A STUDY OF RANDOM WALK BASED ALGORITHMS FOR EFFICIENT GENERATION OF UNIFORM SPANNING TREES

Igor de Oliveira Nunes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Daniel Ratton Figueiredo
Giulio Iacobelli

Rio de Janeiro
Julho de 2020

A STUDY OF RANDOM WALK BASED ALGORITHMS FOR EFFICIENT GENERATION OF UNIFORM SPANNING TREES

Igor de Oliveira Nunes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Daniel Ratton Figueiredo
Giulio Iacobelli

Aprovada por: Prof. Daniel Ratton Figueiredo
Prof. Giulio Iacobelli
Prof. Valmir Carneiro Barbosa
Prof. Roberto Imbuzeiro Oliveira

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2020

Nunes, Igor de Oliveira

A study of random walk based algorithms for efficient generation of uniform spanning trees/Igor de Oliveira Nunes. – Rio de Janeiro: UFRJ/COPPE, 2020.

IX, 53 p.: il.; 29, 7cm.

Orientadores: Daniel Ratton Figueiredo
Giulio Iacobelli

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2020.

Referências Bibliográficas: p. 48 – 53.

1. uniform spanning trees. 2. random walks on graphs. 3. hybrid algorithm. I. Figueiredo, Daniel Ratton *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UM ESTUDO DE ALGORITMOS BASEADOS EM PASSEIOS ALEATÓRIOS PARA A GERAÇÃO EFICIENTE DE ÁRVORES GERADORAS UNIFORMES

Igor de Oliveira Nunes

Julho/2020

Orientadores: Daniel Ratton Figueiredo
Giulio Iacobelli

Programa: Engenharia de Sistemas e Computação

Árvore geradora uniforme é a medida de probabilidade uniforme no conjunto de árvores de geradoras de um grafo. Exploradas desde o consagrado trabalho de Kirchhoff na década de 1840, essas árvores aleatórias estão entre os objetos aleatórios mais antigos e mais amplamente estudados na teoria dos grafos e estão relacionados a diversos temas importantes em probabilidade discreta, além de encontrarem aplicação em muitas áreas. Não é de surpreender que a tarefa de gerar eficientemente árvores geradoras uniformes tenha recebido muita atenção. No entanto, nas últimas décadas, as estratégias mais promissoras para o problema mudaram significativamente. Esse avanço ocorreu com os algoritmos de *Aldous-Broder* e *Wilson*, que constroem árvores utilizando passeios aleatórios. Neste trabalho, estudamos o comportamento transiente destes dois algoritmos. Introduzimos a noção de *branches*, que são caminhos adicionadas às árvores em determinados tempos de parada. Essa interpretação é usada para mostrar uma equivalência transiente entre os dois algoritmos em grafos completos. A equivalência leva a uma abordagem híbrida que gera árvores geradoras uniformes de grafos completos mais rapidamente do que cada um dos dois algoritmos. Propomos uma metodologia para explorar essa abordagem híbrida em um cenário mais geral, mostrando a viabilidade em alguns exemplos. Por fim, mostramos como a metodologia pode ser adaptada para gerar árvores aleatórias com probabilidade proporcional ao número de uma dada subárvore fornecida como parâmetro.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A STUDY OF RANDOM WALK BASED ALGORITHMS FOR EFFICIENT GENERATION OF UNIFORM SPANNING TREES

Igor de Oliveira Nunes

July/2020

Advisors: Daniel Ratton Figueiredo
Giulio Iacobelli

Department: Systems Engineering and Computer Science

The uniform spanning tree is the uniform probability measure on the set of spanning trees of a graph. Explored since Kirchhoff's notorious work the 1840's, spanning trees are among of the oldest and most extensively studied random objects in graph theory. They find application in many fields and are related to many interesting problems in discrete mathematics. Not surprisingly, the task of efficiently generating uniform spanning trees has received much attention since Kirchhoff's work. A breakthrough came with *Aldous-Broder* and *Wilson's* algorithms, which can efficiently generate spanning trees of any graph based on random walks. In this work, we study the transient behavior of both algorithms. We introduce the notion of *branches*, which are paths generated by the two algorithms on particular stopping times. This interpretation is used to show a transient equivalence between the two algorithms on complete graphs. This equivalence yields a hybrid approach to generate uniform spanning trees of complete graphs faster than either of the two algorithms. We also propose a framework to explore this hybrid approach on a more general setting, showing its feasibility in various examples. Lastly, we show how the framework can be adapted to generate random trees with probability proportional to the number of a given sub-tree provided as input.

Contents

List of Figures

Chapter 1

Introduction

Graphs are one of the most fundamental tools of mathematical abstraction and can be used to model pairwise relationships between objects. A graph G consists of an ordered pair (V, E) where V is a set of vertices, usually identified with labels, and E a set of edges. In this work, we will mostly consider graphs that are labelled, simple, connected and undirected, i.e., the elements $e \in E$ are *unordered* pairs of vertices and there is at least one path (sequence of edges) between every pair of vertices.

In graph theory, one important concept is that of *spanning trees*. A spanning tree $T_G = (V_T, E_T)$ of an undirected graph $G = (V, E)$ is a connected acyclic subgraph of G, such that $V_T = V$. Figure 1.1 illustrates a simple graph G and all its spanning trees.

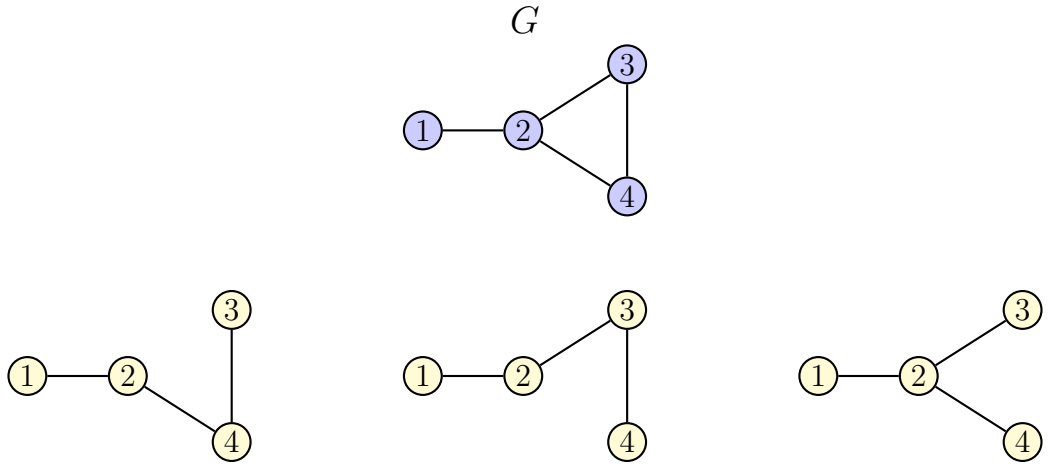


Figure 1.1: An example of a simple, connected and undirected graph and all its spanning trees.

Numerous applications of practical significance are based on spanning trees. Many classical algorithms for finding paths between vertices build spanning trees as an intermediate step [1, 2]. Several protocols used in the Internet and other telecommunication networks are also based on spanning trees [3–5]. Optimization

over spanning trees also gives way to an extensive range of applications, from approaches to the Travelling Salesman problem [6] to image segmentation problems in computer vision [7].

In general, a graph has an exponential number of distinct spanning trees (see Section 2.1.1). In the field of probability theory, an ordinary question arises: *how does a "typical" spanning tree look like?* The investigation of this question brought forth one of the most extensively explored probabilistic objects in graph theory: *uniform spanning trees* (USTs) [8]. The *uniform spanning tree* of a simple connected graph $G = (V, E)$, denoted by $\mathrm{UST}(G)$, is the uniform probability measure on the set of all spanning trees of G, denoted by $\mathcal{T}_G$. The study of this object dates back to *Kirchhoff's theorem* [9], which provides the size of the set $\mathcal{T}_G$ in terms of the eigenvalues of the Laplacian matrix of G (detailed in Section 2.1).

From that time on, it has been shown that the model has surprising connections to a rich set of subjects in discrete probability, statistical mechanics and theoretical computer science. Amongst other topics, some examples are: random walks, percolation, gaussian fields, dimers and sandpile models [10–16]. Uniform spanning trees are also used in generating expander graphs, which have a large number of applications [17]. Furthermore, USTs have played a crucial role in breaking long-standing limits in approximation algorithms for the Travelling Salesman Problem, in both symmetric and asymmetric versions [18, 19].

Uniform spanning trees are also related to an amusing application: *Maze* construction. Mazes, which are also called labyrinths, have many uses, from video games and tourist attractions to psychology experiments and robotics [20–22]. In his book *Mazes for Programmers* [23], Jamis Buck presents numerous algorithms that can be used to generate mazes. Among those, algorithms for generating uniform spanning trees are used to create mazes with uniform probability in the 2D grid as illustrated in Figure 1.2.

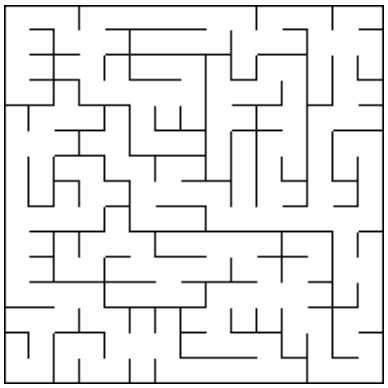


Figure 1.2: Illustration of a maze on the 15×15 2D grid randomly selected with uniform probability (Buck 2015).

Taking into consideration this wide assortment of applications, it is not surpris-

ing that the quest for designing fast algorithms for generating uniform spanning trees has received significant attention over time. As a consequence of Kirchhoff's work, the first endeavors to algorithmically generate uniform spanning trees were the determinant-based approaches. The first algorithms produced a uniform spanning tree in running time $O(mn^3)$ where $n = |V|$ and $m = |E|$ [24, 25]. In a sequence of works, Colbourn *et al.* made improvements to the determinant-based strategy, culminating in an algorithm with the same complexity of a $n \times n$ square matrix multiplication [26–28].

It was only over a century after Kirchhoff's work, around 1990, that David Aldous and Andrei Broder independently discovered a formidable connection between uniform spanning trees and random walks [29, 30]. They showed that the set of edges that a random walk uses to reach each vertex for the first time has the uniform law over all the spanning trees of the graph. This result is discussed in detail in Section 3.1. Although their findings were published independently, both authors acknowledge Persi Diaconis for previous discussions on the topic.

The now called *Aldous-Broder algorithm*, immediate consequence of the above-mentioned theorem, emerged as a much more promising approach to the problem of generating uniform spanning trees in terms of running time complexity. The algorithm is capable of generating a uniform spanning tree of any graph in time proportional to the cover time of the graph, which is significantly smaller than previous running time for most graphs [31]. Regardless of this, the task of improving the field did not settle down. Quite the opposite, the idea gave way to a new paradigm, the so called random walk-based approaches, with many new papers providing successively more efficient approaches [32–35].

The first result that revealed the possibility of using random walks to generate USTs faster than the cover time is due to David Wilson [32]. *Wilson's algorithm* is based on a distinct random process: *loop-erased random walks* (see Section 3.2). The approach generates USTs with expected time proportional to the graph's hitting time, which is never larger than a graph's cover time. Together, Aldous-Broder and Wilson's algorithms are considered two major pillars of current research on the topic, since most of the latest developed theory and applications are based on them.

1.1 Contributions

While both Aldous-Broder and Wilson's algorithms are based on random walks their behavior is fundamentally different. The main theme addressed in this thesis is the analysis of the transient behavior of both algorithms with the purpose of generating uniform spanning trees more efficiently. The main result of this work is a positive result on combining the two algorithms. In fact, considering a complete

graph and an adequate sequence of stopping times, we show that the two algorithms are equivalent in their transient behavior. More formally:

Theorem 4.3.1 (*Aldous-Broder* and *Wilson* transient equivalence)**.** *Let Aldous-Broder and Wilson algorithms have the same initialization for the initial vertex r and let the input graph G be a complete graph. Then, there exist two sequences of stopping times $(\hat{\sigma}_i)_{i\geq 0}$ (for Wilson) and $(\sigma_i^{\text{in}})_{i\geq 0}$ (for Aldous-Broder) such that, for every $i \geq 0$*

$$T^W_{\hat{\sigma}_i} \stackrel{d}{=} T^{AB}_{\sigma_i^{\text{in}}} .$$

where $T^W_{\hat{\sigma}_i}$ and $T^{AB}_{\sigma_i^{\text{in}}}$ are the trees constructed by *Wilson* and *Aldous-Broder* by time $\hat{\sigma}_i$ and σ_i^{in}, respectively. This result leads to a hybrid algorithm that starts with Aldoud-Broder and then switches to Wilson to finish the spanning tree of a complete graph. We show analytically and through simulations that this approach is more efficient than running either of the two algorithms separately (by a constant factor when considering Wilson).

The next contribution is a two-phase framework to explore such hybrid approach in a more general setting. The first phase generates a sub-tree of G according to a distribution which satisfies an "convenient" condition (properly defined in Chapter 5). The second phase finishes by generating a uniform random spanning tree given the initial sub-tree. We show that Wilson's algorithm does the job for second phase, captured by the following theorem:

Lemma 5.0.2 (Second stage)**.** *Let $G = (V, E)$ be a connected graph and ζ be a sub-tree (not spanning) of G. Wilson's algorithm with initial condition ζ returns a spanning tree of G which is uniformly distributed on the set $\{\tau \in \mathcal{T}_G : \tau \supseteq \zeta\}$.*

Wilson with an initial condition is a natural concept defined properly in Section 3.2. While generating a sub-tree ζ that satisfies the condition for the first phase is not trivial, we provide several examples where this is feasible and efficient. For example, when considering any vertex-transitive graph G, a random edge of G satisfies the condition for the first phase.

The running time complexity of the two-phase framework strongly depends on the size of the initial sub-tree. However, the running time complexity of the second phase will be in the same order as Wilson's algorithm on G. Thus, an open challenge for the proposed framework is generating random sub-trees with non-negligible size for the first phase.

Of side interest, the framework can be adapted to generate random spanning trees of the complete graph with probability proportional to the number of any sub-tree provided as input (see Section 5.1). For example, consider as sub-tree ζ a star graph with k nodes, then we can generate a spanning tree T of the complete

graph with probability that is proportional to the number of sub-trees of T that are isomorphic to ζ. This finds application in the area of generating random graphs (trees) that have a law for pre-defined subgraphs (sub-trees) [36–38].

1.2 Outline

The remainder of this work is organized as follows. Chapter 2 presents a summary of the classical results, considered the starting point for the study of uniform spanning trees. Topics are not directly related to random walks, since the connection was only discovered several years later. Chapter 3 introduces the random walk-based approaches and some related work. These approaches represent the main technical background of this work. Chapter 4, introduces a dynamic interpretation for both Aldous-Broder and Wilson's algorithms and then shows how this description reveals a transient equivalence between them for complete graphs. To conclude the chapter, in Section 4.3.1 we present a hybrid algorithm, which takes advantage of this equivalence to generate USTs of complete graphs more efficiently. In Chapter 5 we investigate the possibility of using this idea to generate trees in a more general setting. We describe a general framework and provide some examples of its usage. Section 5.1 shows that the framework can also be used to generate spanning trees of the complete graph with probability proportional to the number of a sub-tree given as input. Finally, Chapter 6 concludes the thesis with a summary of the main results and possible future work.

Chapter 2

Background

In this chapter we present some classical results concerning uniform spanning trees (USTs). We start presenting Kirchhoff's theorem in Section 2.1. The result is commonly regarded as the starting point of the area. We also introduce Cayley's formula as a special consequence of Kirchhoff's theorem. In Section 2.2 we describe Prüfer sequences, a method for enumerating labeled trees which also yields an interesting algorithm to generate them uniformly. Finally, in Section 2.3 we introduce the Galton-Watson branching process, show how it is related to USTs and discuss how this relationship can be used to efficiently generate uniform random trees conditioned on the number of vertices, observed in the critical state.

2.1 Kirchhoff's Theorem

The study of uniform spanning trees dates back to *Kirchhoff's matrix tree theorem* [9] (or simply *Kirchhoff's theorem*), which describes how to determine the size of the set $\mathcal{T}_G$ of all spanning trees of a graph G. The number is obtained in terms of the eigenvalues of the Laplacian matrix $L(G)$ of the graph, defined as:

$$L(G) = D(G) - A(G) \tag{2.1}$$

where $D(G)$ and $A(G)$ are the degree and adjacency matrices of G respectively.

