Local limit of Prim's algorithm

Benoît Corsini

join work with R. Gündlach and R. van der Hofstad

Table of Content

Table of Content

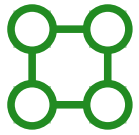
Local limit

Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.

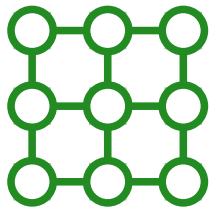
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



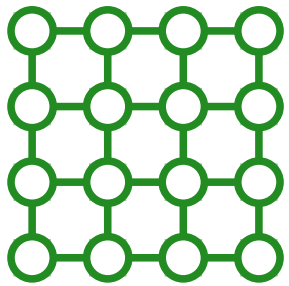
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



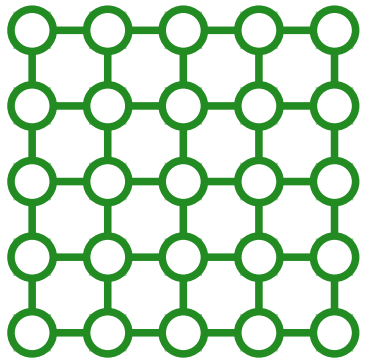
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



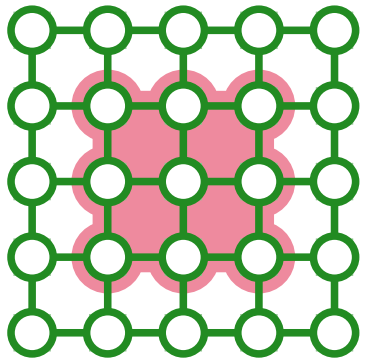
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



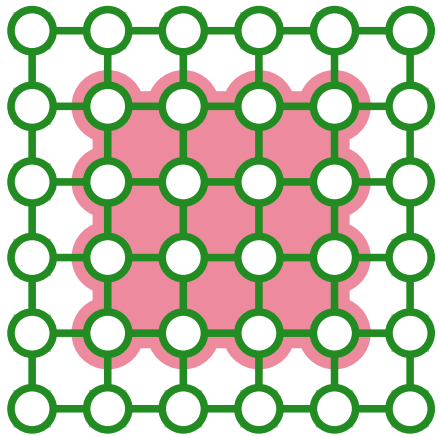
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



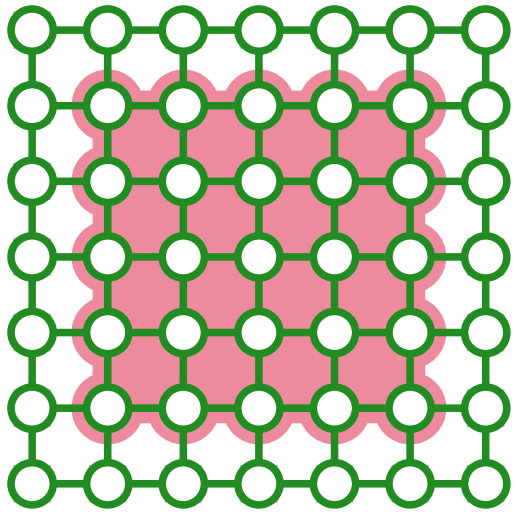
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



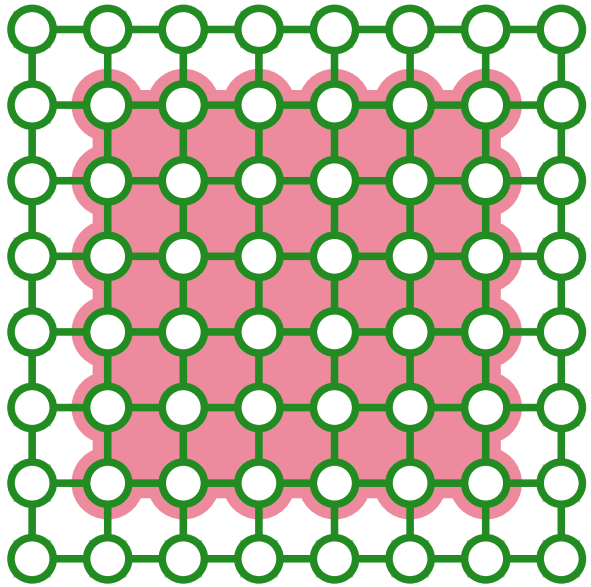
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



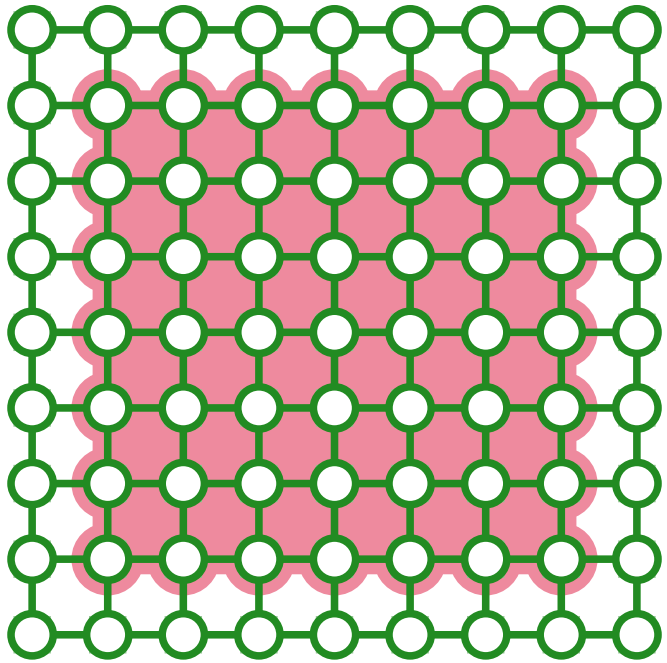
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



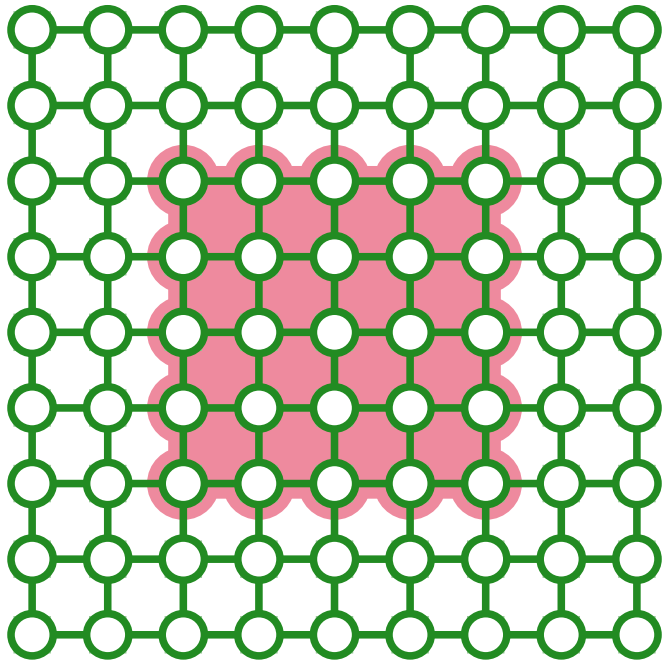
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



