{aligned}
\hat{p}_A(\eta) &= G_A(\eta_0,\eta_0) \times p(\eta_0,\eta_1) \times G_{A_1}(\eta_1,\eta_1) \times \cdots \times G_{A_{k-1}}(\eta_{k-1},\eta_{k-1}) \times p(\eta_{k-1},\eta_k) \\
&= \left(\prod_{j=0}^{k-1} p(\eta_j, \eta_{j+1})\right) \prod_{j=0}^{k-1} G_{A_j}(\eta_j,\eta_j) \\
&= p(\eta) \prod_{j=0}^{k-1} G_{A_j}(\eta_j,\eta_j)
\end{aligned}$$

□

Proposition 2. *If $V = \{x_1, \dots, x_k\} \subset A$ let*

$$F_V(A) = \prod_{j=1}^{k} G_{A_j}(x_j, x_j)$$

where $A_j = A \backslash \{x_1, \dots, x_{k-1}\}$. If $V = A$ denote $F_A(A) = F(A)$ and for a self-avoiding walk $\eta = [\eta_0, \dots, \eta_k]$ then $F_\eta(A) = F_{V'}(A)$ with $V' = \{\eta_0, \dots, \eta_k\}$. Then this quantity does not depend on the ordering of the vertices.

Proof. The idea is to prove the result for transpositions, and extend it to permutations using the fact that any permutation can be written as a product of transpositions. See the lecture notes or [4] for the proof. □

Proposition 3. *If $\eta = [\eta_0, \dots, \eta_k] \in \mathcal{R}_A(x, \delta A)$ then*

$$\hat{p}(\eta) = p(\eta) F_\eta(A)$$

Proof. This result is obtained by using propositions 1 and 2. □

4 Wilson's algorithm on undirected graphs

The proofs in this section are mainly coming from Lawler's book [4].

Let $G = (V, E)$ be a finite, simple (no loop or multiple edges), connected and undirected graph. Let $\deg(v)$ denote the degree of a vertex $v \in V$. Associate with this graph a random walk which, from a vertex v, randomly goes to one of its $\deg(v)$ neighbors.

4.1 The algorithm

Wilson's algorithm allows to construct a **uniform spanning tree** (that is a spanning tree chosen from the uniform distribution over the set of spanning trees of G). The interesting fact about this algorithm is that it does not require to know the total number of spanning trees, and it even computes this number.

Write $V = \{x_0, \dots, x_n\}$ for some $n \in \mathbb{N}$ and set $V_0 = \{x_0\}$ and $E_0 = \emptyset$. The definition of the algorithm is recursive. Let $k \in \mathbb{N}$ and assume $V_k \subsetneq V$ and E_k constructed. Let $j = \min\{i \in [\![0, n]\!], x_i \notin V_k\}$. Then start a random walk from x_j until it reaches V_k. That gives a path $\omega = [e_0, \dots, e_l]$. Let $\eta = LE(\omega)$ be the (chronological) loop-erasure of ω. Add the edges of η to E_k to get E_{k+1}, and add the vertices adjacent to the edges of η to V_k to get V_{k+1}. If $V_{k+1} = V$ stop, and the spanning tree is given by E_{k+1}, otherwise repeat the procedure.

Proposition 4. *The algorithm above produces a spanning tree.*

Proof. Assume that $(V_0, \dots, V_k)$ and $(E_0, \dots, E_k)$ are two increasing sequences produced by the algorithm above. Let's prove by induction that for each $l \in [\![0, k]\!]$, $\mathcal{T}_l = (V_l, E_l)$ is a tree.