Theorem 2.1.1 (Kirchhoff's matrix tree theorem)**.** *Let G be a connected graph with n labeled vertices and let $\lambda_1, \ldots, \lambda_{n-1}$ be the non-zero eigenvalues of its Laplacian matrix $L(G)$. The number $|\mathcal{T}_G|$ of spanning trees of G is equal to $\frac{1}{n}\lambda_1\lambda_2 \ldots \lambda_{n-1}$. Equivalently, $|\mathcal{T}_G|$ is equal to any cofactor of $L(G)$.*

Consider, for example, the graph below:

G

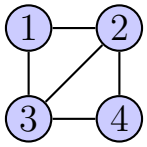


Figure 2.1: An example of a connected graph G, with $n = 4$ labeled vertices.

To obtain the number of spanning trees of G, first we construct its Laplacian matrix:

$$D(G) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \qquad A(G) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$L(G) = D(G) - A(G) = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

Then, we need to obtain a cofactor of L. By deleting the first row and column, for example, we have:

$$L^{1,1}(G) = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

The $(1,1)$-*cofactor* of $L(G)$ is the determinant of $L^{1,1}(G)$, which is equal to 8. Thus, from Theorem 2.1.1 we know that the number of spanning trees of G is 8. Figure 2.2 lists all eight spanning trees of G to illustrate this result:

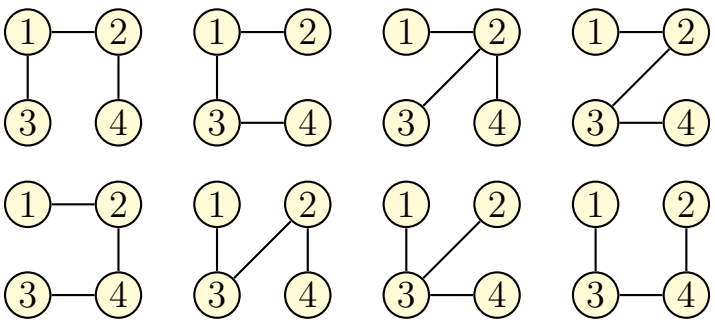


Figure 2.2: Representation of all eight spanning trees of G.

2.1.1 Cayley's Formula

Named after Arthur Cayley, the result known as *Cayley's Formula* [39] states that the number of distinct labeled trees on n vertices is exactly n^{n-2}, for every positive integer n. Although many proofs of the result are known [40], the formula can be derived from a special case of *Kirchhoff's theorem*, since each labeled tree with n vertices is a spanning tree of K_n, the complete graph on n vertices. The Laplacian matrix of a complete graph is:

$$L(K_n) = \underbrace{\begin{bmatrix} n-1 & -1 & \dots & -1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{bmatrix}}_{n}$$

From *Kirchhoff's theorem*, we know that the number of spanning trees of K_n is equal to any cofactor of $L(K_n)$. If we delete the first row and column, we have:

$$L^{1,1}(K_n) = \underbrace{\begin{bmatrix} n-1 & -1 & \dots & -1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{bmatrix}}_{n-1}$$

Note that $L^{1,1}(K_n)$ has the same pattern of $L(K_n)$ ($n-1$ in the diagonal and -1 everywhere else), but has $n-1$ rows and columns. To obtain the determinant of $L^{1,1}(K_n)$ we can construct matrix A by subtracting the bottom row from each of the other rows:

$$A = \underbrace{\left[\begin{array}{cccc|c} n & & & & -n \\ & n & & 0 & -n \\ & & \ddots & & \vdots \\ & 0 & & n & -n \\ \hline -1 & -1 & \dots & -1 & n-1 \end{array}\right]}_{n-1}$$

If we add each of the first $n-2$ columns to the last column we get:

$$A' = \underbrace{\left[\begin{array}{cccc|c} n & & & & 0 \\ & n & & 0 & 0 \\ & & \ddots & & \vdots \\ & 0 & & n & 0 \\ \hline -1 & -1 & \dots & -1 & 1 \end{array}\right]}_{n-1}$$

Since the operation of adding rows and columns to other rows and columns preserves the determinant, we obtain:

$$det(L^{1,1}(K_n)) = det(A) = det(A') = n^{n-2}$$

where the last equality follows as A' is lower triangular, then its determinant is the product of its diagonal values. Remember that $det(L^{1,1}(K_n))$ is the $(1,1)$-*cofactor* of $L(K_n)$, and from Kirchhoff's theorem it is equal to the number $|\mathcal{T}_{K_n}|$ of spanning trees of K_n, demonstrating *Cayley's Formula*.

2.2 Prüfer Sequences

With the purpose of proving *Cayley's formula*, Heinz Prüfer [41] created a method for enumerating every labeled tree on n vertices. Prüfer proved a one-to-one correspondence between sequences of $n-2$ numbers and trees on n labeled vertices. The so called *Prüfer sequence* (or *Prüfer code*) of a labeled tree with n vertices is a unique sequence $\mathcal{S}$ of length $n-2$ on the labels $\{1,\dots,n\}$, with repetitions allowed. Moreover, for any given sequence $\mathcal{S}$ on $\{1,\dots,n\}$ of size $n-2$, there exists only one labeled tree on n vertices whose *Prüfer sequence* is $\mathcal{S}$. Figure 2.3 shows all 16 labeled trees with four vertices and their corresponding *Prüfer sequences*. Equivalently, these are all the spanning trees of K_4, the complete graph with four vertices.

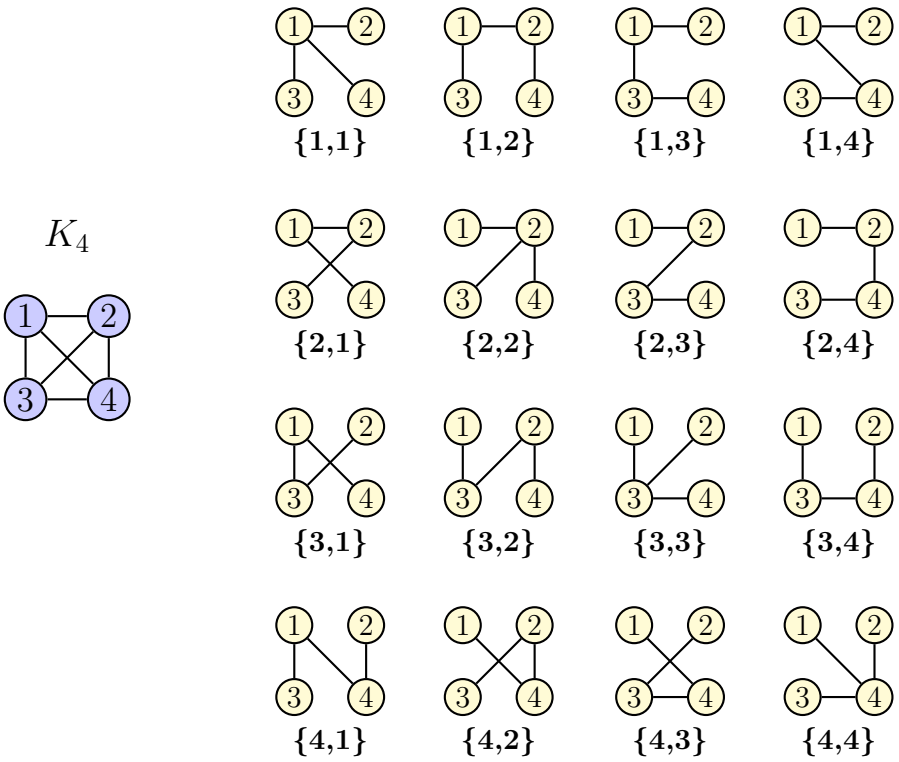


Figure 2.3: Representation of all the spanning trees of the complete graph with four vertices and their corresponding *Prüfer sequences*.

An immediate consequence of this bijection, is that the number of trees on n labeled vertices is exactly the same as the number of possible sequences of length $n-2$ on n numbers. While it would seem hard to count by looking at the trees, it is rather simple to determine that there are exactly n^{n-2} possible sequences, once again, proving *Cayley's formula*.

2.2.1 Algorithms

Another important aspect of Prüfer's result is that it yields a simple algorithm to obtain the sequences from the trees and vice-versa. The following procedure receives a tree as input and returns the corresponding *Prüfer sequence*. We denote by $\ell(T)$ the set of vertices in T that have degree 1, namely the leaves of T (the degree $d(v)$ of a vertex v is simply the number of edges that are incident to it).

tree-to-sequence($T = (V, E)$)

- Set $\mathcal{S} \leftarrow \{\}$
- While $|V| > 2$:
  - Remove from T the vertex $v \in \ell(T)$ with the smallest label and append its neighbor's label at the last position of $\mathcal{S}$;
- Return $\mathcal{S}$

In Figure 2.4, we give an example of input tree T, and illustrate the steps to obtain its corresponding sequence following the above algorithm. The represented steps are:

1) Vertex 1 has the smallest label among the leaves, so we remove it and add its parent's label 3 to $\mathcal{S}$;
2) in the remaining tree, vertex 3 has the smallest label, then we remove it and add 2 to $\mathcal{S}$;
3) finally, we add 2 again to the sequence, since it is the label of the parent of vertex 4, the current smallest leaf.

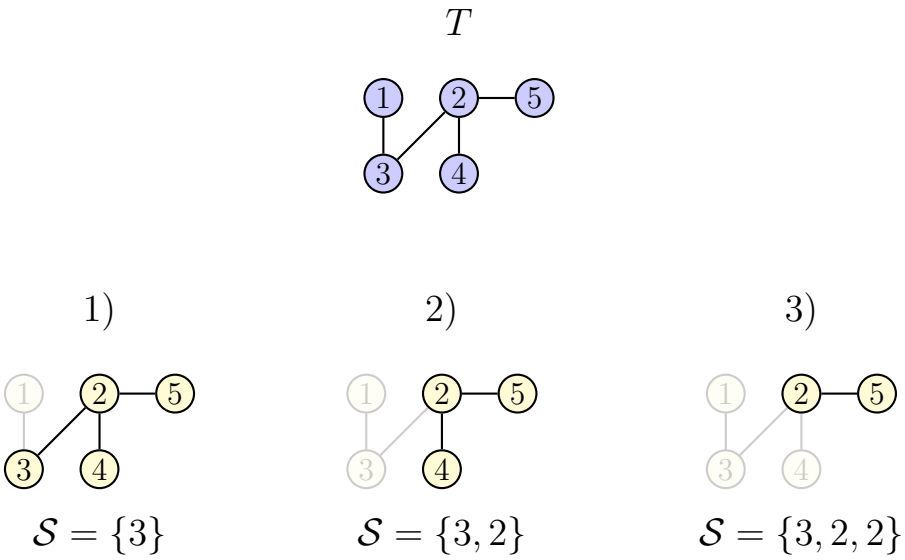


Figure 2.4: An illustation of the steps to obtain a *Prüfer sequence* from a tree.

The inverse process, i.e., constructing a tree from a sequence is also simple. First, note that the degree of a vertex in the tree is always the number of appearances of its label in the sequence plus one. Again, let us denote by $d(v)$ the degree of vertex v. Considering this, the algorithm to construct the tree given a sequence is

described below. Note that we treat the degrees as variables in the algorithm in order to simplify its description.

sequence-to-tree($\mathcal{S}$)

- Set $V \leftarrow |\mathcal{S}| + 2$ vertices, numbered $\{1, \ldots, |\mathcal{S}| + 2\}$, $E \leftarrow \varnothing$
- For each $s \in \mathcal{S}$:
  - For each $v \in V$:
    - if $d(v) = 1$: Set $E \leftarrow E \cup \{v, s\}$, $d(v) \leftarrow d(v) - 1$ and $d(s) \leftarrow d(s) - 1$
- Set $E \leftarrow E \cup \{i, j\}$, where i and j are the only two vertices that remain with degree 1.
- Return $T = (V, E)$

Both algorithms, *tree-to-sequence* and *sequence-to-tree*, can be implemented very efficiently in terms of running time complexity, as they are asymptotically bounded by the label sorting procedure. Using efficient integer sorting strategies, such implementations run in $O(n)$ time [42, 43].

Another important outcome of *Prüfer sequences*, particularly related to the subject of this work, is that the idea can be easily used to efficiently generate uniform spanning trees of complete graphs. It is obvious that, in order to generate one of the n^{n-2} sequences uniformly, one can simply sample $n - 2$ labels from $\{1, \ldots, n\}$ uniformly with replacement and insert them in sequence. If we run *sequence-to-tree* on that sequence, from Prüfer's bijection the resulting tree will also be uniformly chosen among the trees on n vertices.

2.3 Galton-Watson branching process

The *Galton-Watson branching process* is a simple approach initially designed to model the evolution of a population [44] and it is considered one of the fundamental processes in probability theory. Although earlier studied by Bienaymé [45], the model is usually attributed to Galton and Watson [46] due to their anthropological study *"On the probability of the extinction of families"*. Despite its simplicity, the branching process has proved to be remarkably useful, finding applications in a myraid of unrelated areas. The model can be described by the following recursive steps:

1. Starting from a root vertex r, create k children vertices, where k is distributed over non-negative integers according to some distribution ξ;

2. Recursively repeat for each child.

The only parameter of the model is the distribution ξ of the number of children of each vertex, called the *offspring distribution*. The constructed tree is commonly referred to as a *Galton-Watson tree*. Consider the example on Figure 2.5. Let us denote by c_i the number of vertices on the i-th generation, which corresponds to level i in the tree rooted at r. We start from the root vertex on the left in generation 0. Generations 1 and 2 have three vertices each, so $c_1 = c_2 = 3$. The third generation has 2 vertices ($c_3 = 2$) and all the subsequent generations have no vertices. Clearly, to obtain a finite tree like in the example, the offspring distribution must have non-zero value for $\mathbb{P}(\xi = 0)$. In fact, one of the main results of branching processes is that when $\mathbb{E}(\xi) \leq 1$, the tree is always finite and when $\mathbb{E}(\xi) > 1$, there is a non-zero probability that the tree grows indefinitely [47].

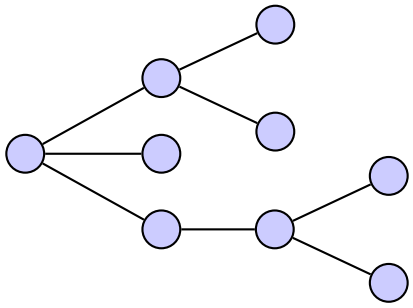


Figure 2.5: Example of tree constructed by a *Galton-Watson* branching process.

2.3.1 The uniform spanning tree as a Galton-Watson tree

Amongst the extensive range of applications, the model generates particularly intriguing trees when the offspring distribution has expected value 1 ($\mathbb{E}(\xi) = 1$) [48]. Such trees are denominated *critical* Galton-Watson trees. The following proposition shows that a specific critical Galton-Watson tree also has a very interesting connection to uniform spanning trees.

Proposition 2.3.1. *Consider the following process:*

- *Construct a Galton-Watson tree with Poisson(1) offspring distribution. Let N (random variable) be the total number of vertices on the obtained tree;*

- *Assign the vertices unique random labels chosen from $\{1, \ldots, N\}$.*

The obtained labeled tree is a uniform spanning tree of the complete graph on N vertices.