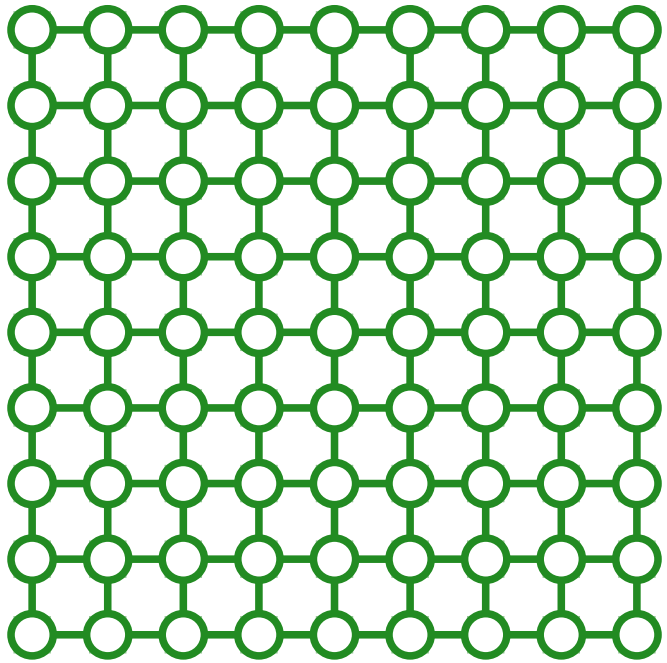
Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



Local limit

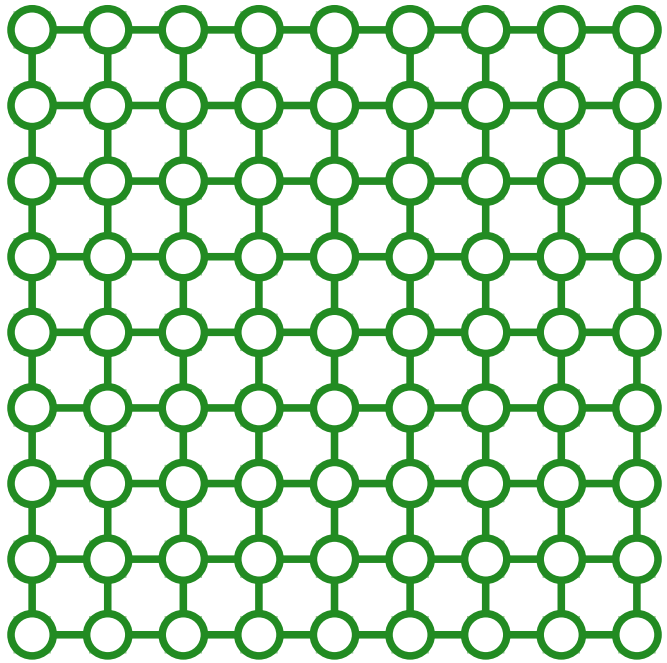
Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.



Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.

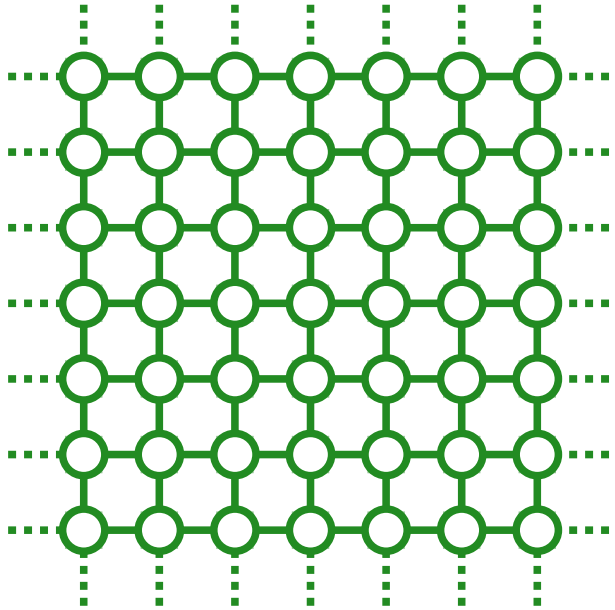
→ The sequence of tori $((\mathbb{Z}/n\mathbb{Z})^d)_{n\geq 1}$ converges towards $\mathbb{Z}^d$.



Local limit

Given a sequence of graphs $(G_n)_{n\geq 1}$, their local limit is the structure as seen from a node chosen uniformly at random.

→ The sequence of tori $((\mathbb{Z}/n\mathbb{Z})^d)_{n\geq 1}$ converges towards $\mathbb{Z}^d$.



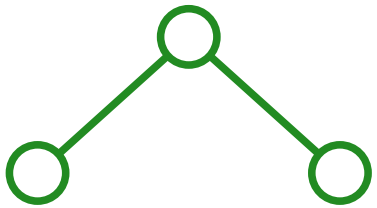
Canopy tree

Canopy tree

Q: What is the local limit of growing binary trees?

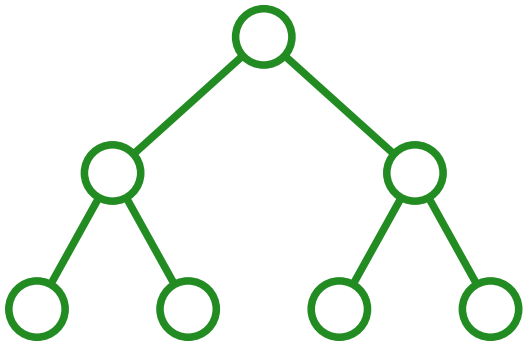
Canopy tree

Q: What is the local limit of growing binary trees?



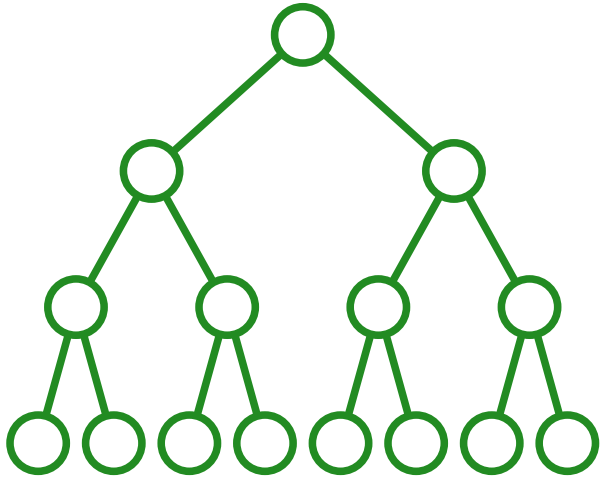
Canopy tree

Q: What is the local limit of growing binary trees?