- $(x_0, \emptyset)$ is a tree so the property is true for $l = 0$.
- Let $l \in [\![0, k-1]\!]$. Suppose that $\mathcal{T}_l$ is a tree. Let's first prove that $\mathcal{T}_{l+1}$ is connected. Let $x, y \in V_{l+1}$. If $x, y \in V_l$ then as $\mathcal{T}_l$ is connected there exists a path made of edges from $E_l \subset E_{l+1}$ which goes from x to y. If $x, y \in V_{l+1} \backslash V_l$ then x and y are part of the path added to E_l to produce E_{l+1} in the algorithm, so there is a path going from x to y. If $x \in V_{l+1}$ and $y \in V_l$, let z be the first vertex of V_l found in the algorithm. With the same argument than before, there exists a path going from x to z. As $\mathcal{T}_l$ is connected there is also a path going from z to y so by concatenating the two paths there exists a path in $\mathcal{T}_{l+1}$ going from x to y. The last case is obtained by symmetry (the graph is undirected). So $\mathcal{T}_{l+1}$ is connected. Furthermore, $\mathcal{T}_l$ has no loops, and E_{l+1} is obtained by adding a loop-erased path (so without loop), and every of the vertices of this path except the last one are in $V_{k+1} \backslash V_k$ so it can not add any loop in $\mathcal{T}_{l+1}$. So $\mathcal{T}_{l+1}$ has no loops and is a tree.

By induction for every $l \in [\![0, k]\!]$ T_l is a tree.
To conclude, as the algorithm stops when $V_k = V$, then the result of the algorithm $\mathcal{T}_k$ is a tree and verifies $V_k = V$ so $\mathcal{T}_k$ is a spanning tree. □

The next step is to prove that every spanning tree of G can be found by the algorithm, and that every spanning tree $\mathcal{T}$ has the same probability to be randomly chosen by the algorithm. These two results combined give that the output of the algorithm is sampled from the uniform distribution over the set of the spanning trees of G. The interesting fact is that the computations will also give the probability for each spanning tree to be chosen, so by extension (as the distribution is uniform) will give the number of spanning trees of G.

4.2 Probabilistic perspective

The interesting facts about this algorithm is that it:

1. Shows that all spanning trees of G have the same probability to be chosen by the algorithm
2. Computes the probability (which is the same for every spanning tree)
3. Computes the actual number of spanning trees: if every spanning tree has the same probability p to be chosen, then they follow an uniform distribution and the total number of spanning trees is given by $1/p \in \mathbb{N}$. A combinatorial problem (computing the number of spanning trees for a given graph) is then solved with a probabilistic approach which was not obvious *a priori*.

Proposition 5. *Let $\mathcal{T}$ be a spanning tree of G. The probability that $\mathcal{T}$ is chosen in Wilson's algorithm is*

$$\left(\prod_{j=1}^{n} \deg(x_j)\right)^{-1} \prod_{j=1}^{n} G_{V_j}(x_j, x_j)$$

where $V_j = V \backslash \{x_1, \dots, x_{j-1}\}$ for $j \in [\![1, n]\!]$.

Remark: remember that the quantity $\prod_{j=1}^{n} G_{V_j}(x_j, x_j)$ does not depend of the order of the x_js, so the probability above does not depend of the ordering of the vertices. Furthermore, this quantity will always be the same, no matter the type of random walk used. However the other term will vary, and the spanning trees with the biggest weights (product of the weights of its edges) could be drawn more often, depending on the random walk used.

Before proving proposition 5, one may establish a useful property that will be used for the proof.

Property 1. *Let $k \in [\![1, n]\!]$, V_{k-1} be a set of vertices created during the algorithm such that $V_{k-1} \neq V$ (which means that the algorithm has not stopped yet and that there are vertices that have not been explored yet) and $x \in V \backslash V_{k-1} \neq \emptyset$. Then there exists a unique self-avoiding walk η^k in $\mathcal{T}$ that starts at v, ends in V_{k-1} and whose vertices are in $V \backslash V_{k-1}$ ($\neq \emptyset$ by assumption) except for the last one.*

Proof. (of property 1)

- **Existence:** as $\mathcal{T}$ is a spanning tree, there exists a path $[\alpha_0, \dots, \alpha_l]$ starting at $\alpha_0 = x$ and ending in $\alpha_l \in V_{k-1}$. One can set $m = \min\{j \in [\![0, l]\!], \alpha_j \in V_{k-1}\}$ and the path $[\alpha_0, \dots, \alpha_m]$ satisfies the condition.
- **Uniqueness:** suppose there exists two different paths $\alpha = [\alpha_0, \dots, \alpha_l]$ and $\beta = [\beta_0, \dots, \beta_i]$ such that $\alpha_0 = \beta_0 = x$, $\alpha_l = \beta_i \in V_{k-1}$ and satisfying the condition on the vertices. Then the path $[\alpha_0, \dots, \alpha_l, \beta_{i-1}, \dots, \beta_0]$ is a path in $\mathcal{T}$ and is a non-trivial loop. That is impossible because $\mathcal{T}$ is a tree so it has no loop.