Proof. Consider the Galton-Watson tree T and let X_i be the random variable denoting the number of children of the i-th vertex. We start describing the probability of the branching process generating a particular structure (unlabeled tree) τ, conditioned on having exactly n vertices, identified by $X_1 = \xi_1, \ldots, X_n = \xi_n$. First, note that this probability is entirely determined by the numbers of offspring of each vertex:

$$\mathbb{P}\left(T = \tau \;\middle|\; |T| = n\right) = \mathbb{P}\left(X_1 = \xi_1, \ldots, X_n = \xi_n \;\middle|\; \sum_{i=1}^{n} X_i = n - 1\right) \tag{2.2}$$

Notice that we condition on $\sum_{i=1}^{n} X_i = n - 1$ because the first vertex is not counted as offspring of another vertex. Since the variables are independent and *Poisson(1)* distributed, we have:

$$\mathbb{P}\left(T = \tau \;\middle|\; |T| = n\right) = \frac{\prod_{i=1}^{n} \mathbb{P}\left(X_i = \xi_i\right)}{\mathbb{P}\left(\sum_{i=1}^{n} X_i = n - 1\right)} = \frac{\prod_{i=1}^{n} \frac{e^{-1}}{\xi_i!}}{\frac{e^{-n} n^{n-1}}{(n-1)!}} = \frac{(n-1)!}{n^{n-1} \prod_{i=1}^{n} \xi_i!} \tag{2.3}$$

Note that this probability still depends on the structure of the tree, given that it is a function of the ξ_i factors. However, we still need to assign the labels to each vertex. Even though we have $n!$ possible permutations of the labels, not all of them represent unique trees since we consider unordered lebeled trees. *Unordered* means that the following trees are all considered identical, since ξ_i is preserved across the trees for all i:

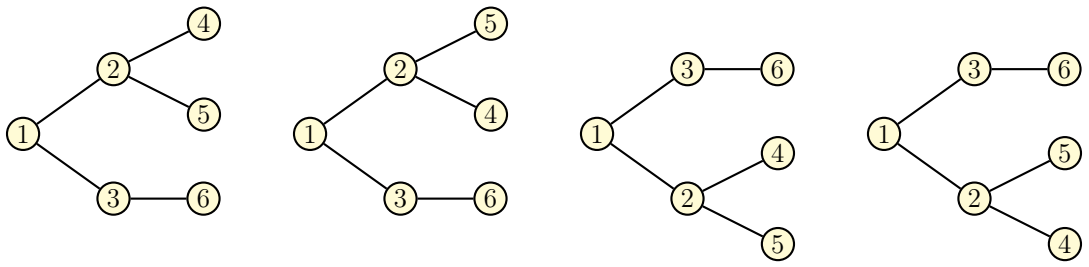


Figure 2.6: Example of permutations of labels that correspond to identical trees, since the number of offspring of every node is preserved across the identical trees.

while the following two are different:

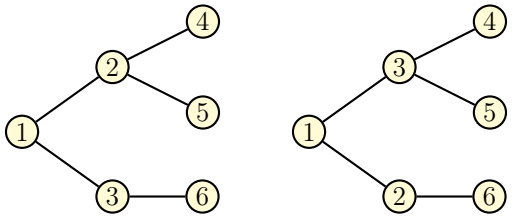


Figure 2.7: Example of permutation of labels that correspond to distinct trees, since ξ_3 is one and two in each of the trees.

Note that changing the order of the vertices of the same generation and making the corresponding adjustments in the following generations results in identical trees. The root vertex can receive one of the n labels. There are $(n-1)!$ different ways to assign labels to the other vertices, of which $\prod_{i=1}^{n} \xi_i!$ correspond to the same unordered labeled tree. Then, the number $\mathcal{L}(\tau)$ of labelings that result in distinct trees is:

$$\mathcal{L}(\tau) = \frac{(n-1)!}{n \prod_{i=1}^{n} \xi_i!} \tag{2.4}$$

Finally, the probability of obtaining a particular labeled tree is $\mathcal{L}(\tau)^{-1}\mathbb{P}\left(T = \tau \;\middle|\; |T| = n\right) = 1/n^{n-2}$. □

2.3.2 Simulating size-constrained Galton-Watson trees

Even though the aforementioned result is very interesting from a theoretical perspective, it does not provide a clear method for sampling uniform spanning trees with n vertices efficiently, since the size of the obtained tree N is a random variable. As a matter of fact, one can use *rejection sampling* [43, 48] discarding the generated trees until obtaining the desired size, but then the running time complexity would intuitively not be linear.

Fortunately, a solution to the problem was proposed by Devroye [48] by providing *the multionomial method*, which splits the task into generating a multinomial random vector, applying a certain permutation and then constructing the tree based on the result. The method is not only restricted to sampling uniform spanning trees. It is capable of generating any size-conditioned Galton-Watson tree and, particularly when $\mathbb{E}(\xi^2) < \infty$, the method is $O(n)$. Since in the process described in Proposition 2.3.1 we have $\xi \sim Poisson(1)$, thus $\mathbb{E}(\xi^2) = 2$, the multinomial method is able to create uniform spanning trees in linear time.

To begin, notice that the branching process generates a particular structure given by its offspring sequence $\{\xi_1, \ldots, \xi_n\}$. In order to obtain the specific tree from the sequence, a traversing order must be established. For example, we can traverse the tree in *BFS* (*breadth first search*) order. In the example of Figure 2.8 we illustrate

the mapping of the offspring sequence $\{3, 2, 0, 1, 0, 0, 2, 0, 0\}$ to a tree. The vertex label indicates the order in which they are inserted in the tree. Although such order is arbitrary between vertices of the same generation, it must be kept so that the sequence maps to a unique tree. Given this traversal ordering, the steps are:

1) The first number of the sequence is 3, indicating that $\xi_1 = 3$. Then, the next three vertices (2, 3 and 4) are added to the tree as children of the first vertex;

2) The next three numbers in the sequence ($\xi_2 = 2$, $\xi_3 = 0$ and $\xi_4 = 1$) indicate *respectively* the offspring of vertices 2, 3 and 4;

3) The following numbers (0, 0 and 2) are, in respective order, the offspring of the least recently added vertices: 5, 6 and 7; Since all the following numbers are zeros, no more vertices need to be inserted and the process finishes.

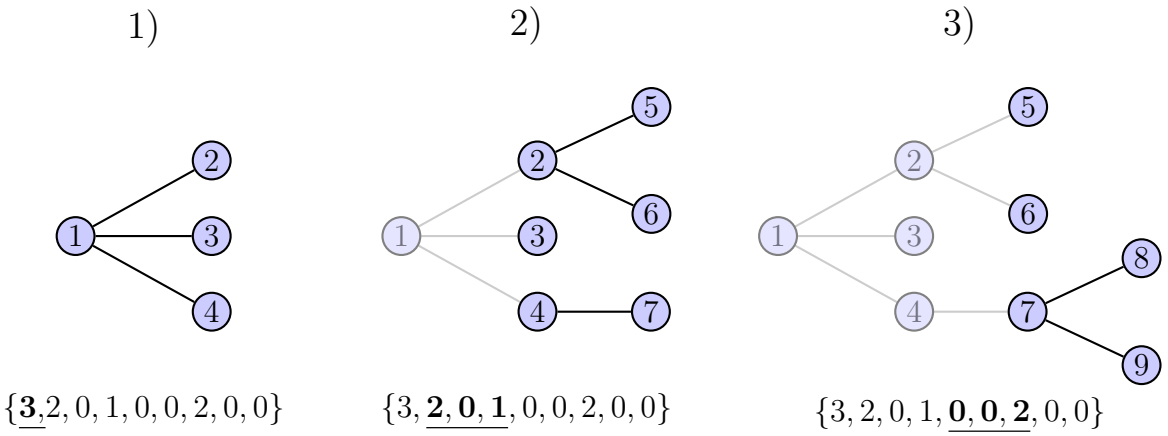


Figure 2.8: Galton-Watson tree constructed from offspring sequence in BFS order.

It is easy to see that the described procedure is linear on the number of vertices. Since we have an efficient method for constructing Galton-Watson trees from offspring sequences, as long as we can sample valid sequences uniformly in linear time, it would give way to a linear algorithm to sample uniform spanning trees.

Observe that all the valid sequences must have n elements that sum $n - 1$, since they indicate the offspring of each vertex and the root vertex is not a child of any other vertex. On top of that, from Equation 2.3 we see that the sequences are distributed as *multinomial* random variables with $n-1$ trials, each assuming one out of n possible values with equal probability $1/n$. Notice, however, that the problem of generating valid sequences here is not as trivial as it is with Prüfer sequences. In particular, while each valid sequence corresponds to a single tree if we establish a traversing order, not all sequences correspond to valid trees. As an example, sequences that start with zeros and sequences that end with non-zeros clearly do not correspond to any tree in the BFS order mentioned above.

The key idea of the multinomial method is exactly how to solve this problem. Devroye noted that for any sequence there is only one circular shift that corresponds to a valid tree for a given traversing order. The result is based on the standard rotation argument for partial sums, commonly referred as the Dvoretzky-Motzkin cycle lemma [49, 50]. The observation implies the following strategy for the simulation of size-constrained Galton-Watson trees:

1. Generate a sequence $\mathcal{S} = \{\xi_1, \ldots, \xi_n\}$ with multinomial distribution ($n-1$ trials, n events of equal probability $1/n$.);
2. Rotate $\mathcal{S}$ until it corresponds to a valid tree for a predefined traversing order;
3. Construct the tree according to the single valid order.

This procedure constitutes *the multinomial method*, which can be implemented in linear running time complexity, and results in a uniform spanning tree of the complete graph on n vertices.

Chapter 3

Approaches based on random walks and related work

A random walk is a stochastic process that models the movement of a particle (walker) in discrete space. The subject represents one of the most popular and important areas in probability theory, particularly in the field of Markov processes, with extensive theoretical developments and a wide range of applications [51, 52]. We denote by $(X_t)_{t\geq 0}$ the sequence of random states representing the position of the walker at discrete time instant t. A basic example is the random walk on $\mathbb{Z}$, the integer line. The particle starts on state $X_0 = 0$ and at any subsequent time t the particle moves to position $X_{t-1} - 1$ or $X_{t-1} + 1$ with equal probability. The exploration of this one-dimensional simple model can be traced back to the study of the "gambler's ruin problem" by Fermat and Pascal [53].

Besides the integer line, random walks can also take place in a variety of more complex spaces. The most commonly studied structures are graphs [54]. A random walk in a graph $G = (V, E)$ is simply a Markov chain process in which the states $(X_t)_{t\geq 0}$ represent the vertices, X_0 can be deterministic or random depending on the model, and each following state X_t is uniformly chosen between the adjacent vertices of X_{t-1}. The definition can be extended for graphs with weights on the edges ($w : E \to (0, \infty)$), in which case the next state is chosen with probability proportional the weight of the edge incident on to the current state.

The remaining of this chapter is dedicated to introducing *Aldous-Broder* and *Wilson's* algorithms, which demonstrate the fascinating relationship between random walks and uniform spanning trees (USTs). These algorithms are considered two fundamental pillars of the recent research on the UST topic, and are also the essential basis of this work.

3.1 Aldous-Broder algorithm

As pointed out in Section 2.1, uniform spanning trees have been studied at least since 1840's. As a consequence of Kirchhoff's work, the first endeavors to algorithmically generate uniform spanning trees were the determinant-based approaches. The first algorithms produced a uniform spanning tree in running time $O(mn^3)$ where $n = |V|$ and $m = |E|$ [24, 25]. It was only in the late 1980s that algorithms based on random walks emerged as a much more promising approach. Such algorithms began with the following notable result due to Aldous [29] and Broder [30]:

Theorem 3.1.1 (Aldous-Broder)**.** *Consider a simple random walk $(X_t)_{t>0}$ on $G = (V, E)$ starting from an arbitrary vertex $r \in V$. Let E_X be the set of edges $\bigcup_{t\geq 0} \big\{\{X_t, X_{t+1}\}$ such that $X_{t+1} \neq X_k$, for all $k \leq t\big\}$. Then $T = (V, E_X)$ is a random spanning tree of G with law* $\mathrm{UST}(G)$.

In order to generate a spanning tree T the random walk can stop at the cover time of G (the first time when the walker has visited all the vertices), as the tree will be completed exactly when the last node is visited. This improved the running time of generating a $\mathrm{UST}(G)$ to the cover time of G which has an expected value of $O(n \log n)$ for most graphs and $O(mn)$ for the worst graphs [31]. The algorithm derived from Theorem 3.1.1 is commonly referred as the *Alodous-Broder algorithm*, and is precisely described below:

Aldous-Broder($G = (V, E)$)

0) Set $E_X \leftarrow \varnothing$ and $X_0 \leftarrow r$, with $r \in V$ chosen arbitrarily;

- While $|E_X| \neq |V| - 1$:
  - Run a simple random walk $(X_t)_{t>0}$ on G, starting at X_0 and for each edge $e = \{X_t, X_{t+1}\}$ such that $X_{t+1} \neq X_k$ for all $k \leq t$, set $E_X \leftarrow E_X \cup \{e\}$;
- Return $T = (V, E_X)$.

As a remark, if the input graph has weighted edges, the algorithm using the weighted walker returns a tree with probability proportional to $\prod_{e\in T} w(e)$. The weighted trees generated according to this law are called *weighted uniform spanning trees*. Another interesting recent result due to Hu *et al.* is the *reverse* Aldous-Broder algorithm [55] . They showed that not only the set of first-entrance edges have the the law of a uniform spanning tree, but that this is also true for a tree constructed using the last-exit edges.

Since this result by Aldous and Broder, much research has been devoted to investigating the possibility of improving the complexity for generating uniform

spanning trees using random walks. Wilson [32] was the first to demonstrate the possibility of generating uniform spanning trees faster than the cover time.

3.2 Wilson's algorithm

In order to build a spanning tree, the random walk clearly needs to visit every vertex of the graph at least once. For this reason, it might seem unlikely that a random walk-based algorithm would generate a uniform spanning tree faster than the cover time of the graph. However, Wilson [32] proposed an algorithm based on loop-erased random walks to generate a UST(G) with a lower expected running time. As suggested by the name, a loop-erased random walk is defined as a trajectory of a random walk in which any loop is erased as soon as it is formed [56]. The fact that the path between a pair of vertices in a uniform spanning tree has the law of a loop-erased random walk was already known for some time [11, 57], but Wilson's algorithm described below was the first method to exploit this fact.

Wilson($G = (V, E), T_o$)

0) $V_X \leftarrow V(T_o)$ and $E_X \leftarrow E(T_o)$;
If $T_o = \varnothing$ then $V_X \leftarrow \{r\}$ with $r \in V$ chosen arbitrarily;

- While $V_X \neq V$:
  - Choose $v \in V \setminus V_X$ uniformly at random
  - Start a random walk on v and stop when it first visits a node in V_X
  - Let p_v be the path taken by the walker once loops have been erased
  - Let $V_X = V_X \cup V(p_v)$ and $E_X = E_X \cup E(p_v)$
- Return $T = (V_X, E_X)$.

Note that T_o is an initial tree of G but for now assumed to be empty, $T_o = \varnothing$ (it will be used in Section 4.3.1). Wilson showed that the expected running time of this algorithm is proportional to the *mean hitting time* of G. The hitting time h_{uv} is the expected number of steps a random walk takes to reach vertex v starting from vertex u and the *mean hitting time* of a graph G is the average hitting time considering all pairs of vertices. The mean hitting time of a graph is always less than the mean cover time (although its worst-case asymptotics is still $O(mn)$ [58]). Moreover, the mean hitting time can be significantly smaller for some graphs. On complete graphs for example, the mean cover time is $\Theta(n \log n)$ whereas the mean hitting time is $\Theta(n)$ (see Section 4.3.1 for a more detailed discussion). On the other hand, a well-known result reffered to as *Matthews bound* [59], showed that for any

graph, the cover time is never bigger than H_{n-1} times the mean hitting time (H_n is the n-th harmonic number which is $\Theta(\log n)$).

As with *Aldous-Broder*, Wilson's algorithm can also be applied to weighted graphs, in which case it returns weighted uniform spanning trees. On top of that, it also finds particular applications such as the generation of random arborescences and random spanning forests. Avena *et al.* [60] surveys some recent results on a certain parametric measure over random forests which can be sampled using a simple modification of Wilson's algorithm. The study includes some interesting applications as a powerful tool for analysing networks.

While both *Aldous-Broder* and *Wilson* algorithms are based on random walks, their behavior is fundamentally different in general. Consider the last few vertices to be added to the spanning tree. In *Aldous-Broder* the random walk may wander a long time before covering these vertices, while in *Wilson* they will be rather quickly added to the tree. In sharp contrast, when considering the first few vertices to be added to the spanning tree, *Aldous-Broder* will quickly cover them but *Wilson* may wonder a long time until it reaches its initial anchor node. Intuitively, *Aldous-Broder* is more efficient in the beginning, to generate the first vertices of the tree, and Wilson is more efficient in the end, to finish the random tree. Can this intuition be formally explored to combine the two approaches into a more efficient algorithm? This is the main theme addressed in the following chapters.

3.3 Recent Advancements

Wilson's algorithm opened the doors to algorithms that can generate USTs more efficiently than the cover time of a random walk. Since then, a sequence of improved algorithms have emerged in the literature. Kelner and Mądry [33] proposed a method for generating approximately uniform spanning trees by trying to skip unnecessary steps, i.e., steps over already-covered regions of the graph (referred as *shortcutting* strategy). The method returns a tree τ with probability $p(\tau)$, where $(1-\delta)/|\mathcal{T}_G| \leq p(\tau) \leq (1+\delta)/|\mathcal{T}_G|$ for some $\delta > 0$, in $\tilde{O}(m\sqrt{n}\log 1/\delta)$ time ($\tilde{O}$ refers to the *soft-O* notation, which is simply *big-O* ignoring logarithmic factors [61]). Later on, the strategy was proved to work for generating uniform (not approximately) spanning trees too [62], resulting in a procedure with $\tilde{O}(m\sqrt{n})$ expected running time.

