This is a *topics course*, and our ambition is to introduce you to a number of exciting objects and motifs from modern probability theory. All these topics will involve random structures or random processes built from (or on top) of a fixed graph G.

Some unifying questions:

- What can we learn about a graph by studying random structures defined on that graph?
- How effectively can we study random structures on infinite graphs via large finite approximations? Does the randomness 'feel the boundary'?
- How efficiently can one *sample* a particular random object on a large finite graph? How does the shape of the graph determine the speed of random processes on the graph?

1 Spanning trees and weak limits

1.1 Background: graphs, trees, USTs

Definition 1.1. A *graph* G consists of a set of vertices $V = V(G)$, and a set of edges $E = E(G) \subseteq \binom{V(G)}{2}$, the set of pairs of distinct vertices.

(In particular, in this course, graphs will be *simple*, so include no *self-loops* nor *multiple edges*.)

- A *rooted graph* (G, ρ) is a graph G, together with a choice of *root* $\rho \in V(G)$.
- Graph H is a *subgraph* of graph G when $V(H) = V(G)$, and $E(H) \subseteq E(G)$.
- Graph H is an *induced subgraph* of graph G when $V(H) \subseteq V(G)$, and $E(H)$ consists of precisely those edges in $E(G)$ which are incident to two vertices in $V(H)$.
- Graph G is *connected* if any pair of distinct vertices $x, y \in V(G)$ may be joined by a path made up of edges from $E(G)$. Otherwise, G has at least two *connected components*.
- The degree $\deg_G(v)$ of a vertex $v \in V(G)$ is the number of edges incident to v. A graph G is *locally-finite* if all the degrees are finite.

Example. Useful graphs which will appear frequently, especially as subgraphs or induced subgraphs, include:

- The complete graph K_n on n vertices, where every edge is present. Also called a *clique*.
- The empty graph E_n on n vertices, with no edges. (This is the *complement* of K_n.)
- The path P_n on n vertices (or of length $n-1$).
- The cycle C_n on n vertices.

We will often be keen to study the *d-dimensional integer lattice*, that is, the graph $\mathbb{Z}^d$ with nearest-neighbour edges $\big\{\{x, y\} : ||x - y||_1 = 1\big\}$. We will often study large finite approximations:

- The d-dimensional box $[0, n]^d$ or $[-n, n]^d$;
- The d-dimensional *torus* $[0, n]^d$, where vertices v, v' are identified when $v_i = v'_i$ for all $i \in [d]$ except one, for which $v_i = 0$ and $v'_i = n$.

- Consider an infinite graph G with vertex set V, and a sequence $V_1 \subset V_2 \subset \cdots \subset V$ where each V_n is finite, and such that $\cup_n V_n = V$. A recurring feature is to study G via (G_n), the sequence of graphs induced on each V_n by G. This is called an *exhaustion* of G by finite subgraphs.

Definition 1.2. A graph T is a *tree* if it is connected, and contains no cycles.

- For a general graph G, a *spanning tree* of G is a subgraph T which is a tree. (Note, T has the same vertex set as G. Also, G has no spanning trees if it is not connected.)

Proposition 1.3. A tree T with n vertices has exactly $n-1$ edges.

Proof. See the discussion of the first problem sheet. □

Definition 1.4. Let $\mathcal{T}$ be the set of all spanning trees of a finite connected graph G. Then, we call T a *uniform spanning tree of* G (UST) if it is distributed according to the uniform measure on $\mathcal{T}$. That is, for each $\mathfrak{t} \in \mathcal{T}$, we have $\mathbb{P}(T = \mathfrak{t}) = \frac{1}{|\mathcal{T}|}$. (*)

THEOREM 1.5 (Cayley's formula)**.** There are exactly n^{n-2} labelled trees with vertex set $[n] := \{1, 2, \ldots, n\}$.