□

Proof. (*of proposition 5*)

Let $\mathcal{T}$ be a spanning tree of G and let's compute the probability that it is the output of the algorithm. Set $V_0 = \{x_0\}$ and $E_0 = \emptyset$. This proof is recursive: let $k \in \mathbb{N}^*$ and assume that $V_{k-1} \neq V$ (which means that the algorithm has not stopped yet and that there are vertices that have not been explored yet). Let v be the vertex which is not in V_{k-1} and has the minimal number, that is $v = x_m$ with $m = \min\{l \in [\![0, n]\!], x_l \notin X_{k-1}\}$. Then according to property 2 there exists a unique self-avoiding walk η^k in $\mathcal{T}$ that starts at v, ends in V_{k-1} and whose vertices are in $V \backslash V_{k-1}$ ($\neq \emptyset$ by assumption) except for the last one. It can be seen as "the" walk from x to V_{k-1}. V_k is obtained by adding the vertices of η^k to V_{k-1} and E_k the edges of η^k to E_{k-1}. With this method, $\mathcal{T}$ has been uniquely described as a sequence of self-avoiding walks $[\eta^1, \dots, \eta^k]$, so the probability of choosing $\mathcal{T}$ is

$$\prod_{j=1}^{k} \left(p(\eta^j) F_{\eta^j}(A \backslash V_{j-1}) \right)$$

But

$$\prod_{j=1}^{m} p(\eta^j) = \prod_{k=1}^{n} \frac{1}{\deg(x_k)}$$

$$\prod_{j=1}^{m} F_{\eta^j}(A \backslash V) = F(A)$$

So finally the probability is:

$$\left(\prod_{j=1}^{n} \deg(x_j) \right)^{-1} \prod_{j=1}^{n} G_{V_j}(x_j, x_j)$$

□

5 Wilson's algorithm on directed graphs

The results and the proofs are approximately the same than for undirected graphs (in fact an arborescence can be transformed into a tree by removing the orientation of the edges). Therefore the results will be given without proof and only with remarks about the small part changing.

Let $G = (V, E)$ be a finite, simple (no self-loop or multiple edges), strongly connected and directed graph. Let $\deg(v)$ denote the degree of a vertex $v \in V$. Associate with this graph a random walk which, from a vertex v, randomly goes to one of its $\deg(v)$ neighbors.

5.1 The algorithm

Wilson's algorithm allows to construct a **uniform spanning arborescence** (that is a spanning arborescence chosen from the uniform distribution over the set of spanning arborescence of G). The interesting fact about this algorithm is that it does not require to know the total number of spanning arborescences, and it even computes this number.

Write $V = \{x_0, \dots, x_n\}$ for some $n \in \mathbb{N}$ and set $V_0 = \{x_0\}$ (x_0 will be the root) and $E_0 = \emptyset$. The definition of the algorithm is recursive. Let $k \in \mathbb{N}$ and assume $V_k \subsetneq V$ and E_k constructed. Let $j = \min\{i \in [\![0, n]\!], x_i \notin V_k\}$. Then start a random walk from x_j until it reaches V_k. That gives a path $\omega = [e_0, \dots, e_l]$. Let $\eta = LE(\omega)$ be the (chronological) loop-erasure of ω. Add the edges of η to E_k to get E_{k+1}, and add the vertices adjacent to the edges of η to V_k to get V_{k+1}. If $V_{k+1} = V$ stop, and the spanning arborescence is given by E_{k+1}, otherwise repeat the procedure.

Proposition 6. *The algorithm above produces a spanning arborescence.*

Proof. Assume that $(V_0, \dots, V_k)$ and $(E_0, \dots, E_k)$ are two increasing sequences produced by the algorithm above. Let's prove by induction that for each $l \in [\![0, k]\!]$, $\mathcal{T}_l = (V_l, E_l)$ is an arborescence.