Based on this last result, and exploring the effective resistance metric [63] and its relation to random walks, Mądry *et al.* [34] proposed an approach that improved the performance to $\tilde{O}(m^{4/3})$ (better on sufficiently sparse graphs). Finally, based on the just mentioned results and in a number of new facts about electrical flows, Schild [35] introduced another shortcutting strategy which resulted in a method that generates uniform spanning trees with running time almost-linear on the number of

edges ($O(m^{1+o(1)})$). It is worth mentioning that the effective resistance metric was also the basis for a very recent and promising new algorithm by Durfee *et al.* [64], which eschews from both determinant and random walk-based approaches and uses Gaussian elimination to generate USTs with high probability in $O(n^{5/3}m^{1/3})$.

Chapter 4

Aldous-Broder and Wilson transient equivalence on complete graphs and Hybrid Algorithm

In this chapter we provide an interpretation for the transient behavior of the *Aldous-Broder* and *Wilson's* algorithms that is instrumental in establishing one of our main results. A balls and urns model is defined and used to generate random spanning trees. Then, we show that this process is equivalent to the *Aldous-Broder* algorithm when running on the complete graph, including in its transient behavior. We use this model to show a transient equivalence between *Aldous-Broder* and *Wilson* on complete graphs. The key idea is to view both algorithms as constructing branches that are added to the tree being generated. This result leads to the *Hybrid* algorithm, which combines the other two approaches to generate uniform spanning trees more efficiently than running either algorithm separately.

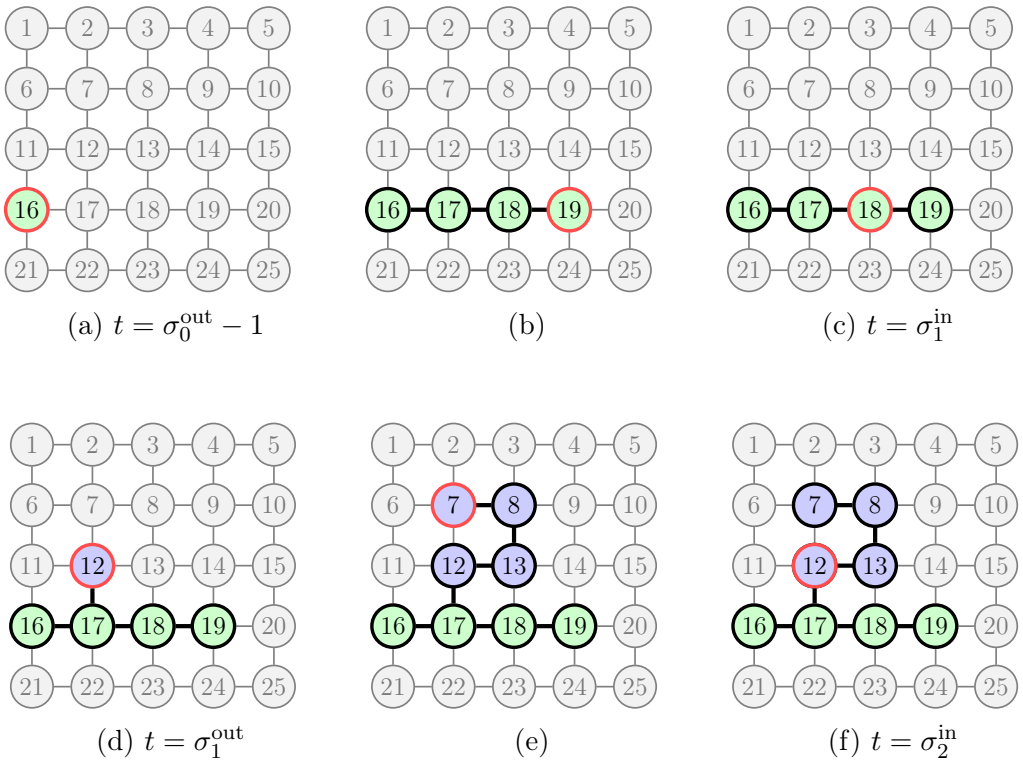
4.1 Aldous-Broder branch distribution

Consider the description of **Aldous-Broder**($G = V, E$) given is Section 3.1. Let V_t denote the set of vertices of G visited by the random walk up to time t, i.e., $V_t = \bigcup_{k=0}^{t} \{X_k\}$, and η the cover time of G. Consider the following double sequence of stopping times: $\sigma_0^{\text{out}} \equiv 1$, and for $i > 0$,

$$\sigma_i^{\text{in}} := \inf\{t > \sigma_{i-1}^{\text{out}} : X_t \in V_{t-1}\} \wedge \eta \,,$$
$$\sigma_i^{\text{out}} := \inf\{t > \sigma_i^{\text{in}} : X_t \notin V_{t-1}\} \wedge \eta \,.$$

Note that, since G is finite and connected, η is finite almost surely, and thus the stopping times above are all well-defined. Based on the stopping times above, we define the *branches* of the random walk $(X_t)_{t\geq 0}$ as follows: for every $i \geq 1$, the i-th

branch is the (self-avoiding) path with vertices $\{X_{(\sigma_{i-1}^{\text{out}}-1)}, \ldots, X_{(\sigma_i^{\text{in}}-1)}\}$. Note that, we refer to a branch as the path graph with vertices $X_{(\sigma_{i-1}^{\text{out}}-1)}, \ldots, X_{(\sigma_i^{\text{in}}-1)}$ and edges $\{X_{j-1}, X_j\}$, for $j = \sigma_{i-1}^{\text{out}}, \ldots, \sigma_i^{\text{in}} - 1$. Figure 4.1 illustrates the definition through an example of the evolution of the Aldous-broder algorithm, during the construction of the first two branches in a 5×5 grid graph. The position of the walker is represented by the red circle and the colored vertices are the already covered ones (each color indicates a different branch). Bold edges represent the set of first-entrance edges, which are included in the tree according to Theorem 3.1.1.



(a) $t = \sigma_0^{\text{out}} - 1$ (b) (c) $t = \sigma_1^{\text{in}}$

(d) $t = \sigma_1^{\text{out}}$ (e) (f) $t = \sigma_2^{\text{in}}$

Figure 4.1: Illustration of the branch construction process by the Aldous-Broder algorithm.

The represented events are:

(a) the random walk starts at time $t = 0$ in an arbitrary vertex (16). By definition, this vertex belongs to the first branch, since $\sigma_0^{\text{out}} \equiv 1$;

(b) while the walker visits new vertices, the corresponding path (vertices and edges) are added to the first branch;

(c) the first time the walker visits any already-covered vertex, the construction of the first branch is completed. This corresponds to the stopping time $t = \sigma_1^{\text{in}}$;

(d) after $t = \sigma_1^{\text{in}}$ the walker may wander a random number of steps (possibly zero) over already-covered vertices. The next time a vertex is visited for the first time (σ_1^{out}), another branch starts;

(e) as the random walk continues, while the following vertices are reached for the first time, they are added to the current branch, along with the corresponding edges;

(f) when a repeated vertex is visited, the current branch is finished. The process goes on until every vertex is visited and added to a branch.

We denote by $T^{AB}_{\sigma_i^{\text{in}}}$ the random sub-tree of G (not necessarily spanning) constructed by the Aldous-Broder algorithm when considering the first-entrance edges up to time σ_i^{in}, i.e., $\bigcup_{t<\sigma_i^{\text{in}}} \{\{X_t, X_{t+1}\} : X_{t+1} \neq X_k \, \forall k \leq t\}$, which corresponds to the edges in the first i branches. For example, the sub-trees highlighted in Figures 4.1c and 4.1f are, respectively, $T^{AB}_{\sigma_1^{\text{in}}}$ and $T^{AB}_{\sigma_2^{\text{in}}}$. Clearly, by Aldous-Broder theorem, T_η is a uniform spanning tree of G. In Figure 4.1 we show an example of the branches constructed by the Aldous-Broder algorithm in a uniform spanning tree of the complete graph with a hundred vertices.

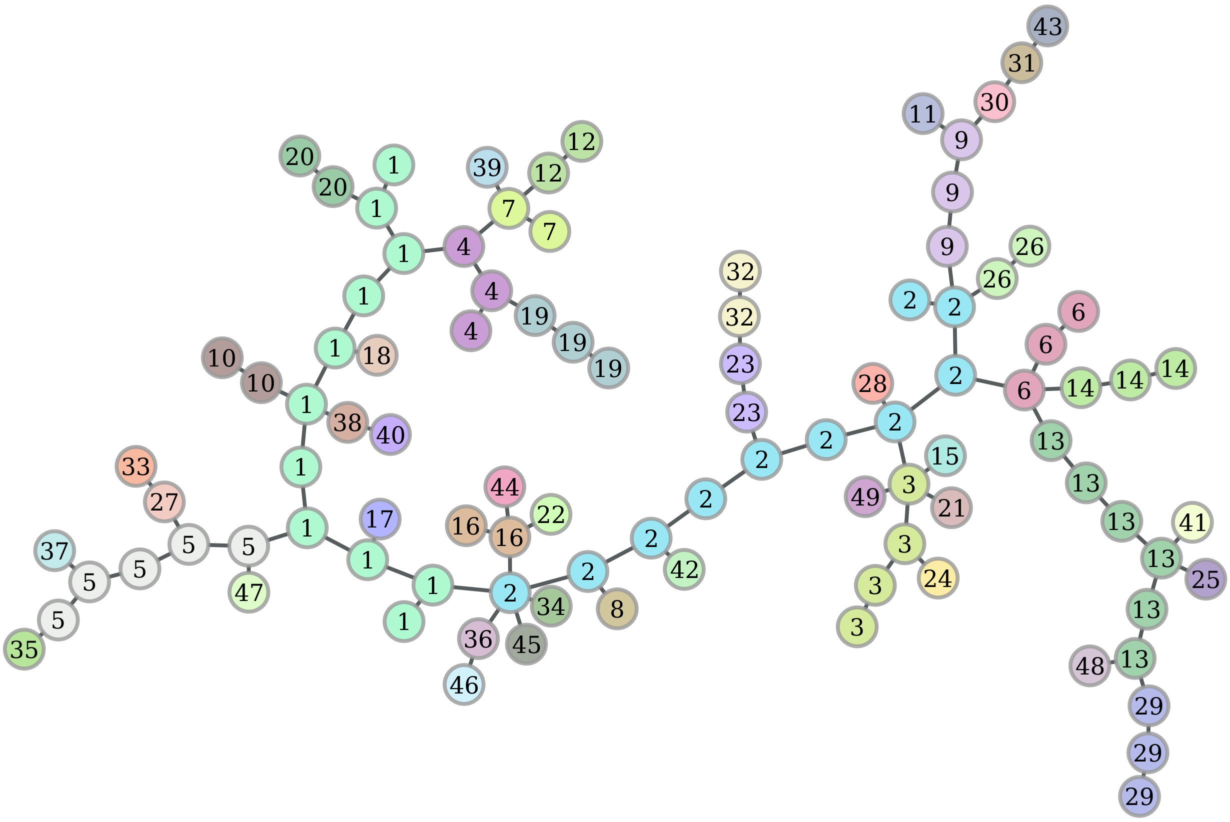


Figure 4.2: An example of a uniform spanning tree of a complete graph with a hundred vertices, generated by the *Aldous-Broder* algorithm. Each vertex is numbered according to its branch number.

4.1.1 Urn Model

Consider an urn $U = \{b_1, \ldots, b_n\}$ containing n uncolored (unlabelled) balls, and let $\{c_1, \ldots, c_n\}$ be a set of n distinct colors. Consider the following color-assignment process:

0) Set $A \leftarrow \varnothing$ and $i \leftarrow 1$;

1) Draw a ball b uniformly at random from the urn U:

   – if $b \in U \setminus A$ (the ball is uncolored), paint it with color c_i, return it to the urn and set $A \leftarrow A \cup \{b\}$;

   – if $b \in A$ (ball is already colored), return it to the urn, set $i \leftarrow i + 1$ and paint a uniformly chosen ball from $U \setminus A$ with color c_i;

2) If $A \neq U$, repeat item 1), otherwise stop.

Note that, the color-assignment process stops after n calls to 1), since one ball is colored at each call. Let $|C_i|$, for $i \leq n$, be the random variable denoting the number of balls painted with color c_i at the end of the process. The next lemma is straightforward.

Lemma 4.1.1. *For every* $h \in \{1, \ldots, n\}$, $\mathbb{P}_U(|C_1| = h) = \frac{h(n-1)!}{n^h(n-h)!}$. *Moreover, for every* $i > 1$, $k \in \{1, \ldots, n-1\}$, *and* $h \in \{1, \ldots, n-k\}$, $\mathbb{P}_U\left(|C_i| = h \,\middle|\, \sum_{j=1}^{i-1} |C_j| = k\right) = \frac{(k+h)(n-k-1)!}{n^h(n-k-h)!}$

Let $C = (|C_1|, \ldots, |C_n|)$ denote the random sequence obtained as a result of the color-assignment urn process (note that $|C_i| = 0$ if color c_i is not used). The following algorithm generates a labelled tree with $n+1$ vertices based on the outcome of the color-assignment process for a urn with n balls. We shall use the following notation: Given a path graph p, let $V(p)$ and $E(p)$ denote its vertex and edge set, respectively, and ℓ_p a leaf of p (vertex of degree 1).

Urn-Tree(C)

0) Set $V \leftarrow \{1, 2, \ldots, n+1\}$, and let $r \in V$ be chosen arbitrarily;
Set $V_T \leftarrow \{r\}$, $E_T \leftarrow \varnothing$;
Set $V \leftarrow V \setminus \{r\}$;

- For $i \in \{1, 2, \ldots, n\}$:
  - Let R be a uniform random subset of V with size $|C_i|$;
  - Let p be a path graph of size $|C_i|$ with the vertices of R placed in a random permutation;
  - Choose a leaf $u = \ell_p$ uniformly at random;
  - Choose a uniform random vertex $v \in V_T$;
  - Let $V_T \leftarrow V_T \cup V(p)$ and $E_T \leftarrow E_T \cup E(p) \cup \{(v, u)\}$;
  - Let $V \leftarrow V \setminus R$;
- Return $T = (V_T, E_T)$.

Note the input to the algorithm is the sequence $|C_i|$, $i = 1, \ldots, n$ and the output is a random labelled tree with $n + 1$ vertices. The tree is constructed by starting with an arbitrary vertex. Then for every iteration, a path of size $|C_i|$ is randomly constructed using the vertices not yet in the tree. This path is then attached to exiting tree by choosing a random leaf of the path and a random vertex in the tree (both uniformly). The process finishes when all colors have been considered.

To illustrate, consider the following urn obtained after the color-assignment process, where $c_1 =$ ■, $c_2 =$ ■, $c_3 =$ ■ and $c_4 =$ ■:

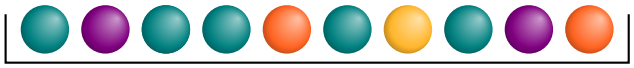


Figure 4.3: Example of urn with $n = 10$ balls obtained after the color-assignment process.

Figure 4.4 shows an example of tree that can be obtained from the *Urn-Tree* algorithm, with the above urn given as input.

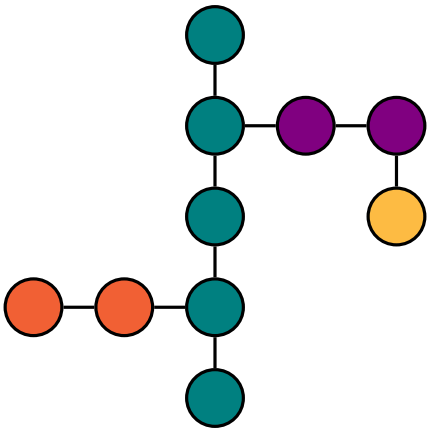


Figure 4.4: Example of tree that can be obtained from the *Urn-Tree* algorithm after the color-assignment illustrated in Figure 4.3

Note that, if $N := \max\{i \leq n : |C_i| \neq 0\}$ denotes the number of used colors in the color-assignment process, then $|C_j| = 0$ for $j > N$. This means that the *Urn-Tree* algorithm can stop after N steps, since the tree has been fully constructed at this step.