Key questions:

- Except for very unusual (in particular, very *sparse*) graphs, the number of spanning trees will be large. For K_n, Cayley's formula gives the number of spanning trees. However, in almost all cases, it will be infeasible to compute precisely. So, if we can't use (*), how can we simulate a UST?
- Intuitively, the presence of two edges e, e' in a realisation of the UST should be *negatively dependent*. How does one make this precise? Is it always true?
- How can we simulate a UST as efficiently as possible?
- How can one extend this framework to infinite graphs, where $|\mathcal{T}|$ will normally be infinite, and so (*) won't work as a definition?

1.2 Generating uniform spanning trees

We will describe two algorithms for generating a UST on a (for now, finite) graph G.

- The *Aldous–Broder algorithm* (1990) based on a random walk on G. We will not give a proof of validity in this course.
- *Wilson's algorithm* (1996), based on a series of *loop-erased random walks* on G. We will give a proof of validity, as a first example of a *coupling* argument.

The Aldous–Broder algorithm

(Introduced roughly simultaneously by David Aldous and Andrei Broder, 1990.)

Our first algorithm is built from the notion of random walk on a graph, which we now define.

Definition 1.6. The (simple) random walk $(X_n)_{n\geq 0}$ on a locally-finite graph G is the Markov chain with state space $V(G)$ and transition probabilities $p_{vw} = \frac{1}{\deg_G(v)}$ when $vw \in E(G)$, and $p_{vw} = 0$ otherwise. In words, the chain moves to its next state by choosing uniformly from amongst the neighbours of its current state, independently of its history.

We will frequently study the *hitting time* τ_v of a vertex $v \in V(G)$,

$$\tau_v := \inf\{n \geq 0,\ X_n = v\},$$

which is a *stopping time*, and also the *cover time*,

$$t_{\text{cov}} := \max_{v\in V(G)} \tau_v = \inf\Big\{n \geq 0 : \{X_0, X_1, \ldots, X_n\} = V(G)\Big\},$$

which records at which step (if any) the walk has first visited *every* vertex.

When G is connected and recurrent (including when G is finite), all the hitting times τ_v and the cover time t_{cov} are finite almost surely. ('Almost surely' means 'with probability equal to one'.) Note that the distributions of the hitting times and the cover time will, in general, depend on the initial state X_0 of the walk, which may be random or deterministic.

The Aldous–Broder algorithm considers all edges along which the random walk on a graph G first visits a previously-unvisited vertex.

Definition 1.7 (Aldous–Broder algorithm)**.** For a connected, recurrent graph G, run a random walk on G from any initial state $X_0 \in V(G)$ until the cover time. Let T be the subgraph of G with edge set

$$E(T) := \Big\{ \{X_{\tau_v - 1}, X_{\tau_v}\} : v \in V(G)\Big\}. \tag{1.1}$$

THEOREM 1.8 (Aldous, Broder, 1990)**.** The random subgraph T is a UST on G.

Proof. One must start by showing that T is a spanning tree on G whenever the cover time is finite. This is an exercise on the problem sheet.

The proof that T is uniformly distributed amongst spanning trees is best seen as a consequence of the elegant and surprising *Matrix tree theorem*, which relates the stationary distribution of a (general / weighted) Markov chain on a graph G to the number (/ weight) of spanning trees involving each edge in the graph.

This proof is omitted and, obviously, **non-examinable**. □

Wilson's algorithm

As we've seen, the Aldous–Broder algorithm is constructed from a random walk on G. Obviously, we could not directly use the path taken by the random walk, because this would contain cycles. The next algorithm is based on the *loop-erased random walk* on G, which is constructed by removing the cycles from the trajectory of a random walk in the order in which they appear.

As we shall discuss later, there are advantages and disadvantages with both algorithms.

Definition 1.9. Let $\gamma = (x_0, x_1, \ldots, x_n) \subseteq V(G)$ be a finite, possibly self-intersecting path in a graph G. For every $v \in \{x_0, x_1, \ldots, x_n\}$, define $\ell_v := \max\{m \in [0, n] : x_m = v\}$, the index of the *last* visit to v.