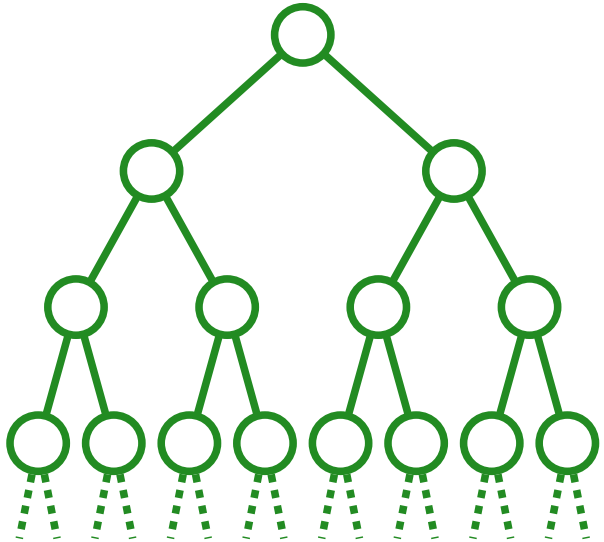
Canopy tree

Q: What is the local limit of growing binary trees?



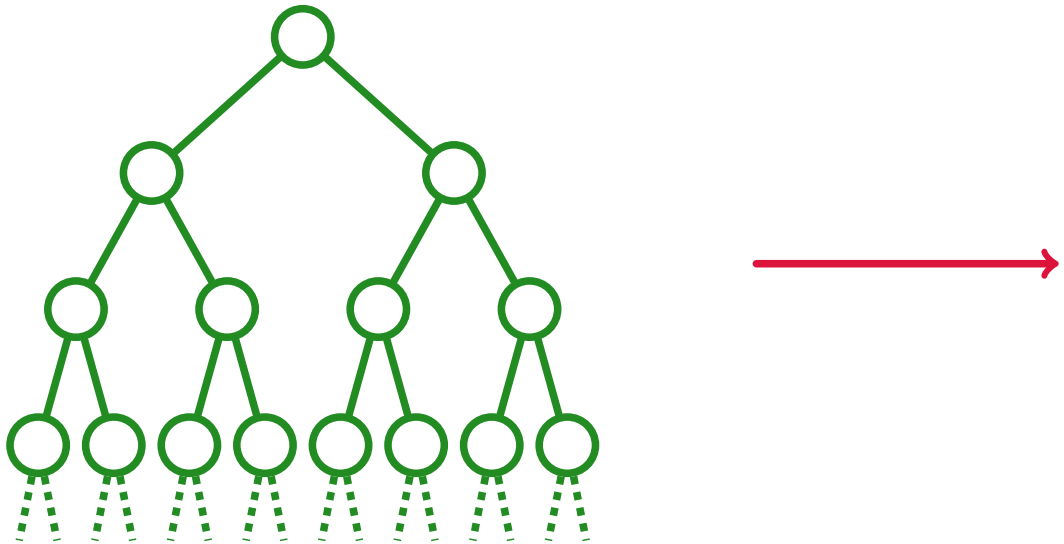
Canopy tree

Q: What is the local limit of growing binary trees?



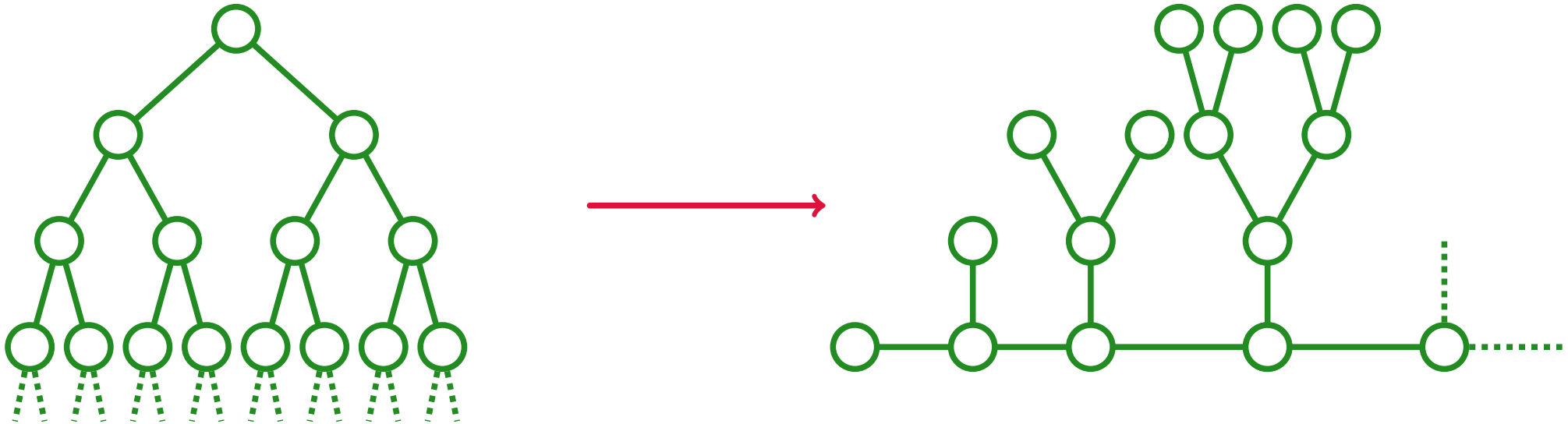
Canopy tree

Q: What is the local limit of growing binary trees?



Canopy tree

Q: What is the local limit of growing binary trees?



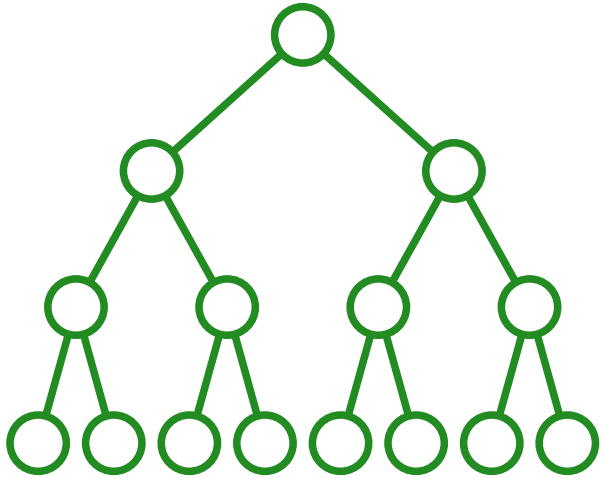
Regular tree

Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?