Consider the *Aldous-Broder* algorithm on a complete graph with n vertices and the *Urn-Tree* on an urn with $n-1$ balls. While both algorithms build random trees with n vertices, they correspond to significantly different random processes. An important difference is their running time. While the expected running time of *Aldous-Broder* is $\Theta(n \log n)$ for a complete graph with n vertices, the *Urn-Tree* algorithm has a worst case running time of $\Theta(n)$ for an urn with $n-1$ balls. Note that the color assignment procedure requires time $\Theta(n)$. The random subset selection and the random permutation in the *Urn-Tree* algorithm can be implemented using the *Knuth Shuffle* procedure (which has a running time in the order of the subset size) [65]. Since random subsets and permutations are generated until the whole set of nodes is covered, this procedure also requires time $\Theta(n)$.

Remarkably, the following theorem establishes not only that the *Urn-tree* algorithm generates a uniform spanning tree of the complete graph, but also that the *Urn-tree* is equivalent to the transient behavior of the *Aldous-Broder* algorithm in the stopping time that induce *branches*.

Let T_i^U denote the random tree generated by *Urn-tree* algorithm up to the attachment of the i-th path graph. Note that the number of vertices in T_i^U, henceforth denoted by $|T_i^U|$, is equal to $1 + \sum_{j=1}^{i} |C_j|$.

Theorem 4.1.2 (Urn-Tree and Aldous-Broder transient equivalence)**.** *The random sub-tree $T^{AB}_{\sigma_i^{\text{in}}}$ generated by Aldous-Broder algorithm on a complete graph of n vertices and T_i^U on an urn U with $(n-1)$ balls, when initialized with the same vertex r, have the same distribution in the sense that $T_i^U \overset{d}{=} T^{AB}_{\sigma_i^{\text{in}}}$, for all $i = 1, \ldots, n-1$*

Proof. Let Θ_i be the random variable denoting the number of edges in the i-th branch of a random walk X on a complete graph with n vertices, as previously defined. We begin observing that $\Theta_1 \stackrel{d}{=} |C_1|$, i.e., if $\mathbb{P}_X$ denote the probability measure defined on the space of trajectories of $(X_t)_{t>0}$, we have

$$\mathbb{P}_X(\Theta_1 = h) = \mathbb{P}_U(|C_1| = h) \ , \quad \text{for every } h \in \{1, \dots, n-1\} \ .$$

The above follows because the random walk in Aldous-Broder can be interpreted as painting vertices as it traverses through the graph until it hits a vertex has already been painted. This process is identical to the urn process since the graph is complete.

Moreover, for $i > 1$ and $k \in \{1, \dots, n-2\}$,

$$\mathbb{P}_X\left(\Theta_i = h \ \middle| \ |T_{\sigma_{i-1}^{\text{in}}}| = k\right) = \mathbb{P}_U\left(|C_i| = h \ \middle| \ 1 + \sum_{j=1}^{i-1} |C_j| = k\right) \ ,$$

for every $h \in \{1, \dots, n-1-k\}$. The intuition here is similar to the case $i = 1$. However, the random walk starts its journey on a graph with k vertices already painted. It paints new vertices until it hits a vertex already painted. Again this random procedure is identical to the urn process since the graph is complete.

Recall that in the *Urn-Tree* algorithm the path graph of size $|C_1|$ is connected to an initial vertex r chosen arbitrarily, while each path graph of size $|C_i|$, with $i > 1$, is connected to a uniformly chosen vertex in T_{i-1}^U. To finish the proof it is enough to show that this is also the case for the branches in Aldous-Broder. The first branch is clearly connected to a single vertex r also chosen arbitrarily, since $X_0 = r$. Consider now the subsequent branches. Given that the i-th branch is connected to $X_{(\sigma_{i-1}^{\text{out}}-1)}$, we must show that the distribution of $X_{(\sigma_{i-1}^{\text{out}}-1)}$ is uniform over the vertices of $T_{\sigma_{i-1}^{\text{in}}}^{AB}$. Note that at time σ_{i-1}^{in}, the $(i-1)$-th branch has just ended and by symmetry $X_{\sigma_{i-1}^{\text{in}}}$ is uniformly distributed over $T_{\sigma_{i-1}^{\text{in}}}^{AB}$. At time $t = \sigma_{i-1}^{\text{in}} + 1$, either the i-th branch starts, i.e., $\sigma_{i-1}^{\text{out}} = t$, which happens with probability $(n - |T_{\sigma_{i-1}^{\text{in}}}^{AB}|)/n$, or the random walk moves to a vertex $T_{\sigma_{i-1}^{\text{in}}}^{AB}$ with probability $|T_{\sigma_{i-1}^{\text{in}}}^{AB}|/n$, thus uniformly. This in particular implies that for every $k \geq 1$, the distribution of $X_{\sigma_{i-1}^{\text{out}}-1}$ conditioned on $\sigma_{i-1}^{\text{out}} = \sigma_{i-1}^{\text{in}} + k$, is uniform on $T_{\sigma_{i-1}^{\text{in}}}^{AB}$. Thus, $X_{(\sigma_{i-1}^{\text{out}}-1)}$ is uniform over the vertices of $T_{\sigma_{i-1}^{\text{in}}}^{AB}$ and $T_i^U \stackrel{d}{=} T_{\sigma_i^{\text{in}}}^{AB}$. □

Corollary 4.1.3. *The random tree T_{n-1}^U produced by Urn-Tree algorithm on an urn with $n-1$ balls is a uniform spanning tree of the complete graph with n vertices.*

Proof. From Theorem 4.1.2, when G is a complete graph, $T_i^U \stackrel{d}{=} T_{\sigma_i^{\text{in}}}^{AB}$, for every $i \leq n-1$. Then by Theorem 3.1.1 and observing that $\sigma_{n-1}^{\text{in}} = \eta$, the result follows. □

While the *Urn-tree* algorithm is a linear time algorithm for generating USTs for the complete graph, it is clearly not the first. A simple procedure proposed by

Aldous generates a UST for the complete graph using n uniform choices on $1, \ldots, n$ and a random shuffle, thus also requiring time $\Theta(n)$ (see Algorithm 2 in [29]) However, the notion of branches in the *Urn-tree* algorithm and its transient equivalence with *Aldous-Broder* will be important in establishing an important connection with Wilson's algorithm (in the next section).

4.2 Wilson's branch distribution

Recall that Wilson's algorithm constructs loop-erased paths with vertices not yet in the tree. This suggests a very natural definition for the tree branches. Specifically, let $\hat{\sigma}_0 \equiv 0$ and $T^W_{\hat{\sigma}_0}$ the single initial vertex r, and recursively define $\hat{\sigma}_i := \inf\left\{t > \hat{\sigma}_{i-1} : X_t \in T^W_{\hat{\sigma}_{i-1}}\right\}$, where $T^W_{\hat{\sigma}_{i-1}}$ denotes the tree build by the *Wilson* algorithm up to time $\hat{\sigma}_{i-1}$. The i-th *branch* $\hat{p}_i$ corresponds to the loop erasure path of the walk $(X^i_0, \ldots, X_{\hat{\sigma}_i})$, where X^i_0 is a uniform vertex in $V \setminus V(T^W_{\hat{\sigma}_{i-1}})$. As before, $\hat{p}_i$ is a path graph where $V(\hat{p}_i)$ and $E(\hat{p}_i)$ denote its vertex and edge sets, respectively and its size will be denoted by $|\hat{p}_i| = |V(\hat{p}_i)| - 1$. Moreover, $T^W_{\hat{\sigma}_{i-1}}$ is the random tree corresponding to the first $(i-1)$ branches generated by the Wilson's algorithm.

While Wilson's algorithm produces uniform spanning trees for any labelled graph, it is challenging to analyse the distribution of its *branches* in the general case. Thus, the complete graph is first considered, and the analysis of the branch distribution unveils a strong relationship between *Aldous-Broder* and *Wilson* which shall be discussed in Section 4.3.

The first important property of the branch distribution on complete graphs is that the labels of the vertices in any branch are simply a random permutation of a random subset of the labels. This is due to the fact that every vertex is structurally identical and also connected to every other vertex. So, the probability that a vertex appears in a specific branch, and in a specific order, does not depend on the vertex itself. This implies that the distribution of the branches is entirely defined by their sizes.

However, explicitly computing the branch size distribution in *Wilson* algorithm is not trivial in general due to its loop erasing mechanism. Specifically, if $\hat{p}_i(t)$ denotes the i-th branch at time t before completion, i.e., with $t < \hat{\sigma}_i$, the size of $\hat{p}_i(t)$ may in the next step:

1) Increase by one and stop: if $X_{t+1} \in V(T^W_{\hat{\sigma}_{i-1}})$, equivalently $\hat{\sigma}_i = t + 1$;

2) Increase by one and continue: if $X_{t+1} \notin V(T^W_{\hat{\sigma}_{i-1}})$ and $X_{t+1} \notin V(\hat{p}_i(t))$;

3) Decrease to $\ell \in \{1, \ldots, |\hat{p}_i(t)|\}$ and continue: if $|\hat{p}_i(t)| \geq 2$, and $X_{t+1} \in V(\hat{p}_i(t))$ and is in the ℓ-th position of the current loop erased random walk.

On a complete graph with n vertices, for $i > 1$ and every $k \in \{i, \dots, n-1\}$ we have that $\mathbb{P}\big(\hat{\sigma}_i = t+1 \mid |T^W_{\hat{\sigma}_{i-1}}| = k\big) = \frac{k}{n-1}$. Moreover, if we define $\Delta\hat{p}_i(t) := |\hat{p}_i(t+1)| - |\hat{p}_i(t)|$, then for every $h \in \{1, \dots, n-k-1\}$:

$$\mathbb{P}\big(\Delta\hat{p}_i(t) = 1 \big| |\hat{p}_i(t)| = h, |T^W_{\hat{\sigma}_{i-1}}| = k\big) = \frac{n-k-h}{n-1}\,,$$
$$\mathbb{P}\big(\Delta\hat{p}_i(t) = -\ell \big| |\hat{p}_i(t)| = h, |T^W_{\hat{\sigma}_{i-1}}| = k\big) = \frac{1}{n-1}\,, \qquad \forall \ell \in \{1, \dots, h-1\}\,.$$

The case of the first branch size corresponds to setting $k = 1$. In light of the above, to study the branch size distribution in *Wilson*'s algorithm it suffices to analyse the absorbing Markov chain illustrated in Figure 4.5 where the state of the Markov chain corresponds to $\hat{p}_i(t)$, namely the size of the i-th branch.

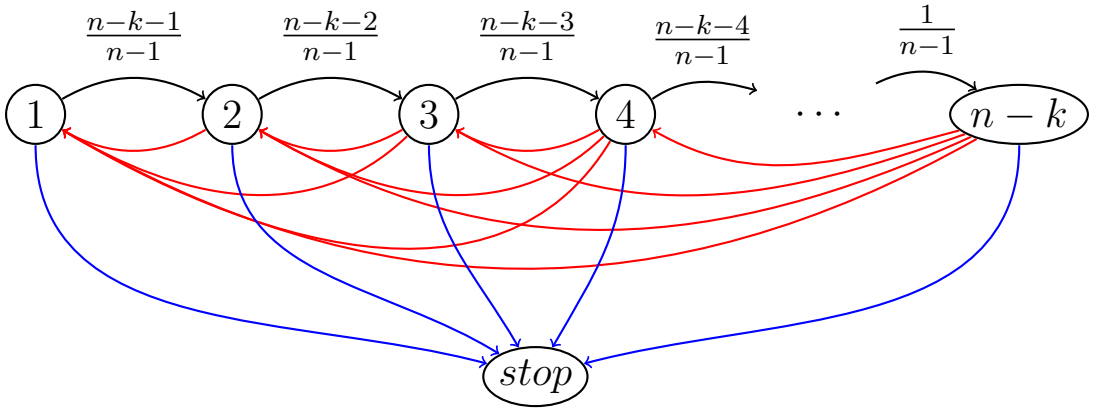


Figure 4.5: Absorbing Markov chain representing the *Wilson*'s algorithm construction of a branch $\hat{p}_i$ in the a complete graph with n vertices, given that $|T^W_{\hat{\sigma}_{i-1}}| = k$. The numbered states represent the branch size h and *stop* is an absorbing state indicating the end of the branch construction. The process always starts from $h = 1$. The blue, black and red arrows correspond to the events 1), 2) and 3), respectively.

Specifically, the probability $\mathbb{P}(|\hat{p}_i(\hat{\sigma}_i)| = h \big| |T^W_{\hat{\sigma}_{i-1}}| = k)$ that the i-th branch has size h, given that the number of vertices in the previously constructed branches is k, corresponds to the probability of making a transition to the *stop* state from state h (i.e., probability of *absorption* from state h).

Consider the ergodic Markov chain in Figure 4.6 obtained by *lumping* together the *stop* state and the $h = 1$ state. The probability of absorption from state h in the original chain (starting from state $h = 1$) corresponds to the probability that, in the ergodic chain (starting from $h = 1$), the blue edge incident to state h is the first blue edge traversed. This latter probability in the ergodic chain matches exactly its stationary distribution. Intuitively, since the ergodic chain regenerates every time a blue edge is traversed, the probability that a specific blue edge is the first one traversed corresponds to the long term relative frequency of times that it is traversed when compared to the other blue edges. This relative frequency, in turn,

is proportional to the long term relative frequency of times that the chain visits state h. Since the probability of crossing a blue edge given that the chain is in a specific state does not depend on the state (the transition probability associated to every blue edge is the same), the proportionality constant is the same for every state. Thus, the long term relative frequency of times that the blue edge incident to state h is traversed is the same as the stationary distribution of h in the ergodic chain.

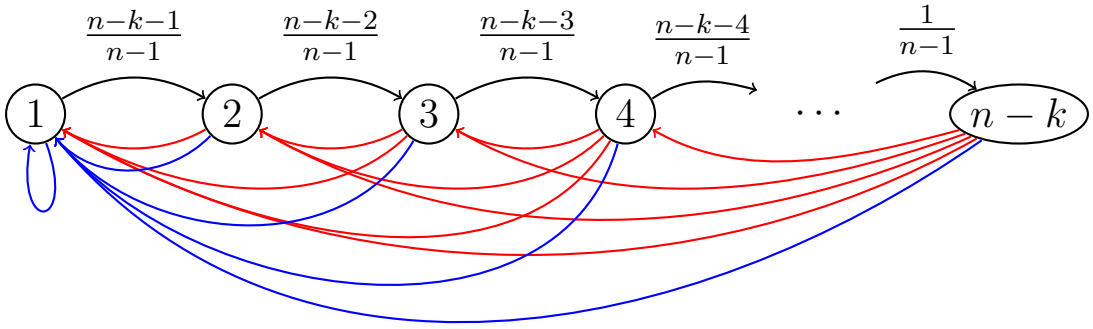


Figure 4.6: Ergodic Markov chain obtained by lumping together state *stop* and state 1 in the absorbing chain in Figure 4.5.

The above procedure establishes a clear strategy to determine the branch size distribution of *Wilson* on complete graphs, which consists in determining the stationary distribution of the Markov chains illustrated in Figure 4.6. Note that the Markov chain that must be analyzed depends only on n and the number of vertices already in the tree, namely k. Nevertheless, explicitly solving the balance equations and finding a closed-form solution for the stationary distribution is often not trivial.

However, verifying that a candidate distribution is a stationary distribution for a Markov chain is rather easy. Interestingly, the next lemma shows that the distributions in Lemma 4.1.1 are the stationary distribution for the corresponding Markov chain.