Now, set $v_0 = x_0$, and then inductively define $v_{i+1} = x_{\ell_{v_i}+1}$ for $i \geq 1$, terminating at $v_{n'}$ for which $v_{n'} = x_n$ (and so $\ell_{v_i} = n$). The path $(v_0, v_1, \ldots, v_{n'})$ does not self-intersect, and is called the *loop-erasure* of γ.

Note that this definition applies equally to the loop-erasure of *infinite* paths $(x_0, x_1, x_2, \ldots)$, provided no vertex $v \in V(G)$ occurs infinitely many times in the path.

The loop-erasure of a finite path of random walk on G is called *loop-erased random walk* (LERW).

Definition 1.10 (Wilson's algorithm)**.** Given a connected, recurrent graph G, let $(v_0, v_1, \ldots, v_{n-1})$ be an enumeration of the vertices. We define a sequence of random trees $T_0 \subset T_1 \subset T_2 \subset \cdots$ as follows:

- Let T_0 be the tree with a single vertex $\{v_0\}$.
- When $i \geq 0$, and the vertex set of T_i is a strict subset of $V(G)$, define T_{i+1} as follows (independently from what has happened before):
  - Let j be the smallest label in $\{1, 2, \ldots, n-1\}$ such that $v_j \notin T_i$.
  - Let τ_{T_i} be the hitting time of T_i for a random walk $(X_0, X_1, \ldots)$ on G with initial vertex $X_0 = v_j$, and let $\gamma = (X_0, \ldots, X_{\tau_{T_i}})$.
  - Define the edge set of T_{i+1} to be the union of the edge set of T_i and the loop-erasure of γ. Note that T_{i+1} is also a tree.
- When T_i spans $V(G)$, declare $T = T_i$.

THEOREM 1.11 (Wilson, 1996)**.** Given a connected, recurrent graph G, the random spanning tree T given by Wilson's algorithm is uniformly distributed amongst spanning trees of G.

Proof of Theorem 1.11 (G finite)

We begin with a naive algorithm, which will be good motivation. Given a (finite) connected graph G with n vertices, and an identified vertex $v_0 \in V(G)$, by a *spanning tree directed towards* v_0 we mean collection of edges which form a spanning tree, where each edge is given the (unique) orientation towards v_0 in this tree.

Now, for every vertex v_i in $V(G)\backslash\{v_0\}$, select an edge $\overrightarrow{v_i w_i}$ directed away from v_i.

Claim: Such a collection of directed edges

- Either is a spanning tree directed towards v_0;
- Or includes a directed cycle.

The *proof of this claim* is an exercise on the problem sheet. This claim motivates the following.

Naive algorithm: For each vertex $v_i \in V(G)\backslash\{v_0\}$ independently, select an edge $\overrightarrow{v_i w_i}$ uniformly amongst those incident to v_i (*).

- If the resulting collection of directed edges is a spanning tree directed towards v_0, stop, and return the undirected version of this spanning tree.
- If the resulting collection of directed edges includes a directed cycle, start again.

Each spanning tree $\mathfrak{t}$ occurs with probability

$$\prod_{v_i \in V(G)\setminus\{v_0\}} \frac{1}{\deg_G(v_i)}$$

under (*). In particular, this is equal for all spanning trees $\mathfrak{t}$, and so the naive algorithm generates a UST on G.

Wilson's algorithm may be viewed sa adapting the naive algorithm above so that resampling the directed edges happens only at those vertices which are part of a problematic directed cycle.

Proof of Theorem 1.11. We will now prove that Wilson's algorithm generates a UST. We begin by defining a larger probability space, rich enough to support several realisations of LERW. (This is a good example of a *coupling*, which we will explore more fully in the coming lectures.)

We identify a vertex $v_0 \in V(G)$ as in Wilson's theorem. For every $v_i \in V(G)\setminus\{v_0\}$, let $(w_i^{(1)}, w_i^{(2)}, \ldots)$ be a sequence of choices from the neighbours of v_i in G. One may construct a walk $(X_0, X_1, \ldots)$ on G, *absorbed at* v_0, as a function of this information, as follows.

$$\text{Given } (X_0, X_1, \ldots, X_n = v) \text{ s.t. } k \text{ of } (X_0, \ldots, X_n) \text{ are vertex } v, \text{ set } \begin{cases} X_{n+1} = v_0 & X_n = v_0 \\ X_{n+1} = w_i^{(k)} & X_n = v_i. \end{cases}$$

Ie, after the kth visit to $v_i \neq v_0$, the random walk moves on the directed edge $\overrightarrow{v_i w_i^{(k)}}$ to vertex $w_i^{(k)}$.