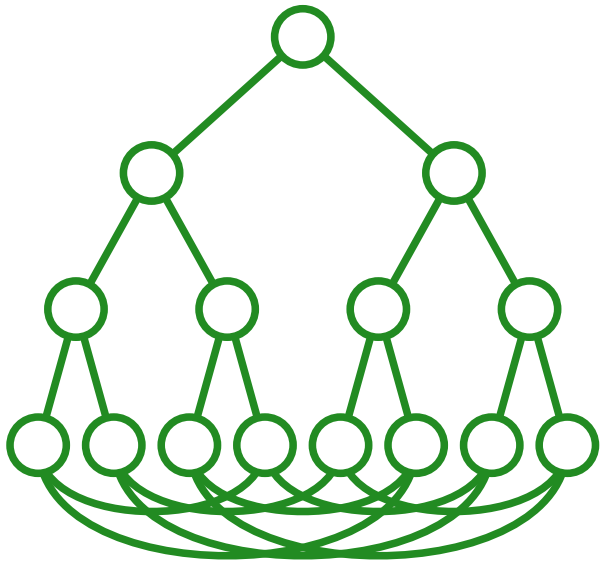
Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



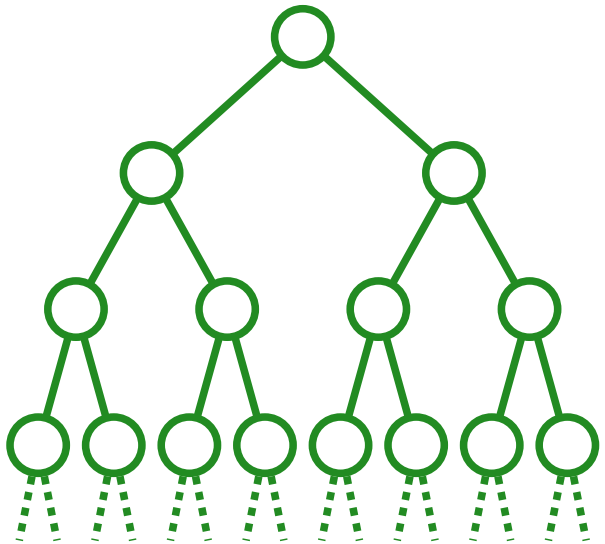
Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



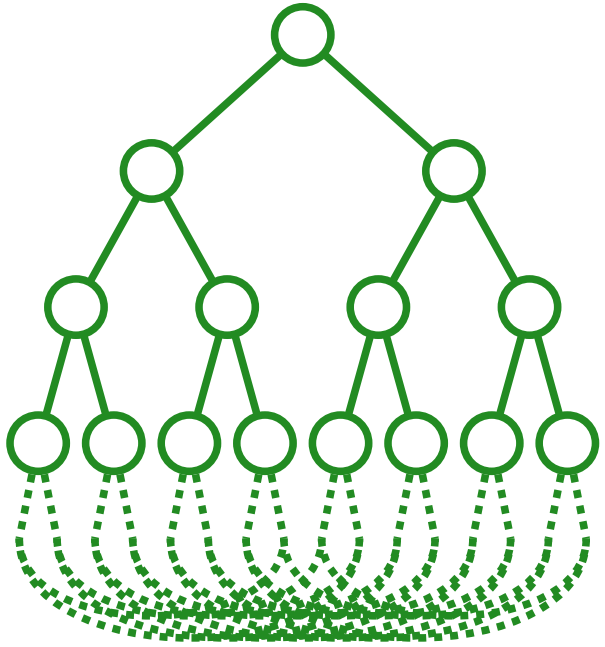
Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



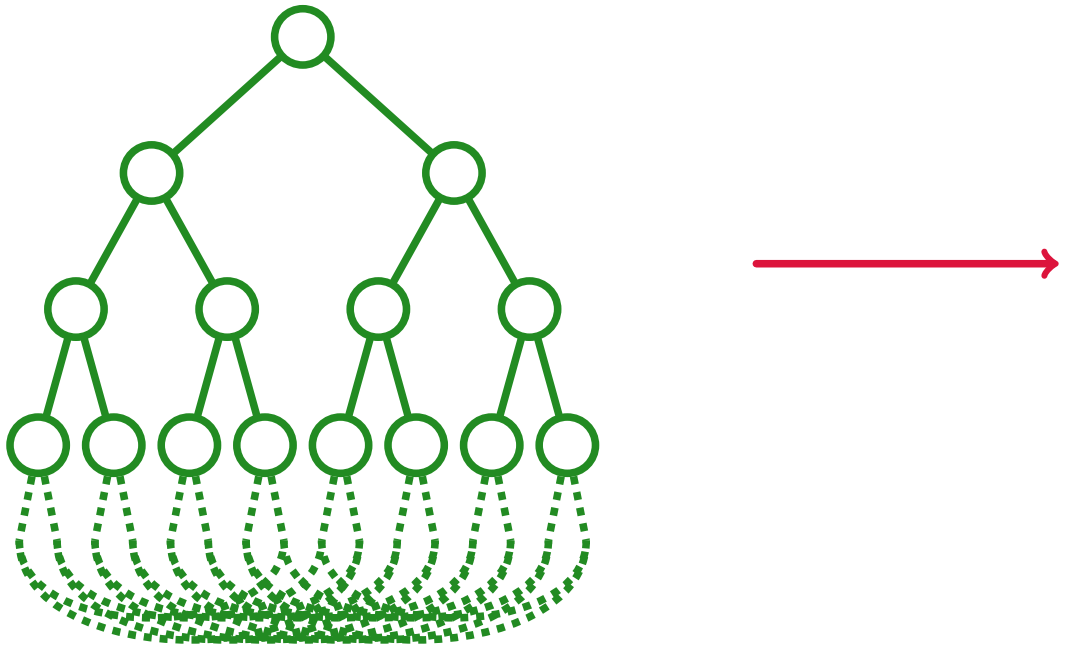
Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