- $(x_0, \emptyset)$ is an arborescence so the property is true for $l = 0$.
- Let $l \in [\![0, k-1]\!]$. Suppose that $\mathcal{T}_l$ is an arborescence. First, choose $v \in_{l+1}$. If $v \in V_l$ then by assumption there exists a unique path from v to x_0. Otherwise if $v \in V \backslash V_l$, v is in $\eta = [\eta_0, \dots, \eta_k]$ the loop-erased path added to $\mathcal{T}_l$. Then $\eta_k \in V_l$ and there exists a unique path going from η_k to x_0. There also exists a unique path going from v to η_k (which is a truncated part of η), so by concatenation of theses two paths there exists a unique path going from v to x_0. Furthermore, $\mathcal{T}_l$ has no loops, and E_{l+1} is obtained by adding a loop-erased path (so without loop), and every of the vertices of this path except the last one are in $V_{k+1} \backslash V_k$ so it can not add any loop in $\mathcal{T}_{l+1}$. So $\mathcal{T}_{l+1}$ has no loops and is an arborescence.

By induction for every $l \in [\![0, k]\!]$ T_l is an arborescence.
To conclude, as the algorithm stops when $V_k = V$, then the result of the algorithm $\mathcal{T}_k$ is a tree and verifies $V_k = V$ so $\mathcal{T}_k$ is a spanning arborescence. □

The next step is to prove that every spanning arborescence of G can be found by the algorithm, and that every spanning arborescence $\mathcal{T}$ has the same probability to be randomly chosen by the algorithm. These two results combined give that the output of the algorithm is sampled from the uniform distribution over the set of the spanning trees of G. The interesting fact is that the computations will also give the probability for each spanning arborescence to be chosen, so by extension (as the distribution is uniform) will give the number of spanning arborescences of G.

5.2 Probabilistic perspective

The interesting facts about this algorithm is that it:

1. Shows that all spanning arborescences of G have the same probability to be chosen by the algorithm
2. Computes the probability (which is the same for every spanning arborescence)
3. Computes the actual number of spanning arborescences: if every spanning arborescence has the same probability p to be chosen, then they follow an uniform distribution and the total number of spanning arborescences is given by $1/p \in \mathbb{N}$. A combinatorial problem (computing the number of spanning arborescences for a given graph) is then solved with a probabilistic approach which was not obvious *a priori*.

Proposition 7. *Let $\mathcal{T}$ be a spanning arborescence of G. The probability that $\mathcal{T}$ is chosen in Wilson's algorithm is*

$$\left(\prod_{j=1}^{n} \deg(x_j) \right)^{-1} \prod_{j=1}^{n} G_{V_j}(x_j, x_j)$$

where $V_j = V \backslash \{x_1, \dots, x_{j-1}\}$ for $j \in [\![1, n]\!]$.

Remark: remember that the quantity $\prod_{j=1}^{n} G_{V_j}(x_j, x_j)$ does not depend of the order of the x_js, so the probability above does not depend of the ordering of the vertices. Furthermore, this quantity will always be the same, no matter the type of random walk used. However the other term will vary, and the spanning arborescences with the biggest weights (product of the weights of its edges) could be drawn more often, depending on the random walk used.

Before proving proposition 7, one may establish a useful property that will be used for the proof.

Property 2. *Let $k \in [\![1, n]\!]$, V_{k-1} be a set of vertices created during the algorithm such that $V_{k-1} \neq V$ (which means that the algorithm has not stopped yet and that there are vertices that have not been explored yet) and $x \in V \backslash V_{k-1} \neq \emptyset$. Then there exists a unique self-avoiding walk η^k in $\mathcal{T}$ that starts at v, ends in V_{k-1} and whose vertices are in $V \backslash V_{k-1}$ ($\neq \emptyset$ by assumption) except for the last one.*

Proof. (of property 1)