Lemma 4.2.1. *For every $k \in \{1, \dots, n-1\}$ the stationary distribution of the corresponding Markov chain depicted in Figure 4.6 is:*

$$\pi(h) = \frac{(k+h)(n-k-1)!}{n^h(n-k-h)!} \qquad \text{for } h \in \{1, \dots, n-k\} .$$

Proof. Every state h with $h \neq 1$, has one incoming *black arrow* from the state $h-1$ and one *red arrow* from each state $h+\ell$. Therefore, we have:

$$\pi(h) = \pi(h-1)\frac{n-k-(h-1)}{n-1} + \sum_{\ell=h+1}^{n-k} \pi(\ell)\frac{1}{n-1} \ .$$

The state $h = 1$, in contrast, has $n-k$ incoming *blue arrows* and $n-k-1$ *red arrows*. Then:

$$\pi(h) = \sum_{\ell=2}^{n-k} \pi(\ell)\frac{k+1}{n-1} + \pi(1)\frac{k}{n-1}$$

The result follows since $\pi(h)$ from the lemma statement is the solution of the presented balance equations. □

4.3 Aldous-Broder and Wilson transient equivalence

Remarkably, Lemma 4.2.1 showed that the stationary distribution of the Markov chains that represent the branch sizes in *Wilson* are identical to the conditional branch size distribution of *Aldous-Broder* when considering a complete graph. Thus, the following interesting result follows immediately:

Theorem 4.3.1 (*Aldous-Broder* and *Wilson* transient equivalence)**.** *Let Aldous-Broder and Wilson algorithms have the same initialization for the initial vertex r and let the input graph G be a complete graph. Then, there exist two sequences of stopping times $(\hat{\sigma}_i)_{i\geq 0}$ (for Wilson) and $(\sigma_i^{\text{in}})_{i\geq 0}$ (for Aldous-Broder) such that, for every $i \geq 0$*

$$T^W_{\hat{\sigma}_i} \stackrel{d}{=} T^{AB}_{\sigma_i^{\text{in}}} \ .$$

Proof. From Lemma 4.2.1 and Theorem 4.1.2. □

Note that the step-by-step construction of the UST on complete graphs by *Aldous-Broder* and *Wilson* are fundamentally distinct. However, Theorem 4.3.1 shows that if these two processes are observed at specific stopping times, the "transient" (partial) trees built by both algorithms are identically distributed. Interestingly, these stopping times correspond to the time a branch is constructed in either algorithm. Moreover, this equivalence regards also the labels of the vertices and not only the (partial) tree structure. Of course, it requires the algorithms to have the same initialization, in the sense that vertex r must the chosen identically by both algorithms.

4.3.1 Hybrid Algorithm

Theorem 4.3.1 shows that *Aldous-Broder* to *Wilson* are statistically equivalent on the appropriate stopping times. This suggests that a UST can be constructed by one algorithm until a certain stopping time and then finished by the other algorithm. This gives rise to a hybrid algorithm that can switch between *Aldous-Broder* and *Wilson* (and back) at the corresponding stopping times.

In particular, consider the following hybrid algorithm that starts with *Aldous-Broder* and constructs i branches and then switches to *Wilson* to finish the job:

Hybrid($G = (V, E), i$)

0) Set $E_X \leftarrow \varnothing$ and $X_0 \leftarrow r$, with $r \in V$ chosen arbitrarily;

- Until $t = \sigma_i^{\text{in}}$:
  - Run a simple random walk $(X_t)_{t>0}$ on G, starting at X_0 and for each edge $e = \{X_t, X_{t+1}\}$ such that $X_{t+1} \neq X_k$ for all $k \leq t$, set $E_X \leftarrow E_X \cup \{e\}$;
- Set $V_X \leftarrow \{X_0, \ldots, X_{\sigma_i^{\text{in}}}\}$;
- Return **Wilson**$(G, T_o = (V_X, E_X))$;

Note that *Wilson* receives as parameter the partial tree constructed by *Aldous-Broder* and start execution given this partial tree (see algorithm in Section 3.2). The next proposition states that the *Hybrid* algorithm returns a uniform spanning tree of the complete graph G.

Proposition 4.3.2. *Let $T^{AB}_{\sigma_i^{\text{in}}}$ be a random tree of the complete graph G, constructed by the first i branches of Aldous-Broder algorithm. Wilson algorithm with initial condition $T^{AB}_{\sigma_i^{\text{in}}}$ returns a UST(G).*

The proposed hybrid algorithm is particularly interesting in terms of running time complexity. Clearly, the *cover time* is dominated by the last steps of the random walk in *Aldous-Broder* while the *hitting time* is highest in the first steps of the random walk in *Wilson*. In fact, the most time consuming steps of the two algorithms are very similar: at the end, the random walk in *Aldous-Broder* must find the last remaining vertex to be added to the tree, and at the beginning, the random walk in *Wilson* must find the single anchor node. On a complete graph with n vertices, it is easy to conclude that both these times are geometrically distributed, with success probability $1/(n-1)$. Thus, the proposed hybrid algorithm has the potential do reduce the running time necessary to generate a UST.

The following lemma, due to Wilson [32], characterizes the time complexity of Wilson's algorithm in terms of the number steps taken by the random walk:

Lemma 4.3.3 (Wilson [32]). *Let $\omega(G)$ be the mean hitting time of G. The expected number of steps taken by the random walk in* ***Wilson(****G, $T_o = \varnothing$) is $2\omega(G)$.*

On a complete graph, the *mean hitting time* is n. Thus, the random walk takes on average $2n$ steps to return a spanning tree when running *Wilson*. For *Aldous-Broder* on a complete graph, it is well known that the expected number of random walk steps required to return a spanning tree is nH_n, where H_n denotes the n-th harmonic number (this comes from the equivalence between the cover time of a random walk on a complete graph and the *coupon collector's problem*).

The following proposition establishes the advantages of the hybrid algorithm. Indeed, it avoids the long phases of both algorithms and exploits their good phases, creating a synergy to efficiently generate a UST. In fact, the hybrid algorithm is more efficient than either of the algorithms alone.

Proposition 4.3.4. *The expected number of random walk steps made by Hybrid($G, 1$) is $n + \Theta(\sqrt{n})$.*

Proof. The expected number of steps made by **Hybrid**$(G, 1)$ is the sum of the average number of steps it takes for **Aldous-Broder**(G) to construct the first branch and the average number of steps **Wilson**(G, T_o) takes to complete the tree.

Consider an urn with n labelled balls and the process of drawing balls one at a time uniformly at random with replacement. Note that the time required for *Aldous-Broder* to construct the first branch is identical to the time required to draw the first repeated ball. It is known that this has expected time $\sqrt{\pi n/2} + \Theta(1/\sqrt{n})$ (it is a variation of the birthday problem. See for example Section 1.2.11.3 in [66]).

From Lemma 4.3.3, the expected number of steps it takes for *Wilson* to build a uniform spanning tree of a complete graph G is $2n$. Moreover, on average, n of those steps are made to construct the first branch ($Geo(1/n)$). Therefore, n additional steps on average are required to finish the tree after the construction of the first branch of *Aldous-Broder*. Hence, the expected number of random walk steps required by **Hybrid**$(G, 1)$ to build a UST(G) is $n + \Theta(\sqrt{n})$. □

In order to provide more insight into the running time of these algorithms, consider the average number of steps taken by the random walk to include at least k edges in the spanning tree. Figure 4.7 shows a simulation result directly comparing the three algorithms on a complete graph with $n = 1000$ vertices (results shown are the average over one thousand tree generations for each algorithm). As expected, *Aldous-Broder* is efficient in the beginning (when almost every step brings a new edge to the tree) but requires a long time in the end, to finish the tree. In contrast, *Wilson* is linear on n but it requires a long time include its first edges on the tree (exactly 1000 steps, on average). The hybrid algorithm starts efficiently (following

along *Aldous-Broder* and remains efficient (switching to *Wilson*). Note that the *Hybrid* is approximately two times faster than *Wilson*, taking roughly 1000 steps to build the tree.

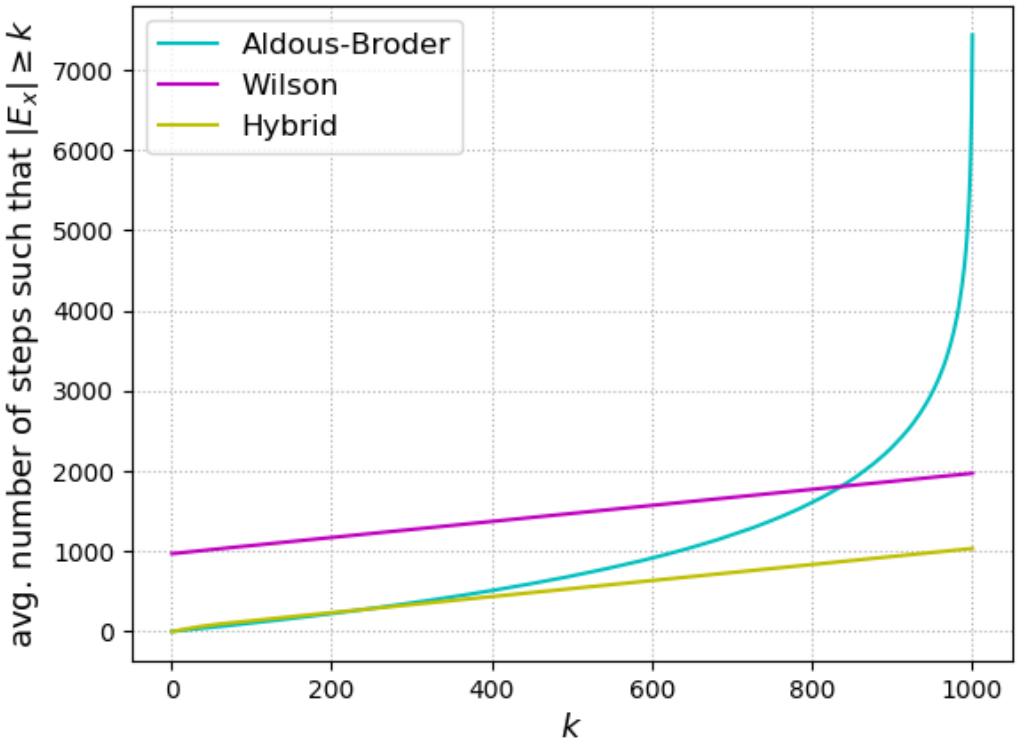


Figure 4.7: Average running time measured in number of random walk steps to generate a UST. Each curve shows the average number of random walk steps required by each algorithm to include at least k edges in the tree being constructed ($n = 1000$, average over 1000 independent tree generations).

While the proposed hybrid algorithm first runs *Aldous-Broder* and then switches to *Wilson*, it is perfectly possible to have a reverse hybrid algorithm, where *Wilson* runs and constructs a few branches and then *Aldous-Broder* finishes the job. In this case, a uniform random node within the built tree would have to be chosen to start the random walk after switching to *Aldous-Broder* at an appropriate stopping time. In fact, it is also possible to go back and forth between the algorithms, always switching at the appropriate stopping time. However, from the perspective of running time, the proposed hybrid is the most efficient.

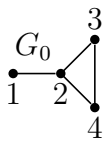
Chapter 5

Towards a two-stage general framework for generating uniform spanning trees

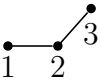
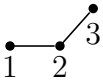
The previous chapter presented a two-stage approach to construct uniform spanning trees for complete graphs: (i) build i branches of the spanning tree using *Aldous-Broder*; (ii) switch to *Wilson* to finish the tree. The strategy not only maintains the uniformity across the spanning trees, but also takes less time to generate the tree. Can this *Hybrid* algorithm be used to construct uniform spanning trees for any graph?

While *Hybrid* clearly constructs spanning trees of general graphs (and it is expected to be more efficient than either *Aldous-Broder* or *Wilson*), there is no guarantee such trees will be uniformly distributed. Unfortunately, the random sub-trees generated by the first branch of Aldous-Broder and Wilson's algorithm are not identically distributed on a general graph. In particular, Theorem 4.3.1 does not hold for arbitrary graphs as the following simple example shows.

Let G_0 be the graph depicted below:



Let $T^{AB}_{\sigma_1^{\text{in}}}$ be the random sub-tree generated by the first branch of *Aldous-Broder* algorithm on G_0, starting from a uniformly chosen vertex, and $T^{W}_{\hat{\sigma}_1}$ the random sub-tree generated by the first branch of *Wilson*'s algorithm on G_0, then

$$\mathbb{P}\left(T^{AB}_{\sigma_1^{\text{in}}} = \right.$$  $$\left.\right) = \frac{1}{12} \quad \text{while} \quad \mathbb{P}\left(T^{W}_{\hat{\sigma}_1} = \right.$$  $$\left.\right) = \frac{1}{9}\,.$$

Note that if we generate the first branch using *Wilson*, i.e., $T^W_{\hat{\sigma}_1}$, and then run **Wilson**$(G, T^W_{\hat{\sigma}_1})$ the final tree will be an uniform spanning tree of G, as expected. However, if we generate the first branch of *Aldous-Broder*, i.e., $T^{AB}_{\sigma_1^{\text{in}}}$, and then run **Wilson**$(G, T^{AB}_{\sigma_1})$, the final spanning tree is not necessarily uniform when an arbitrary graph is considered, as the above example shows. However, can this idea of a two-stage procedure be saved? In what follows, this question is addressed and a promising answer is provided.

Given a connected graph G, let S_G denote the set of all possible sub-trees (not necessarily spanning) of G. We shall denote by ζ an element of S_G, while τ shall denote an element of $\mathcal{T}_G$, i.e., a spanning tree of G. With a slight abuse of notation, we shall write $\zeta \subseteq \tau$ to denote that ζ is a sub-tree of τ. Given a sub-tree $\zeta \in S_G$, we denote by $\mathcal{T}_G(\zeta) = \{\tau \in \mathcal{T}_G : \tau \supseteq \zeta\}$, the set of all spanning trees of G which contain the sub-tree ζ, and by $|\mathcal{T}_G(\zeta)|$ its cardinality.

Let Y be a random variable taking values in S_G, whose distribution satisfies the following condition: for all $\tau, \tau' \in \mathcal{T}_G$

$$\sum_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau}} \frac{1}{|\mathcal{T}_G(\zeta)|} \mathbb{P}(Y = \zeta) = \sum_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau'}} \frac{1}{|\mathcal{T}_G(\zeta)|} \mathbb{P}(Y = \zeta) \,, \tag{5.1}$$

Intuitively, the above condition says that Y cannot differentiate between τ and τ' when generating sub-trees. The condition is subtle because different sub-trees ζ will induce different sets of spanning trees $\mathcal{T}_G(\zeta)$ that have different sizes. Note that it is not sufficient for every tree to contain Y with the same probability. The condition also imposes a restriction on the sizes $|\mathcal{T}_G(\zeta)|$ of the induced tree sets. Thus, Y must not introduce bias with respect to τ on the weighted average across all sub-trees of τ.

Consider the following two-stage procedure to generate a uniform spanning tree of an arbitrary graph G:

Proposition 5.0.1. *The following two-stage (*ts*) procedure returns a uniform spanning tree of a connected graph G.*

1. *Draw a random variable Y taking values in S_G whose distribution satisfies Equation* (5.1)*;*

2. *Draw a spanning tree of G from the set $\mathcal{T}_G(Y)$ uniformly.*

Proof. Let T denote the random spanning tree of G returned by the two-stage procedure (note that the two-stage procedure always return an element of $\mathcal{T}_G$). Let $\mathbb{P}_{\text{ts}}(T = \tau)$ denote the probability that the two-stage procedure returns $\tau \in \mathcal{T}_G$. To show that T is uniformly distributed on $\mathcal{T}_G$, it suffices to show that $\mathbb{P}_{\text{ts}}(T = \tau) =$

$\mathbb{P}_{\text{ts}}(T = \tau')$, for every $\tau, \tau' \in \mathcal{T}_G$. Given $\tau \in \mathcal{T}_G$, it holds that

$$\mathbb{P}_{\text{ts}}(T = \tau) = \mathbb{P}_{\text{ts}}\left(T = \tau, \bigcup_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau}} \{Y = \zeta\} \right)$$
$$= \sum_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau}} \mathbb{P}_{2.}(T = \tau | Y = \zeta) \mathbb{P}_{1.}(Y = \zeta) = \sum_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau}} \frac{1}{|\mathcal{T}_G(\zeta)|} \mathbb{P}_{1.}(Y = \zeta) \ ,$$

and the claim follows form Equation (5.1). □

While Proposition 5.0.1 provides a two-stage procedure to generate uniform spanning tree of any graph G, it is not clear whether this procedure can give rise to an efficient algorithm. In particular, both the first and second stages must be implemented efficiently. Some promising answers are described in what follows.

SECOND-STAGE: Interestingly, *Wilson* is an efficient algorithm that can be used to implement the second stage in Proposition 5.0.1. In particular, given a connected graph G, and $\zeta \in S_G$ a sub-tree (not spanning) of G, **Wilson(G, ζ)** returns a spanning tree of G in the set $\{\tau \in \mathcal{T}_G : \tau \supseteq \zeta\}$ uniformly. This is formally stated in the next lemma.