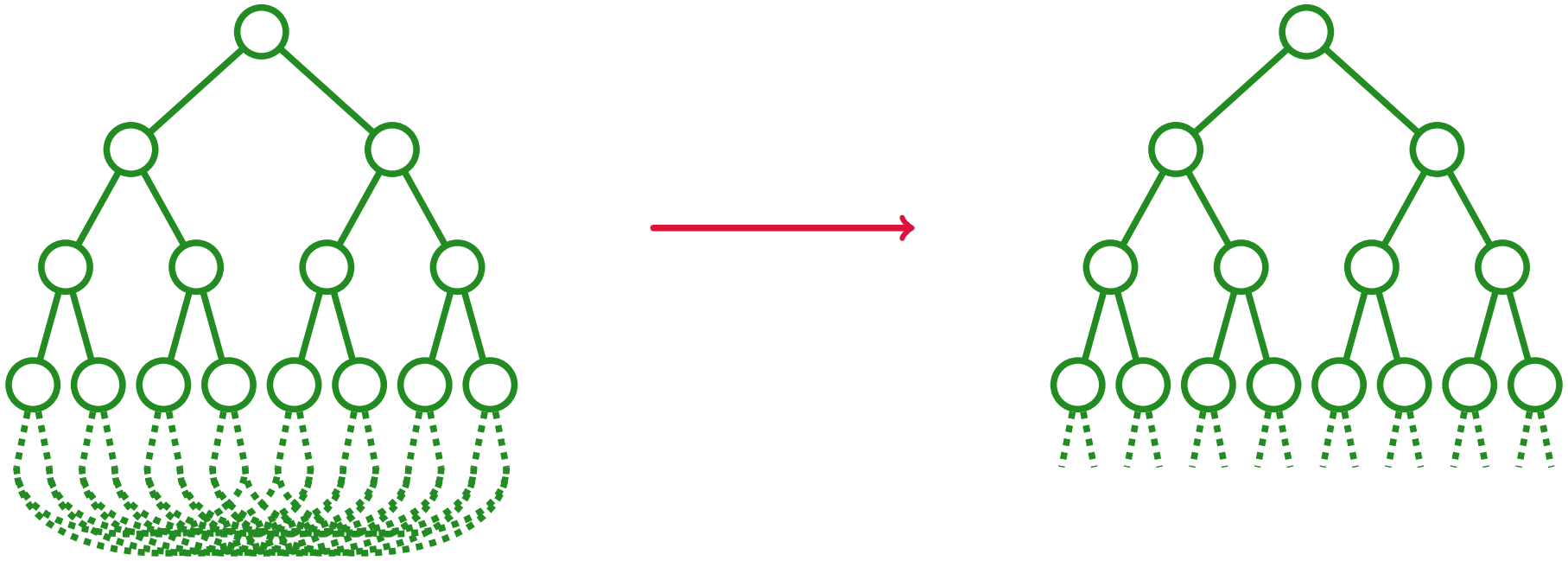
Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



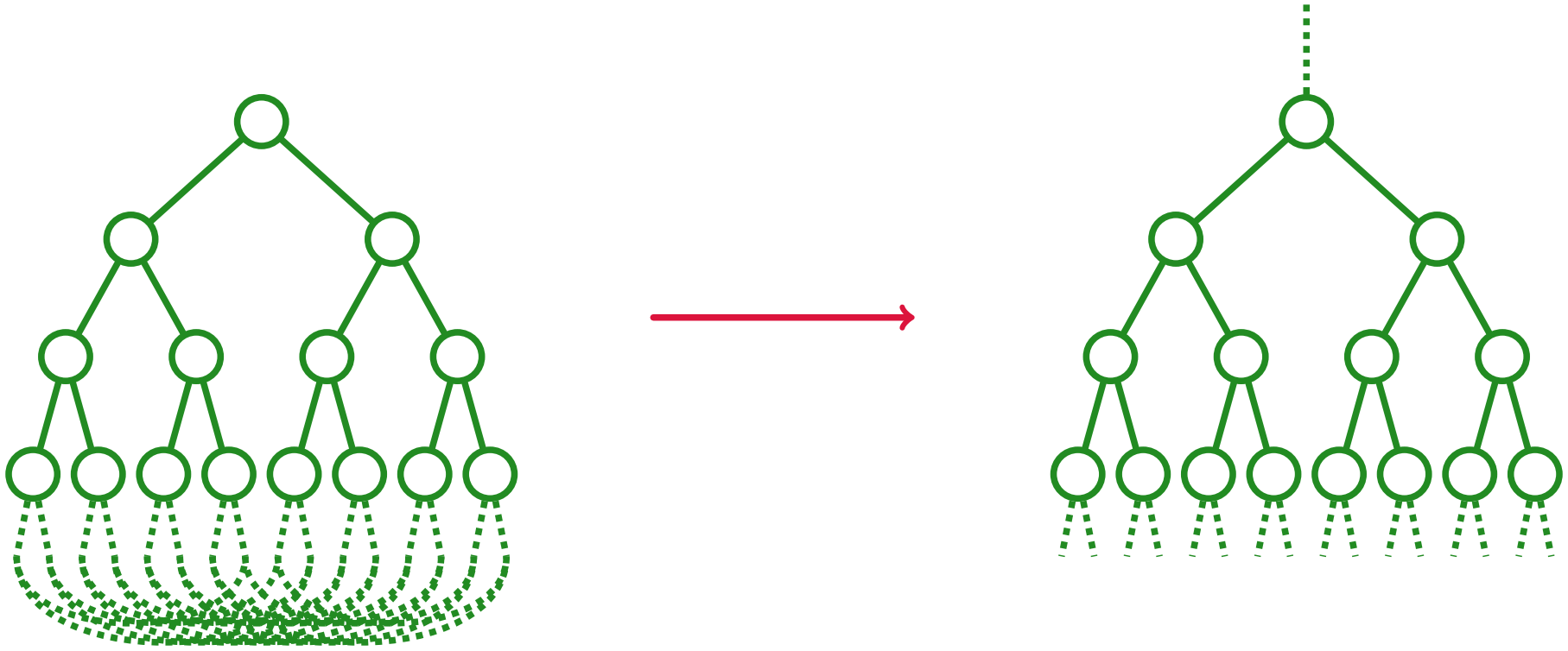
Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?



Regular tree

Q: Is there a sequence of trees whose limit is the infinite binary tree?

→ **Almost:** d-regular graphs with diverging girth converge to the infinite d-ary tree.