- **Existence:** as $\mathcal{T}$ is a spanning arborescence, there exists a path $[\alpha_0, \dots, \alpha_l]$ starting at $\alpha_0 = x$ and ending in $\alpha_l \in V_{k-1}$. One can set $m = \min\{j \in [\![0, l]\!], \alpha_j \in V_{k-1}\}$ and the path $[\alpha_0, \dots, \alpha_m]$ satisfies the condition.
- **Uniqueness:** suppose that there exists two such paths ω_1 and ω_2, going from v to respectively $u_1 \in V_{k-1}$ and $u_2 \in V_{k-1}$, with $\omega_1 \neq \omega_2$. As (V_{k-1}, E_{k-1}) is an arborescence, ω_1 and ω_2 can be extended in two paths $\tilde{\omega}_1$ and $\tilde{\omega_2}$ going from v to x_0. This is not possible as $\mathcal{T}$ is an arborescence (there can not exist two different paths going from v to x_0).

□

Proof. (*of proposition 5*)

Let $\mathcal{T}$ be a spanning arborescence of G and let's compute the probability that it is the output of the algorithm. Set $V_0 = \{x_0\}$ and $E_0 = \emptyset$. This proof is recursive: let $k \in \mathbb{N}^*$ and assume that $V_{k-1} \neq V$ (which means that the algorithm has not stopped yet and that there are vertices that have not been explored yet). Let v be the vertex which is not in V_{k-1} and has the minimal number, that is $v = x_m$ with $m = \min\{l \in [\![0, n]\!], x_l \notin X_{k-1}\}$. Then according to property 1 there exists a unique self-avoiding walk η^k in $\mathcal{T}$ that starts at v, ends in V_{k-1} and whose vertices are in $V \backslash V_{k-1}$ ($\neq \emptyset$ by assumption) except for the last one. It can be seen as "the" walk from x to V_{k-1}. V_k is obtained by adding the vertices of η^k to V_{k-1} and E_k the edges of η^k to E_{k-1}. With this method, $\mathcal{T}$ has been uniquely described as a sequence of self-avoiding walks $[\eta^1, \dots, \eta^k]$, so the probability of choosing $\mathcal{T}$ is

$$\prod_{j=1}^{k} \left(p(\eta^j) F_{\eta^j}(A \backslash V_{j-1}) \right)$$

But

$$\prod_{j=1}^{m} p(\eta^j) = \prod_{k=1}^{n} \frac{1}{\deg(x_k)}$$

$$\prod_{j=1}^{m} F_{\eta^j}(A \backslash V) = F(A)$$

So finally the probability is:

$$\left(\prod_{j=1}^{n} \deg(x_j) \right)^{-1} \prod_{j=1}^{n} G_{V_j}(x_j, x_j)$$

□

6 Application: maze generation

One of the advantages of Wilson's algorithm is that it is very visual. One implementation of the algorithm in Python can be found here. However, as I am not an expert in animation in Python, the animation is not very fast. Faster implementation can be found online, for instance here. One of the first idea that comes to mind when looking at the construction of the spanning tree of a grid $[\![0, n]\!]^2$ is that it looks like a maze. However, it is not exactly a maze: for instance one cannot draw a path going from the bottom left to the top right because it would draw a line between two components of the graph which would then not be connected, and the output of Wilson's algorithm would not be a spanning tree. However it is possible to fix this issue. Instead of working on the grid $[\![0, n]\!]^2$, one can work on its dual graph made of the squares of the grid. On this dual graph, a spanning tree will connect every square of the grid, and for any pair of squares there will exist a unique path between them. An example of such a maze is in figure 1.