Similarly, one may construct a walk on G absorbed at any collection of vertices, and also its loop-erasure, so long as the path is finite.

We think of the collection $(w_i^{(k)})$ as a *stack of instructions* left at vertex v_i, with $w_i^{(1)}$ as, initially, the top of the stack. In the following construction, we will iteratively discard the top element of some stacks, so the top-most (or *visible*) instruction on some stacks changes from $w_i^{(k)}$ to $w_i^{(k+1)}$.

The visible instructions always define a set of directed edges, just as in the claim and naive algorithm above. When this includes at least one directed cycle, we may remove one of these directed cycles, by discarding the visible instruction from every stack corresponding to a vertex in the cycle. The collection of visible instructions thus changes, and maybe further directed cycles may be removed in subsequent steps.

This process, referred to as *cycle popping*, may be iterated until the visible instructions induce no directed cycle, and we let N be the number of cycles 'popped' in the procedure. Either $N = \infty$, or $N < \infty$, and in the latter case (by the earlier claim) what remains is a spanning tree $\mathfrak{t}$ directed towards v_0.

Obviously, we have to choose some order to pop the cycles. It is not clear *a priori* whether $\mathfrak{t}$ and N will necessarily be independent of this choice of cycle-popping ordering. Fortunately, this is true!

Lemma 1.12 (Cycle-popping lemma)**.** For any choice of $(w_i^{(k)}, i = 1, \dots, n-1, k \geq 1)$, for any order of cycle-popping, N and $\mathfrak{t}$ are the same. That is:

- either $N = \infty$ for all orderings of cycle-popping;
- or $N < \infty$ is the same for all orderings of cycle-popping, and all such orderings give the same directed spanning tree $\mathfrak{t}$.

Proof. Note that this is a *deterministic* result, not a probabilistic result! A proof of this lemma will be given in the problems class. □

Now, we will consider the situation where, independently for each $v_i \neq v_0$, $(w_i^{(1)}, w_i^{(2)}, \dots)$ is a sequence of IID copies of W_i, an independent choice from amongst the neighbours of v_i in G. Note then, that we can use this (random) information to simulate Wilson's algorithm (where for LERW, cycle-popping happens in 'chronological' order), and finite termination of Wilson's algorithm at a tree $\mathfrak{t}$ exactly corresponds to cycle-popping with $N < \infty$ and $\mathfrak{t}$ directed towards $\{v_0\}$.

It is clear by construction (of Wilson's algorithm) that in this setting, N is almost surely finite. We will show that $\mathfrak{t}$ is a UST on G. Each cycle which is popped consists of a collection of directed edges of the form $\overrightarrow{v_i w_i^{(k)}}$, where k indexes its position in the stack. Let $\mathcal{C}_1, \dots, \mathcal{C}_M$ be some collection of these indexed directed cycles, which can be popped in this order (which is certainly independent of whatever is left underneath). Note that we view eg the 'cycles' $\{v_1 w_1^{(1)}, v_2 w_2^{(1)}\}$ and $\{v_1 w_1^{(2)}, v_2 w_2^{(3)}\}$ as distinct.

Then the probability that the random stacks of instructions $(w_i^{(k)})$ give these indexed, directed cycles, and leave the spanning tree $\mathfrak{t}$ directed towards $\{v_0\}$ as the visible instructions underneath the popped cycles, is precisely

$$\prod_{m=1}^{M} \prod_{v_i w_i^{(k)} \in \mathcal{C}_m} \frac{1}{\deg_G(v_i)} \times \prod_{v_i \in V(G) \setminus \{v_0\}} \frac{1}{\deg_G(v_i)}, \tag{1.2}$$

where the second term arises for exactly the same reasons as in the naive algorithm discussed earlier.