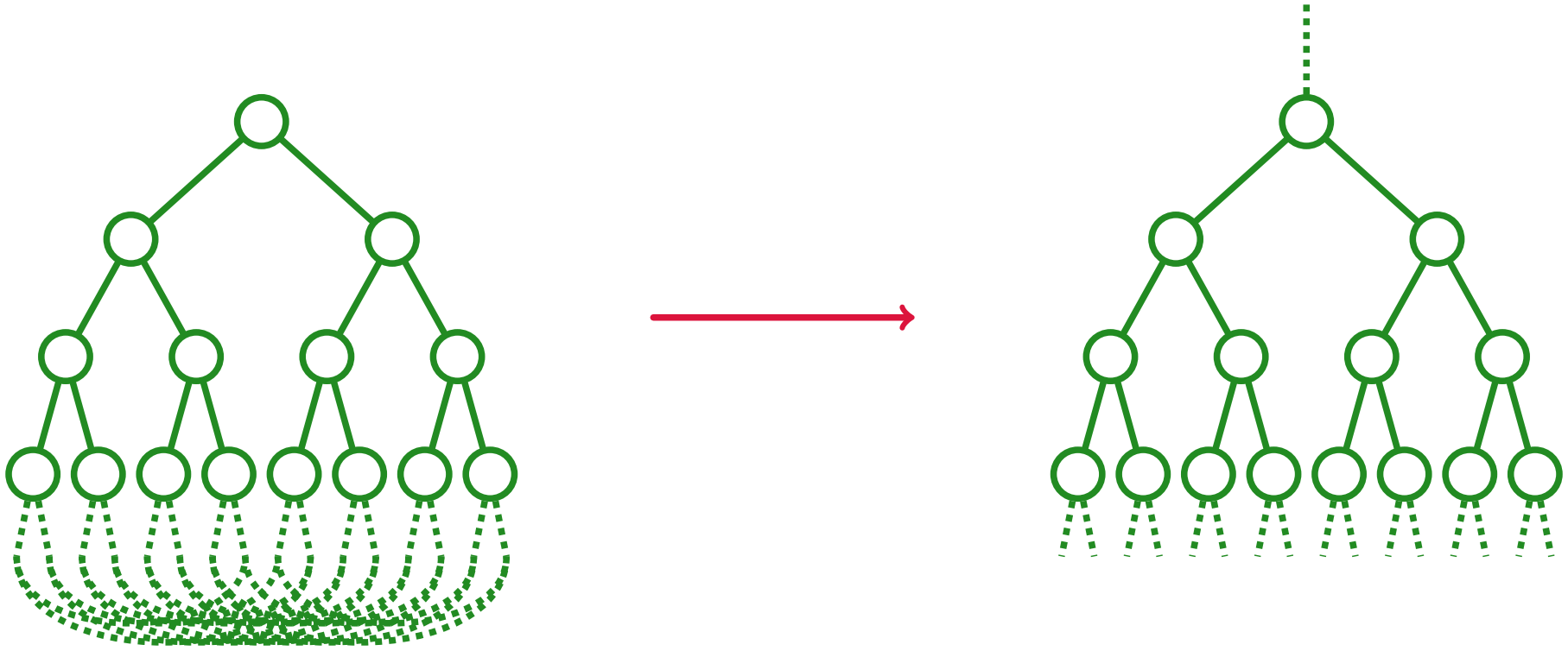


Table of Content

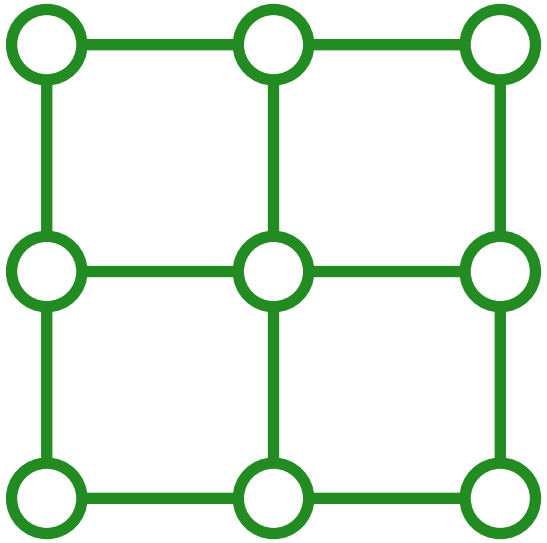
Minimum spanning tree

Minimum spanning tree

Given a weighted graph, the minimum spanning tree is the unique tree connecting all nodes which also minimizes the total edge weights.

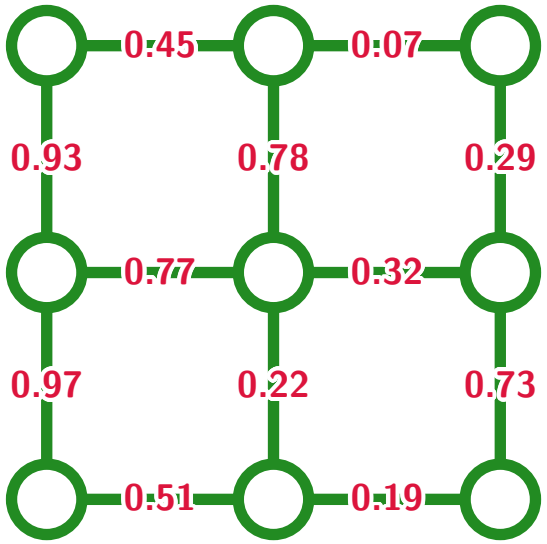
Minimum spanning tree

Given a weighted graph, the minimum spanning tree is the unique tree connecting all nodes which also minimizes the total edge weights.



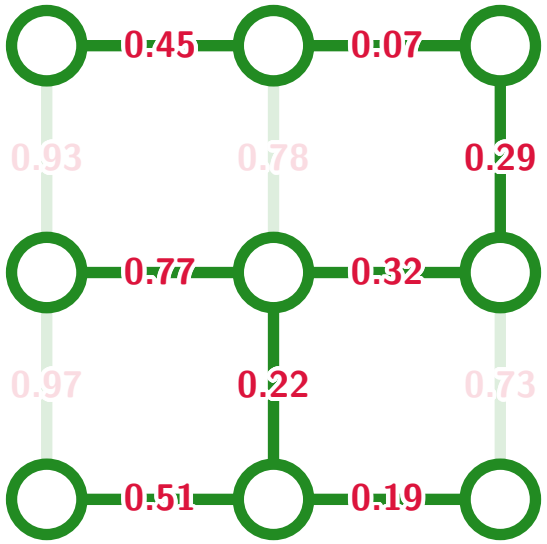
Minimum spanning tree

Given a weighted graph, the minimum spanning tree is the unique tree connecting all nodes which also minimizes the total edge weights.



Minimum spanning tree

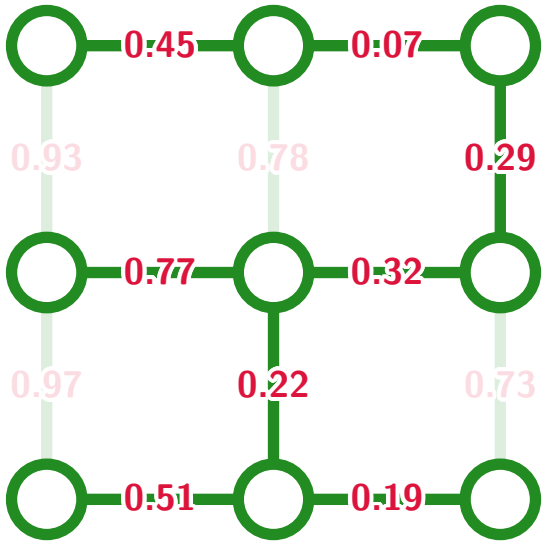
Given a weighted graph, the minimum spanning tree is the unique tree connecting all nodes which also minimizes the total edge weights.



Minimum spanning tree

Given a weighted graph, the minimum spanning tree is the unique tree connecting all nodes which also minimizes the total edge weights.

→ Can we find an "easy" way to compute the minimum spanning tree?



Prim's algorithm

Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.