Lemma 5.0.2 (Second stage)**.** *Let $G = (V, E)$ be a connected graph and ζ be a sub-tree (not spanning) of G. Wilson's algorithm with initial condition ζ returns a spanning tree of G which is uniformly distributed on the set $\{\tau \in \mathcal{T}_G : \tau \supseteq \zeta\}$.*

Proof. The proof is based on the following two steps:

a) We collapse the sub-tree ζ into a single vertex, denoted by r, ignoring its internal edges while keeping all the edges connecting vertices in $V(\zeta)$ with vertices in $V \setminus V(\zeta)$. Note that this procedure gives rise to a new graph G' which may have multiple edges incident on r. In particular, there exists a set of (external) vertices $\widehat{V} \subset V \setminus V(\zeta)$ such that the number of edges between $v \in \widehat{V}$ and r is the same as the number of neighbors that v has in $V(\zeta)$, denoted by d'_v. Thus, $v \in \widehat{V}$ has d'_v multiple edges incident on r in G'.

   From the perspective of a simple random walk, the multiple edges incident to a vertex $v \in \widehat{V}$ could be treated as a single weighted edge, with weight equal to d'_v; this weighted graph is denoted by G'_*. Note that all other edges have weight 1.

b) We use a variation of Wilson's algorithm for weighted graph that generates a spanning tree of the weighted graph with probability proportional to the product of the edge weights in the corresponding tree [32].

Note that every tree in the set $\{\tau \in \mathcal{T}_G : \tau \supset \zeta\}$ corresponds to one and only one spanning tree of G', and vice versa (assuming each of the multiple edges can be identified individually). Thus, it is sufficient to show that *Wilson* on G', with initial condition r, returns a spanning tree in $\mathcal{T}_{G'}$ uniformly.

The set of spanning trees of G' can be partitioned accordingly to which of the vertices in $\widehat{V}$ are connected to r, ignoring which specific edge among the multiple edges are in the spanning tree. Specifically, for $\tau \in \mathcal{T}_{G'}$, if we denote by $N_r(\tau)$ the set of neighbors of r in τ, we have that

$$\mathcal{T}_{G'} = \bigcup_{I \subseteq \widehat{V}} \underbrace{\{\tau \in \mathcal{T}_{G'} : N_r(\tau) = I\}}_{:= S_I}$$

Note that $\{S_I : I \subseteq \widehat{V}\}$ (the set of spanning trees of G' in which r is connected to the vertex I) is a partition of $\mathcal{T}_{G'}$, and thus for any spanning tree $\tau \in \mathcal{T}_{G'}$ there exists a unique $I \subseteq \widehat{V}$ such that $\tau \in S_I$; specifically $\tau \in S_{N_r(\tau)}$. Hence, if T denotes the random spanning tree generated by Wilson's algorithm on G', we have that $\mathbb{P}(T = \tau) = \mathbb{P}\left(T = \tau | S_{N_r(\tau)}\right) \mathbb{P}\left(S_{N_r(\tau)}\right)$, for any $\tau \in \mathcal{T}_{G'}$.

Now let us observe that $\mathbb{P}\left(S_{N_r(\tau)}\right)$ corresponds to the probability that Wilson's algorithm on the weighted graph G'_* (no more multiple edges) returns a specific spanning tree in $\mathcal{T}_{G'_*}$. By point *b*), the probability that Wilson's algorithm on the weighted graph G'_* returns $\tilde{\tau} \in \mathcal{T}_{G'_*}$ is equal to $C \prod_{v \in N_r(\tilde{\tau})} d'_v$, where C is a normalizing constant; in fact, the edges $\{r, v\}$ of the tree, with $v \in N_r(\tilde{\tau})$ have weights d'_v, whereas all other edges have unitary weights. As far as $\mathbb{P}\left(T = \tau | S_{N_r(\tau)}\right)$ is concerned, since the random walk is simple, for any $v \in N_r(\tau)$, the walker chooses one (and only one) of the possible multiple edges incident on r uniformly, and independently across the different $v \in N_r(\tau)$. Thus, $\mathbb{P}\left(T = \tau | S_{N_r(\tau)}\right) = 1/\left(\prod_{v \in N_r(\tau)} d'_v\right)$. Hence, for any $\tau \in \mathcal{T}_{G'}$ the probability $\mathbb{P}(T = \tau)$ does not depend on τ, and thus is uniform on $\mathcal{T}_{G'}$. □

FIRST-STAGE: The main concern with the first stage is constructing and sampling a random variable whose distribution satisfies Equation (5.1). Below we list some examples.

Example 1 (Wilson's branches): Let G be an arbitrary connected graph G and let $T^W_{\hat{\sigma}_i}$ be the random sub-tree of G generated by the first i branches of *Wilson*'s algorithm. Then $T^W_{\hat{\sigma}_i}$ satisfies Equation (5.1). To see the latter, note that if we run the two-stage procedure, when the first stage consists in drawing $T^W_{\hat{\sigma}_i}$ and the second stage in running Wilson's algorithm with initial condition $T^W_{\hat{\sigma}_i}$, the resulting algorithm is equivalent to Wilson's (classical) algorithm on G starting from a single vertex, whose final outcome is indeed uniform on $\mathcal{T}_G$. Thus, if we denote

by T the random tree generated by the two-stage procedure when the first stage is Wilson's first i branches and the second is Wilson's algorithm with the obtained initial condition, we have that for every $\tau, \tau' \in \mathcal{T}_G$, it holds that $\mathbb{P}_{\text{ts}}(T = \tau) = \mathbb{P}_{\text{ts}}(T = \tau')$. Moreover

$$\mathbb{P}_{\text{ts}}(T = \tau) = \sum_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau}} \mathbb{P}_{2.}(T = \tau | T^W_{\hat{\sigma}_i} = \zeta) \mathbb{P}(T^W_{\hat{\sigma}_i} = \zeta) = \sum_{\substack{\zeta \in S_G \\ \mathcal{T}_G(\zeta) \ni \tau}} \frac{1}{|\mathcal{T}_G(\zeta)|} \mathbb{P}(T^W_{\hat{\sigma}_i} = \zeta) \ ,$$

and thus, $T^W_{\hat{\sigma}_i}$ satisfies Equation (5.1).

Example 2 (Aldous-Broder branches on complete graphs): Let G be a complete graph an let $T^{AB}_{\sigma_i^{\text{in}}}$ be the random sub-tree of G generated by the first i branches of *Aldous-Broder* algorithm. Then $T^{AB}_{\sigma_i^{\text{in}}}$ satisfies Equation (5.1), for every i. The latter follows directly from Example 1, together with Theorem 4.3.1.

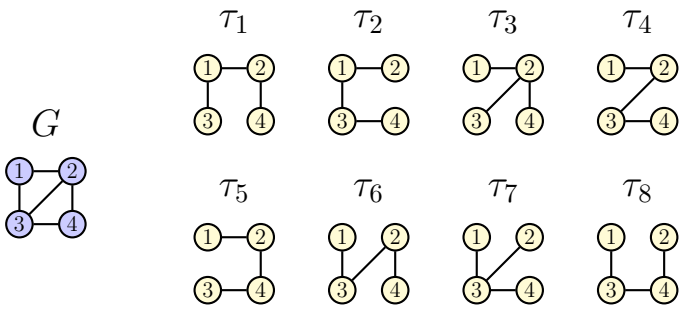
It is worthwhile mentioning that a similar result will not hold in general graphs. Specifically, if $T^{AB}_{\sigma_i^{\text{in}}}$ denotes the random sub-tree generated by the first i branches of *Aldous-Broder* algorithm on an arbitrary graph G, the distribution of $T^{AB}_{\sigma_i^{\text{in}}}$ will not, in general, satisfy Equation (5.1). For example, the sub-tree generated by the first branch of *Aldous-Broder* algorithm on G_0, given at the beginning of this section, (starting from a uniform vertex), does not satisfy Equation (5.1).

Example 3 (Uniform edge in edge-transitive graphs): Let $G = (V, E)$ be an edge-transitive graph, an let Y be a random variable uniformly distributed on E. Then Y satisfies Equation (5.1). As a matter of fact, for every $e \in E$, we have that $\mathbb{P}(Y = e) = 1/|E|$ and moreover, since the graph G is edge transitive, for every two spanning trees $\tau, \tau' \in \mathcal{T}_G$ it holds that

$$\sum_{\substack{e \in E \\ \mathcal{T}_G(e) \ni \tau}} \frac{1}{|\mathcal{T}_G(e)|} = \sum_{\substack{e \in E \\ \mathcal{T}_G(e) \ni \tau'}} \frac{1}{|\mathcal{T}_G(e)|} = \frac{n-1}{|\mathcal{T}_G(e_o)|} \ ,$$

where e_o is an arbitrary edge of E. The result follows since every spanning tree has $(n-1)$ edges, and thus appears in exactly $(n-1)$ terms in the sum. Moreover, since the graph is edge transitive, the number of spanning trees traversing an specific edge $e \in E$ is the as any other another edge $e' \in E$ due to the automorphism between e and e'.

Interestingly, choosing an uniform edge for a graph that is not edge-transitive will not necessarily make the requirements, as the next simple example shows. Consider the following graph G and its spanning trees $\tau_1, \ldots, \tau_8$:



The cardinality of $\mathcal{T}_G(e)$ for $e \in \{(1,2),(1,3),(2,4),(3,4)\}$ is five since each of these edges appear in exactly five spanning trees. However, the cardinality of $\mathcal{T}_G((2,3)))$ is four as edge $(2,3)$ appears in exactly four spanning trees. Thus, choosing an edge uniformly at random and then choosing a spanning tree uniformly at random given the chosen edge will generate a bias towards spanning that have the edge $(2,3)$. In fact, the probability of choosing τ_i is $12/100$ when $i \in \{1,2,5,8\}$ and $13/100$ when $i \in \{3,4,6,7\}$.

Example 4 (Aldous-Broder first branch on a cycle): Let G be a cycle graph on n vertices, henceforth denoted by $\mathcal{C}_n$ and $(X_t)_{t\geq 0}$ be a simple random walk on $\mathcal{C}_n$. Let $\sigma_1^{\text{in}} := \inf\{t \geq 0 : X_t \in \cup_{k=0}^{t-1}\{X_k\}\}$ and denote by $T^{AB}_{\sigma_1^{\text{in}}}$ the sub-tree generated by the first-entrance edges up to time σ_1^{in}. The random variable $T^{AB}_{\sigma_1^{\text{in}}}$ satisfies Equation (5.1). To see the latter, let us first observe that if $\zeta \in S_{\mathcal{C}_n}$ is a sub-tree of $\mathcal{C}_n$, and $E(\zeta)$ denotes the set of edges of ζ, it holds that

$$\mathbb{P}\left(T^{AB}_{\sigma_1^{\text{in}}} = \zeta\right) = \frac{1}{n}2^{-|E(\zeta)|}\,.$$

Moreover, given $\zeta \in S_{\mathcal{C}_n}$, the set of spanning tree of $\mathcal{C}_n$ containing ζ satisfies $|\mathcal{T}_{\mathcal{C}_n}(\zeta)| = n - |E(\zeta)|$. Using that all spanning trees of $\mathcal{C}_n$ are isomorphic to each other, for every $\tau, \tau' \in \mathcal{T}_{\mathcal{C}_n}$, we obtain that

$$\sum_{\substack{\zeta \in S_{\mathcal{C}_n} \\ \mathcal{T}_{\mathcal{C}_n}(\zeta)\ni\tau}} \frac{1}{n-|E(\zeta)|}\frac{1}{n}2^{-|E(\zeta)|} = \sum_{\substack{\zeta \in S_{\mathcal{C}_n} \\ \mathcal{T}_{\mathcal{C}_n}(\zeta)\ni\tau'}} \frac{1}{n-|E(\zeta)|}\frac{1}{n}2^{-|E(\zeta)|}\,.$$

Example 5 (Sub-tree generated by a simple random walk on a cycle): Let G be a cycle graph on n vertices, henceforth denoted by $\mathcal{C}_n$. Given $k \geq 0$, let us consider the random sub-tree T_k^X induced by a simple random walk $(X_t)_{t\geq 0}$ on $\mathcal{C}_n$ up to time k, starting from a uniformly chosen vertex of $\mathcal{C}_n$, i.e., T_k^X is the tree with vertex set $\cup_{t=0}^{k}\{X_t\}$ and edge set $\cup_{t=1}^{k}\{X_{t-1}, X_t\}$. Then, if η denotes the cover time of $\mathcal{C}_n$, for every $k \geq 1$ the random variable $T^X_{(k\wedge\eta)-1}$ satisfies Equation (5.1). To see the latter, let $\zeta \in S_{\mathcal{C}_n}$ be a sub-tree of $\mathcal{C}_n$, and denote by $E(\zeta)$ the set of edges of ζ. Note that, $\mathbb{P}\left(T^X_{(k\wedge\eta)-1} = \zeta\right) = 0$, if $k < |E(\zeta)|$, whereas $\mathbb{P}\left(T^X_{(k\wedge\eta)-1} = \zeta\right)$ is a

function only of n, k and $|E(\zeta)|$, if $k \geq E(\zeta)$ (i.e., given n and k sub-trees having the same number of edges will have the same probability). Moreover, given $\zeta \in S_{\mathcal{C}_n}$, the set of spanning tree of $\mathcal{C}_n$ containing ζ satisfies $|\mathcal{T}_{\mathcal{C}_n}(\zeta)| = n - |E(\zeta)|$. Using that all spanning trees of $\mathcal{C}_n$ are isomorphic to each other, for every $\tau, \tau' \in \mathcal{T}_{\mathcal{C}_n}$, we obtain that, for any fixed k,

$$\sum_{\substack{\zeta \in S_{\mathcal{C}_n} \\ \mathcal{T}_{\mathcal{C}_n}(\zeta) \ni \tau}} \frac{1}{n - |E(\zeta)|} \mathbb{P}\left(T^X_{(k \wedge \eta)-1} = \zeta\right) = \sum_{\substack{\zeta \in S_{\mathcal{C}_n} \\ \mathcal{T}_{\mathcal{C}_n}(\zeta) \ni \tau'}} \frac{1}{n - |E(\zeta)|} \mathbb{P}\left(T^X_{(k \wedge \eta)-1} = \zeta\right) .$$

Examples 4 and 5 consider a cycle graph G with n vertices. Uniform spanning trees of G can be very efficiently generated by simply removing from G an edge chosen uniformly at random. Thus, while the two-stage procedure in these two examples does not lead to an efficient algorithm, the goal here is to provide a random variable based on random walks that satisfies Equation (5.1) along with an efficient sampling procedure for this variable. This could shed light on crafting such random variable for a broader class of graphs.

5.1 Random trees linearly biased by a given structure

In this section we discuss another noteworthy application of Lemma 5.0.2, which is stated in the following Theorem.

Theorem 5.1.1. *Given $G = (V, E)$ a complete graph with n vertices, and any $\zeta \in S_G$. A spanning tree $\tau \in \mathcal{T}_G$ can be generated with probability that is proportional to the number of sub-trees of τ that are isomorphic to ζ in $O(n)$ time.*

Proof. Consider T, the random variable (random tree) denoting the output of **Wilson**(G, ζ), where G is a complete graph on n vertices and ζ is any sub-tree of G (not necessarily spanning). Now consider T^*, the random tree obtained by uniformly shuffling the labels of T. Clearly, since G is complete, T^* also assumes a spanning tree in $\mathcal{T}_G$.

Consider the probability $\mathbb{P}_\zeta\left(T^* = \tau\right)$ that T^* is a specific spanning tree $\tau \in \mathcal{T}_G$. The probability of generating τ is the sum of probabilities of all the sequences of events that can result in it. Since we shuffle the labels in the end, each sub-tree ζ^* of τ that is isomorphic to ζ corresponds to one such event, then:

$$\mathbb{P}_\zeta(T^* = \tau) = \sum_{\substack{\zeta^* \in S_G: \\ \zeta^* \subseteq \tau \,\wedge\, \zeta^* \simeq \zeta}} \mathbb{P}\big(\textbf{Wilson}(G, \zeta^*) = \tau\big)$$

where $\zeta^* \simeq \zeta$ indicates that there exists an isomorphism between ζ and ζ^*, and $\{\mathbf{Wilson}(G, \zeta^*) = \tau\}$ represents the event that Wilson's algorithm with initial condition ζ^* returns τ.