Note that (1.2) splits as a product of a term depending on $(\mathcal{C}_1, \dots, \mathcal{C}_M)$ and a term (in fact, a constant term) depending on $\mathfrak{t}$. So in fact we have the stronger result that *conditional* on any $(\mathcal{C}_1, \dots, \mathcal{C}_M)$, $\mathfrak{t}$ has the uniform distribution amongst spanning trees. The required statement follows immediately by 'averaging' (/ law of total $\mathbb{P}$) over the sequence of cycles. □

Remark. The proof shows that one may choose which vertex to start the 'next LERW' from in Wilson's algorithm with considerable flexibility.

Remark. This proof instantly generalises to the case of *weighted* spanning trees, generated by an algorithm involving the loop-erasure of *weighted* random walks.

1.3 Negative association and related topics

Couplings and stochastic domination

The construction used in this proof of the validity of Wilson's algorithm is an example of the *coupling method*. This is a crucial and recurring tool in discrete probability theory, where one compares two or more random structures by constructing them jointly in the same probability space.

Concretely, given random variables X_1 and X_2 defined on probability spaces $(\Omega_1, \mathcal{F}_1, \mathbb{P}_1)$ and $(\Omega_2, \mathcal{F}_2, \mathbb{P}_2)$, respectively, a *coupling* of X_1, X_2 is a pair of random variables (Y_1, Y_2) defined on some new probability space $(\Omega, \mathcal{F}, \mathbb{P})$ for which $Y_1 \stackrel{d}{=} X_1$ and $Y_2 \stackrel{d}{=} X_2$.

(This level of formality is rarely used in practice.)

Some more examples will be given in lectures, after which these notes will be updated.

A particularly useful situation is when there exists a coupling of random variables X, Y for which $\mathbb{P}(X \leq Y) = 1$. In this case, we say that Y *stochastically dominates* X, or $X \leq_{st} Y$.

When Y stochastically dominates X, two immediate consequences are:

- $\mathbb{E}[X] \leq \mathbb{E}[Y]$;
- for any $x \in \mathbb{R}$, we have $\mathbb{P}(X > x) < \mathbb{P}(Y > x)$.
- Note that these statements make sense for *any* coupling of X, Y.

As we shall see, the notion of stochastic domination extends to more general partial orderings, for example with respect to subgraphs of a fixed graph G.

Negative dependence

Since a spanning tree of a graph G must include a fixed number of edges, it is intuitively reasonable to assume that the presence of edges e, e' in a UST T will be negatively-correlated. Similarly, if we have fewer edges from which to assemble a spanning tree, then any given edge e is more likely to appear in a UST.

We make these statements precise in the following result.

Proposition 1.13. Given a finite, connected graph G, let T be a UST on G.

a) For any pair e, e' of distinct edges in $E(G)$,

$$\mathbb{P}\left(e' \in E(T) \,\middle|\, e \in T\right) \leq \mathbb{P}\left(e' \in E(T)\right). \tag{1.3}$$

More generally, if A and B are disjoint, acyclic collections of edges in $E(G)$, then

$$\mathbb{P}\left(B \subset E(T) \,\middle|\, A \subset E(T)\right) \leq \mathbb{P}\left(B \subset E(T)\right). \tag{1.4}$$

b) Let H be a connected subgraph of G, and let T_H be a UST on H. Then, for any acyclic collection A of edges in $E(H)$. Then

$$\mathbb{P}\left(A \subset E(T_H)\right) \geq \mathbb{P}\left(A \subset E(T)\right). \tag{1.5}$$

Aspects of these *negative dependence* results for the UST will be discussed in lectures. In summary:

- These results are best proved using the so-called *electrical network interpretation* of finite graphs, originating with Kirchhoff (1840s). We have omitted this topic from the 2019/20 edition of this course.
- *Negative* dependence results are often very hard, and several intuitively 'obvious' results are in fact famous open problems.
- The theory of *positive dependence* is much better understood, and will be introduced properly in the next section of this course.