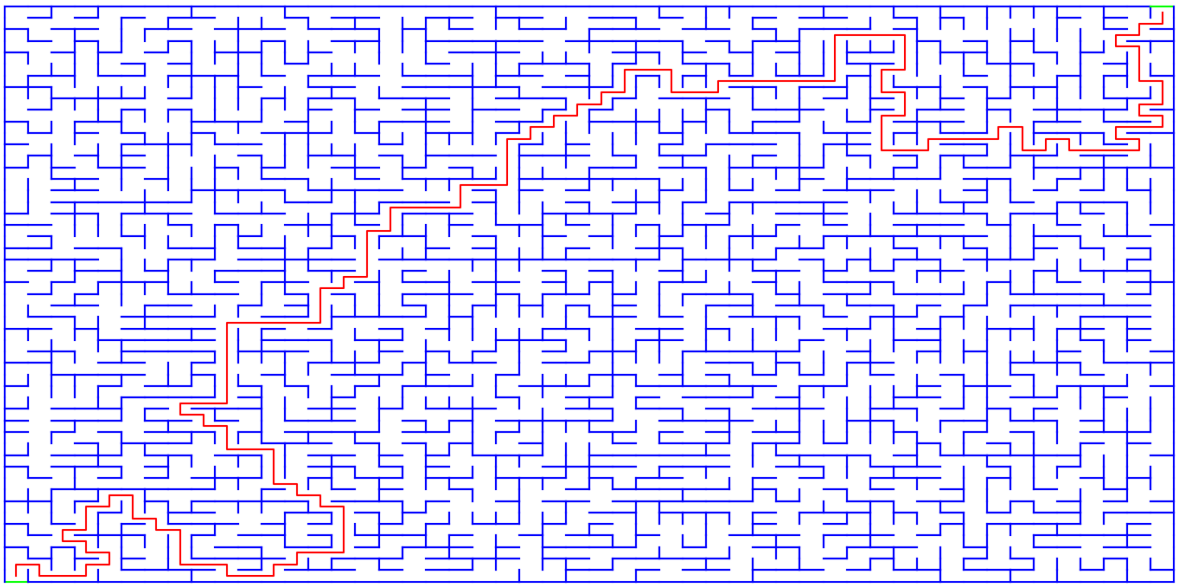

Figure 1: Example of a maze generated, on a grid 50×50, with the unique path drawn

7 Wilson's proof

The idea Wilson used in his article was to, instead of doing random walks, do one single step of randomness at the beginning of the algorithm and then proceed only deterministic steps. His idea was the following: for every vertex $v \in V$ of the graph (excepts for one $v_0 \in V$ set as the root), draw a sequence $(v_n)_{n\in\mathbb{N}}$ of neighbors according to the distribution induced by the Markov chain on this vertex, and transform this sequence into a stack. Then, for each vertex one has created an infinite stack of random neighbors. Once this is done, the set of tops of the stacks form a directed graph. If this graph is a spanning arborescence, then the output is this spanning arborescence. Otherwise it means that there is a cycle in the graph induced. Pop one such cycle by removing the top of each stack of the vertices involved, and repeat this step until the induced graph by the top of the stacks is a spanning arborescence. In his article [5], Wilson proved that this algorithm will terminate almost surely, and also the interesting fact that the order in which one pops the cycles do not matter.

8 Execution time

One key feature of an algorithm is its execution time. Indeed, no matter how smart or beautiful this algorithm is, it is not useful if it can not be executed in reasonable time (and it is even better if it is fast). Wilson's algorithm is stochastic and can last infinitely long (but with probability 0) so one measure of its execution time is its mean execution. For this algorithm, the mean execution time is $O(\tau)$ where τ is the mean hitting time (this notion will be defined).

First, one need some properties on Markov chain. Assume that the chain has a stationary distribution π (for instance if the chain is irreducible and aperiodic). Let i be one state of the Markov chain, and S a stopping time (if you are not familiar with this notion, just assume that for every $n \in \mathbb{N}$, $\{S = n\} \in \sigma(X_0, \dots, X_n)$). An interesting property, due to ergodic theorems, is the following (found in):

Proposition 8. *Assume S is finite almost surely, that $\mathbb{E}_i[S] < +\infty$ and that $X_S = i$. Let j be a state of the chain. Then*

$$\mathbb{E}_i\left[\sum_{k=0}^{S} \delta_{X_k,j}\right] = \pi_j \mathbb{E}_i[S]$$

The proof of this proposition is available in [1]. With this property comes another one, which will be useful for the computation of the mean execution time:

Proposition 9. *Let $j \neq i$ be a state of the chain, and $T_i = \min\{k, X_k = i\}$ be the first time that the chain reaches i (same definition for j). Then:*

$$\mathbb{E}_j\left[\sum_{k=0}^{T_i} \delta_{X_k,j}\right] = \pi_j\left(\mathbb{E}_i[T_j] + E_j[T_i]\right)$$