From Lemma 5.0.2, we know that $\mathbb{P}(\mathbf{Wilson}(G, \zeta^*) = \tau) = 1/|\mathcal{T}_G(\zeta^*)|$. Moreover, the fact that G is complete tells us that $|\mathcal{T}_G(\zeta')| = |\mathcal{T}_G(\zeta'')|$ for all $\zeta' \simeq \zeta''$. Then, the probability of generating a specific tree τ is proportional to the cardinality of the sum, which is exactly the number of sub-trees of τ that are isomorphic to ζ. Finally, note that the procedure consists of running *Wilson* on a complete graph, which takes $O(n)$ time (as discussed in Section 4.3.1) and performing a random shuffle of the labels, which also takes $O(n)$ running time [65]. Then, the total running time complexity of the procedure is $O(n)$ □

The distribution of the number of sub-trees, such as leaves or vertices of degree k, in uniform spanning trees have been extensively studied by combinatorialists [67]. The above procedure can be used to efficiently sample spanning trees of the complete graph with probability biased by given sub-trees. For example, consider running the procedure with the initial sub-tree ζ being a star graph with k nodes. The resulting process generates spanning trees with probability proportional to the number of vertices of degree k. If ζ is a path of length l, the procedure generates trees proportionally to the number of paths of length l contained in it. The generation of random trees constrained by given structures finds application in a variety of areas, from decision making algorithms to real-life network optimization problems [36–38].

Chapter 6

Conclusion

This work considered the problem of efficiently sampling from the uniform probability measure the set of spanning trees of a graph $\mathcal{T}_G$. The problem has been studied for many decades and, even so, it continues to draw attention, and it is constantly present in the main publishing venues in theoretical computer science. Our focus was on the random walk based approaches, considered the most-promising strategy for generating uniform spanning trees efficiently. The random walk approach is built upon the celebrated Aldous-Broder and Wilson's algorithms.

We started by investigating the transient dynamics of the two algorithms (which is considerably different in general) by looking at the partial trees at special sequences of stopping times. Such times decompose the trees in *branches* that we introduced and defined for both *Aldous-Broder* and *Wilson*. For the Aldous-Broder algorithm, we showed how a simple, but rather interesting, balls and urns model can be used to illustrate the construction of branches on a complete graph. Making use of this interpretation and of the branch definition for Wilson's algorithm, we proved that the sub-trees built by both algorithms with corresponding number of branches are identical in distribution for complete graphs. Since *Aldous-Broder* constructs branches faster in the beginning and slower in the end, and *Wilson* does exactly the opposite, we used their equivalence to propose an hybrid algorithm that switches from *Aldous-Broder* to *Wilson* and is more efficient than either of the two algorithms (by a constant factor when considering *Wilson*).

This result motivated an investigation of the possibility of applying the idea on a more general setting, beyond complete graphs. We showed that the equivalence of Aldous-Broder and Wilson's branch distribution does not hold on any graph, and so, whereas the hybrid algorithm does produce spanning trees of general graphs, the uniformity is lost. In spite of that, we proposed a two-stage framework for the general problem and showed that the trees generated by the procedure are uniform. The first stage consists of sampling a random variable Y from S_G, the set of all sub-trees of G, according to a special distribution. The second stage consists of

drawing uniformly a spanning tree from the set $\mathcal{T}_G(Y)$ of spanning trees of G that contain Y.

We showed that Wilson's algorithm can be used in the second phase. This is a consequence of Lemma 5.0.2, which shows the interesting fact that Wilson's algorithm can be used to generate trees that are uniform, conditioned on containing any sub-tree given as input. However, this framework requires finding a process to generate Y. It is noteworthy that the Hybrid algorithm turns out to be one particular use case of the framework in which Aldous-Broder algorithm is used to efficiently sample the random variable Y satisfying its requirements. On top of that, we provided a few other examples that illustrate the usage of the two-stage procedure.

Lastly, we discussed another application of Lemma 5.0.2. We introduced a simple modification to the framework in order to sample random trees on n vertices with the following distribution. Given any sub-tree ζ and a complete graph G, we showed how to sample a tree τ with probability proportional to the number of sub-trees contained in τ that are isomorphic to ζ, in time linear on the number of vertices of G.

6.1 Future work

The main topic for future work regards finding other settings where the framework can be applied to generate uniform spanning trees efficiently. Since Wilson's algorithm not only meets the requirements for the second-stage, but is also quite efficient for the task, the main question to be considered is how to tackle the first-stage. Some ideas for investigation are:

- are there other classes of graphs (beyond the complete graph and the cycle graph) where the Aldous-Broder branches can be used in the first-stage?
- are there other stopping time sequences that make the transient structure of Aldous-Broder satisfy the first stage?
- is there a different random walk based strategy (or any strategy) for sampling sub-trees efficiently that satisfies the first-stage?

Another interesting idea that arises from Lemma 5.0.2 is:

- Consider a graph G and let G' be a connected sub-graph of G. Let τ denote a spanning tree of G and consider its intersection with G', namely τ'. Note that τ' is a spanning forest of G'. Clearly, the uniform measure over random spanning trees of G induces a distribution over spanning forests of G'. Can we describe this distribution? Is it possible to sample from it efficiently?

The answer to this intriguing question could give way to a variation of the proposed two-stage framework where first τ' is generated and then τ is generated conditioned on τ'. The second phase might be accomplished by a variation of Wilson's algorithm. If the first phase is resolved, the framework could give lead to an efficient algorithm to generate USTs for arbitrary graphs.

References

[1] DIJKSTRA, E. W., OTHERS. "A note on two problems in connexion with graphs", *Numerische mathematik*, v. 1, n. 1, pp. 269–271, 1959.

[2] HART, P. E., NILSSON, N. J., RAPHAEL, B. "A formal basis for the heuristic determination of minimum cost paths", *IEEE transactions on Systems Science and Cybernetics*, v. 4, n. 2, pp. 100–107, 1968.

[3] MOY, J. "OSPF version 2", *RFC 2328*, 1998.

[4] PERLMAN, R. "An algorithm for distributed computation of a spanning tree in an extended lan", *ACM SIGCOMM Computer Communication Review*, v. 15, n. 4, pp. 44–53, 1985.

[5] CALEFFI, M., FERRAIUOLO, G., PAURA, L. "Augmented tree-based routing protocol for scalable ad hoc networks". In: *IEEE International Conference on Mobile Adhoc and Sensor Systems*, pp. 1–6, 2007.

[6] KRUSKAL, J. B. "On the shortest spanning subtree of a graph and the traveling salesman problem", *Proceedings of the American Mathematical society*, v. 7, n. 1, pp. 48–50, 1956.

[7] FELZENSZWALB, P. F., HUTTENLOCHER, D. P. "Efficient graph-based image segmentation", *International journal of computer vision*, v. 59, n. 2, pp. 167–181, 2004.

[8] LYONS, R., PERES, Y. *Probability on trees and networks*, v. 42. Cambridge University Press, 2017.

[9] KIRCHHOFF, G. "Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird", *Annalen der Physik*, v. 148, n. 12, pp. 497–508, 1847.

[10] ALDOUS, D., GRIMMETT, G. R., HOWARD, C. D., et al. *Probability on discrete structures*, v. 110. Springer Science & Business Media, 2013.

[11] BURTON, R., PEMANTLE, R. "Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances", *The Annals of Probability*, pp. 1329–1371, 1993.

[12] FORMAN, R. "Determinants of Laplacians on graphs", *Topology*, v. 32, n. 1, pp. 35–46, 1993.

[13] BENJAMINI, I., LYONS, R., PERES, Y., et al. "Special invited paper: uniform spanning forests", *Annals of probability*, pp. 1–65, 2001.

[14] KASSEL, A., WU, W. "Transfer current and pattern fields in spanning trees", *Probability Theory and Related Fields*, v. 163, n. 1-2, pp. 89–121, 2015.

[15] JÁRAI, A. A., REDIG, F., SAADA, E. "Approaching criticality via the zero dissipation limit in the abelian avalanche model", *Journal of Statistical Physics*, v. 159, n. 6, pp. 1369–1407, 2015.

[16] KENYON, R. "Spanning forests and the vector bundle Laplacian", *The Annals of Probability*, v. 39, n. 5, pp. 1983–2017, 2011.

[17] HOORY, S., LINIAL, N., WIGDERSON, A. "Expander graphs and their applications", *Bulletin of the American Mathematical Society*, v. 43, n. 4, pp. 439–561, 2006.

[18] GHARAN, S. O., SABERI, A., SINGH, M. "A randomized rounding approach to the traveling salesman problem". In: *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 550–559, 2011.

[19] ASADPOUR, A., GOEMANS, M. X., MĄDRY, A., et al. "An O (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem", *Operations Research*, v. 65, n. 4, pp. 1043–1061, 2017.

[20] GALLAGHER, M., RYAN, A. "Learning to play Pac-Man: An evolutionary, rule-based approach". In: *IEEE Congress on Evolutionary Computation. CEC'03.*, v. 4, pp. 2462–2469, 2003.

[21] HARRISON, F., HOSSEINI, A., MCDONALD, M. "Endogenous anxiety and stress responses in water maze and Barnes maze spatial memory tasks", *Behavioural brain research*, v. 198, n. 1, pp. 247–251, 2009.

[22] GUPTA, B., SEHGAL, S. "Survey on techniques used in autonomous maze solving robot". In: *IEEE Next Generation Information Technology Summit (Confluence)*, pp. 323–328, 2014.

[23] BUCK, J. *Mazes for programmers: code your own twisty little passages*. Pragmatic Bookshelf, 2015.

[24] GUENOCHE, A. "Random spanning tree", *Journal of Algorithms*, v. 4, n. 3, pp. 214–220, 1983.

[25] KULKARNI, V. G. "Generating random combinatorial objects", *Journal of Algorithms*, v. 11, n. 2, pp. 185–207, 1990.

[26] COLBOURN, C. J., DEBRONI, B. M., MYRVOLD, W. J. "Estimating the coefficients of the reliability polynomial", *Congressus Numerantium*, v. 62, pp. 217–223, 1988.

[27] COLBOURN, C. J., DAY, R. P., NEL, L. D. "Unranking and ranking spanning trees of a graph", *Journal of Algorithms*, v. 10, n. 2, pp. 271–286, 1989.

[28] COLBOURN, C. J., MYRVOLD, W. J., NEUFELD, E. "Two algorithms for unranking arborescences", *Journal of Algorithms*, v. 20, n. 2, pp. 268–281, 1996.

[29] ALDOUS, D. J. "The random walk construction of uniform spanning trees and uniform labelled trees", *Journal on Discrete Mathematics*, v. 3, n. 4, pp. 450–465, 1990.

[30] BRODER, A. Z. "Generating random spanning trees", *IEEE Symposium on Foundations of Computer Science (FOCS)*, v. 89, pp. 442–447, 1989.

[31] BRODER, A. Z., KARLIN, A. R. "Bounds on the cover time", *Journal of Theoretical Probability*, v. 2, n. 1, pp. 101–120, 1989.

[32] WILSON, D. B. "Generating random spanning trees more quickly than the cover time". In: *ACM Symposium on Theory of Computing (STOC)*, pp. 296–303, 1996.

[33] KELNER, J. A., MĄDRY, A. "Faster generation of random spanning trees". In: *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 13–21, 2009.

[34] MĄDRY, A., STRASZAK, D., TARNAWSKI, J. "Fast generation of random spanning trees and the effective resistance metric". In: *SIAM Symposium on Discrete algorithms (SODA)*, pp. 2019–2036, 2015.

[35] SCHILD, A. "An almost-linear time algorithm for uniform random spanning tree generation". In: *Symposium on Theory of Computing (STOC)*, pp. 214–227, 2018.

[36] LI, X., WONG, W. H. "Sampling motifs on phylogenetic trees", *Proceedings of the National Academy of Sciences (PNAS)*, v. 102, n. 27, pp. 9481–9486, 2005.

[37] DEO, N., KUMAR, N. "Computation of constrained spanning trees: A unified approach". In: *Network Optimization*, pp. 196–220. Springer, 1997.

[38] LIU, F. T., TING, K. M., FAN, W. "Maximizing tree diversity by building complete-random decision trees". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 605–610. Springer, 2005.

[39] CAYLEY, A. "A theorem on trees", *Quarterly Journal of Mathematics*, v. 23, pp. 376–378, 1889.

[40] IACOBELLI, G., FIGUEIREDO, D. R., BARBOSA, V. C. "Counting trees with random walks", *Expositiones Mathematicae*, v. 37, n. 1, pp. 96–102, 2019.

[41] PRÜFER, H. "Neuer Beweis eines Satzes über Permutationen", *Archiv für Mathematik and Physik*, v. 27, pp. 142–144, 1918.

[42] CHEN, H.-C., WANG, Y.-L. "An efficient algorithm for generating Prüfer codes from labelled trees", *Theory of Computing Systems*, v. 33, n. 1, pp. 97–105, 2000.

[43] DEVROYE, L. "Nonuniform random variate generation", *Handbooks in operations research and management science*, v. 13, pp. 83–121, 2006.

[44] GOLDSCHMIDT, C. "A short introduction to random trees", *Mongolian Mathematical Journal*, v. 20, pp. 53–72, 2016.

[45] BIENAYMÉ, I.-J. "De la loi de multiplication et de la durée des familles", *Société Philomatique Paris Extraits*, v. 5, n. 37-39, pp. 4, 1845.

[46] WATSON, H. W., GALTON, F. "On the probability of the extinction of families", *The Journal of the Anthropological Institute of Great Britain and Ireland*, v. 4, pp. 138–144, 1875.

[47] VAN DER HOFSTAD, R. *Random graphs and complex networks*, v. 1. Cambridge university press, 2016.

[48] DEVROYE, L. "Simulating size-constrained Galton–Watson trees", *SIAM Journal on Computing*, v. 41, n. 1, pp. 1–11, 2012.

[49] COMTET, L. *Advanced Combinatorics: The art of finite and infinite expansions*. Springer Science & Business Media, 2012.

[50] DERSHOWITZ, N., ZAKS, S. *The cycle lemma and some applications*. Relatório técnico, Computer Science Department, Technion, 1982.

[51] SPITZER, F. *Principles of random walk*, v. 34. Springer Science & Business Media, 2013.

[52] BHATTACHARYA, R. N., WAYMIRE, E. C. *Stochastic processes with applications*. SIAM, 2009.

[53] TODHUNTER, I. *A History of the Mathematical Theory of Probability: From the Time of Pascal to that of Laplace*. Cambridge University Press, 2014.

[54] LOVÁSZ, L., OTHERS. "Random walks on graphs: A survey", *Combinatorics, Paul erdos is eighty*, v. 2, n. 1, pp. 1–46, 1993.

[55] HU, Y., LYONS, R., TANG, P. "A reverse Aldous/Broder algorithm", *arXiv preprint arXiv:1907.10196*, 2019.

[56] LAWLER, G. F., OTHERS. "A self-avoiding random walk", *Duke Mathematical Journal*, v. 47, n. 3, pp. 655–693, 1980.

[57] PEMANTLE, R. "Choosing a spanning tree for the integer lattice uniformly", *The Annals of Probability*, v. 19, n. 4, pp. 1559–1574, 1991.

[58] BRIGHTWELL, G., WINKLER, P. "Maximum hitting time for random walks on graphs", *Random Structures & Algorithms*, v. 1, n. 3, pp. 263–276, 1990.

[59] MATTHEWS, P. "Covering problems for Brownian motion on spheres", *The Annals of Probability*, v. 16, n. 1, pp. 189–199, 1988.

[60] AVENA, L., CASTELL, F., GAUDILLIÈRE, A., et al. "Random forests and networks analysis", *Journal of Statistical Physics*, v. 173, n. 3-4, pp. 985–1027, 2018.

[61] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to algorithms*. MIT press, 2009.

[62] MĄDRY, A. *From graphs to matrices, and back: new techniques for graph algorithms*. PhD Dissertation, Massachusetts Institute of Technology, 2011.

[63] TETALI, P. "Random walks and the effective resistance of networks", *Journal of Theoretical Probability*, v. 4, n. 1, pp. 101–109, 1991.

[64] DURFEE, D., KYNG, R., PEEBLES, J., et al. "Sampling random spanning trees faster than matrix multiplication". In: *ACM Symposium on Theory of Computing (STOC)*, pp. 730–742, 2017.

[65] KNUTH, D. E. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[66] KNUTH, D. E. *The art of computer programming: Fundamental algorithms*, v. 3. Addison-Wesley Publishing Company, 1973.

[67] ALDOUS, D. "The continuum random tree. I", *The Annals of Probability*, pp. 1–28, 1991.