Prim's algorithm

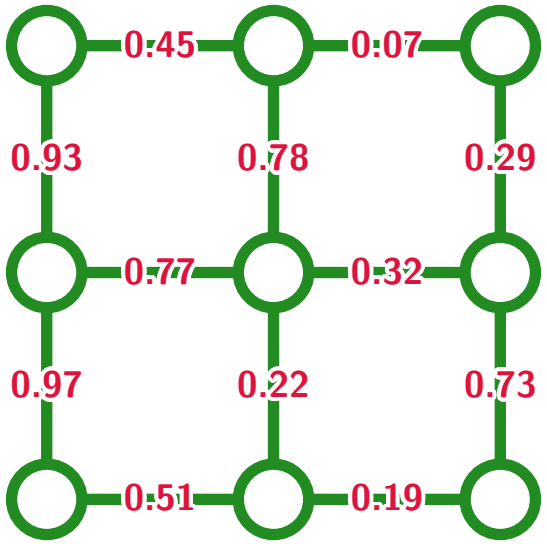
Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.

Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.

Prim's algorithm

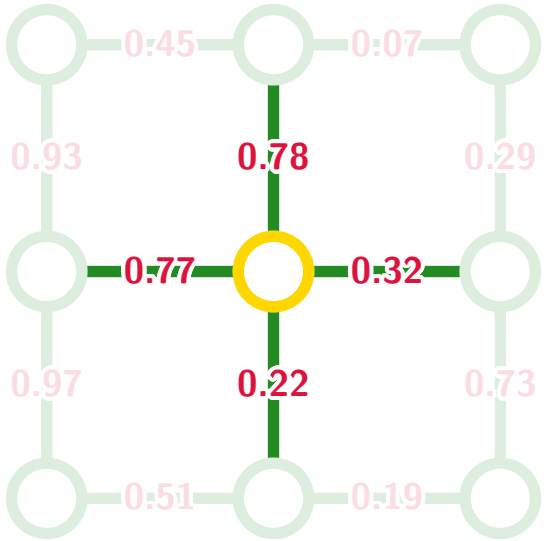
Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.

Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

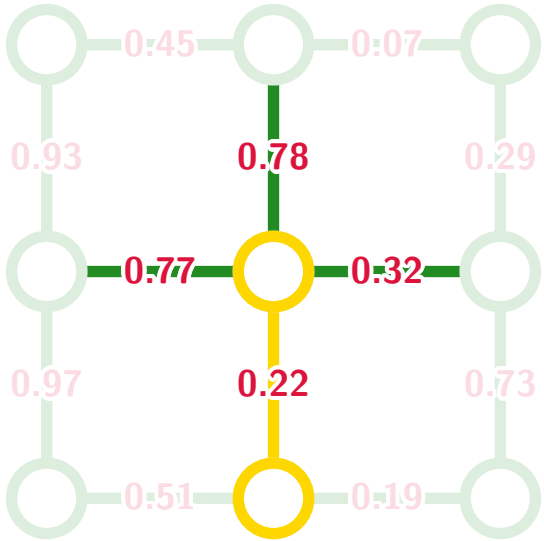
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

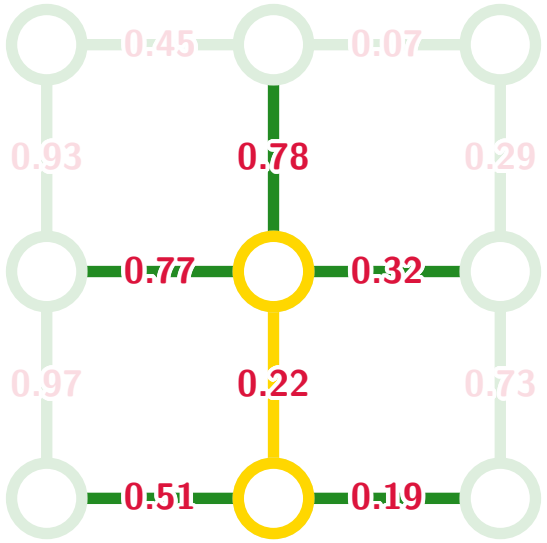
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

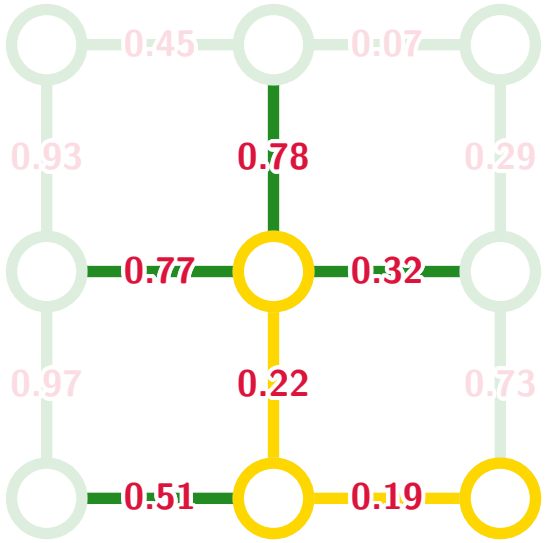
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

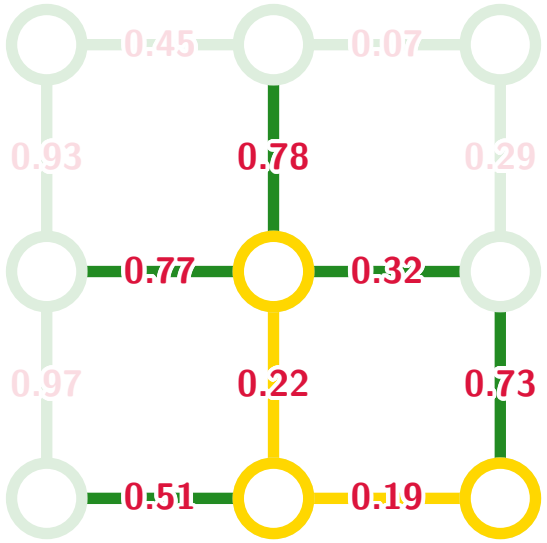
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

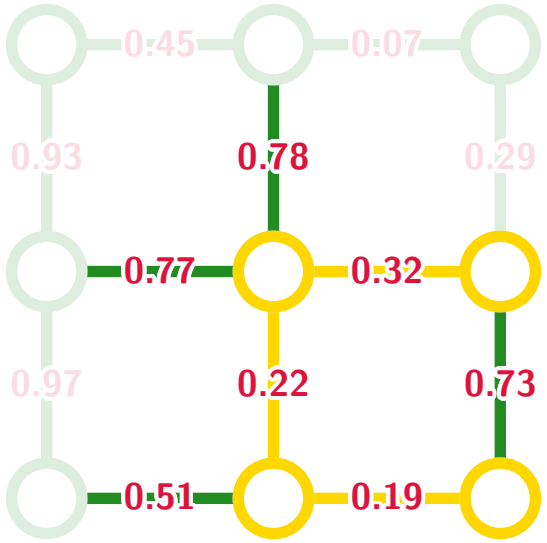
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