Proof. Choosing $S := \min\{n \in \mathbb{N}, X_n = i \text{ and } \exists k \leq n, X_k = j\}$ (which can be translated as "first return to i after visiting j") in proposition 8 gives:

$$\mathbb{E}_i\left[\sum_{k=0}^{S} \delta_{X_k,j}\right] = \pi_j \mathbb{E}_i\left[S\right]$$

Furthermore, imagine that the chain starts from i, and define $T := \min\{k, X_k = j\}$. Then the following equality states:

$$\begin{aligned}
\mathbb{E}_i\left[\sum_{k=0}^{S} \delta_{X_k,j}\right] &= \mathbb{E}_i\left[\sum_{k=0}^{T-1} \delta_{X_k,j}\right] + \mathbb{E}_j\left[\sum_{k=T}^{S} \delta_{X_k,j}\right] \\
&= \mathbb{E}_j\left[\sum_{k=T}^{S} \delta_{X_k,j}\right] \text{ by property of } T \\
&= \mathbb{E}_j\left[\sum_{k=0}^{S} \delta_{X_k,j}\right] \text{ because when starting from } j, T = 0
\end{aligned}$$

Moreover, $\mathbb{E}_i[S] = \mathbb{E}_i[T_j] + \mathbb{E}_j[T_i]$ because $\mathbb{E}_i[T_j]$ is the mean time before reaching j for the first time starting from i, and $\mathbb{E}_j[T_i]$ the opposite. Finally, combining all these results:

$$\begin{aligned}
\mathbb{E}_i\left[\sum_{k=0}^{S} \delta_{X_k,j}\right] &= \mathbb{E}_j\left[\sum_{k=0}^{S} \delta_{X_k,j}\right] \\
&= \pi_j\left(\mathbb{E}_i[T_j] + \mathbb{E}_j[T_i]\right)
\end{aligned}$$

□

Assume that the graph is directed, and fix r the root for the spanning arborescence. Using proposition 9, one can derive that the expected number of random steps in the algorithm is lower than $\sum_{v \in V \setminus \{r\}} \pi_v\left(\mathbb{E}_u[T_r] + \mathbb{E}_r[T_u]\right)$. Indeed, it corresponds to the number of times each vertex will be hit before the algorithm ends.

Definition 7. *The **mean hitting time** is defined as:*

$$\tau = \sum_{i,j} \pi_i \pi_j \mathbb{E}_i[T_j]$$

With this definition, if ones defines the root π-randomly, then the mean execution time is bounded by $\sum_{i,j} \pi_i \pi_j \left(\mathbb{E}_u[T_r] + \mathbb{E}_r[T_u]\right) = 2\tau$ (summing a weighted combination of the previous formula with weights set to π). So the execution time of Wilson's algorithm is $O(\tau)$.

Compared to a lot of other algorithms, Wilson's algorithm will be faster in average. Indeed, Aldous' algorithm is $O(T_c)$ where T_c is the cover time. The cover time is the maximum expected number of steps before visiting every state, maximum taken over every possible initial state. In mathematical terms, $T_c = \max_i \mathbb{E}_i[A]$ where $A = \min\{n \in \mathbb{N}, S \subset \{X_0, \ldots, X_n\}\}$ with S the set of the states of the chain. In his article, Wilson cites other algorithms which are $O(|V|^3)$ or $O(|E|^{2,376})$ (this exponent is in fact the lowest known for the computation of a product of matrices with the Coppersmith-Winograd algorithm, see for instance [3]) which also are slower than his algorithm.

References

[1] D. Aldous and James Fill. Reversible Markov Chains and Random Walks on Graphs. January 2002.

[2] David J. Aldous. The Random Walk Construction of Uniform Spanning Trees and Uniform Labelled Trees. *SIAM J. Discrete Math.*, 3(4):450–465, November 1990.

[3] Matthew Anderson and Siddharth Barman. The Coppersmith-Winograd Matrix Multiplication Algorithm. page 11.

[4] Greg Lawler. *Random Paths and Fields*. 2010.

[5] David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, STOC '96, pages 296–303, New York, NY, USA, July 1996. Association for Computing Machinery.