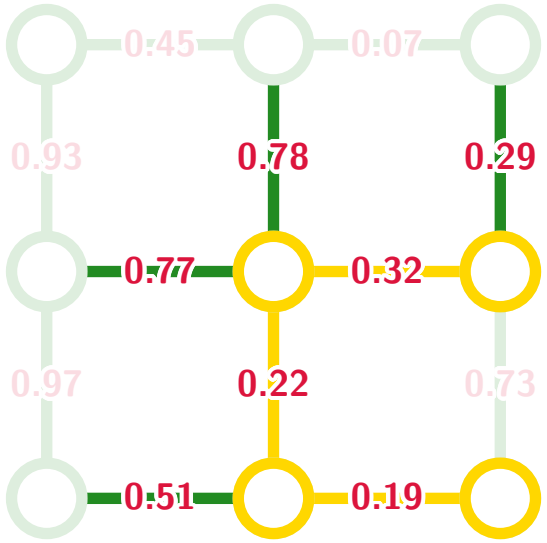
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

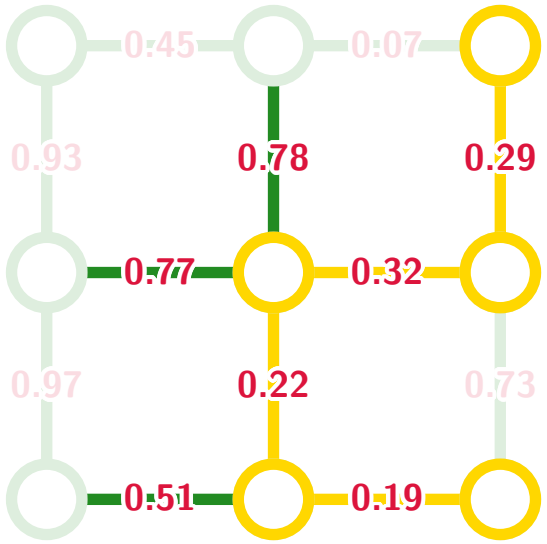
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

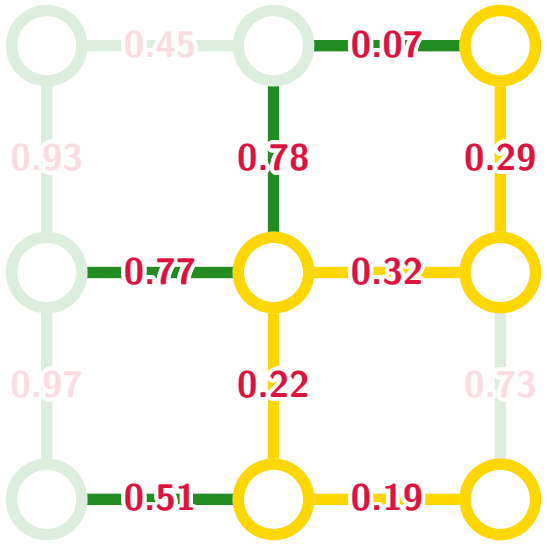
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

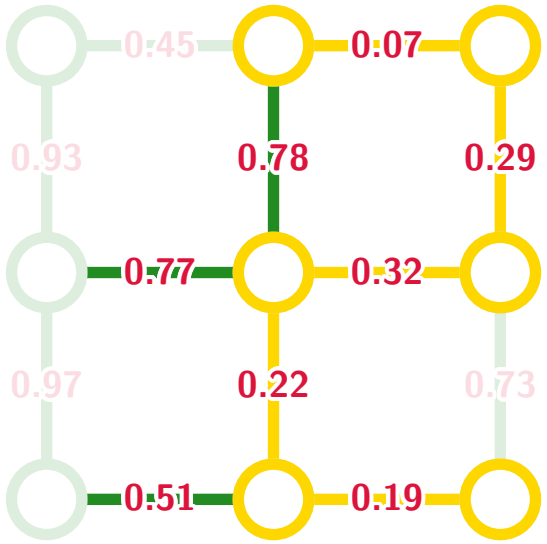
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

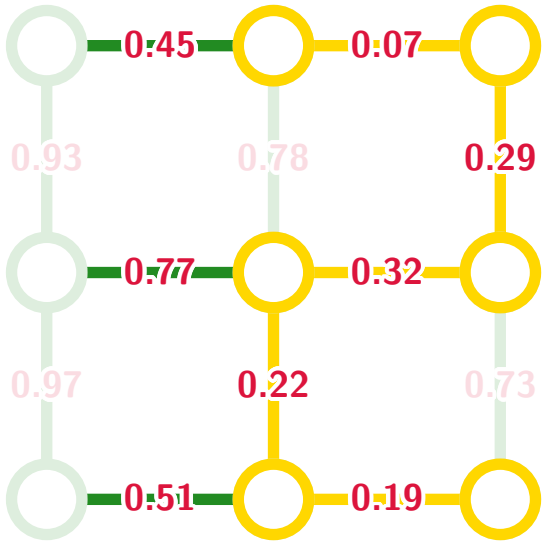
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

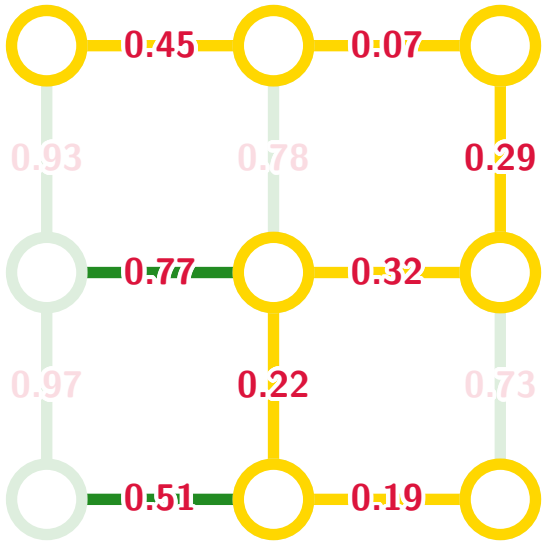
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

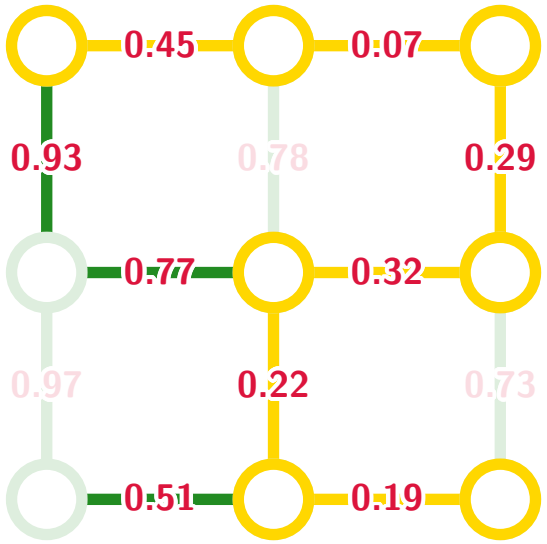
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

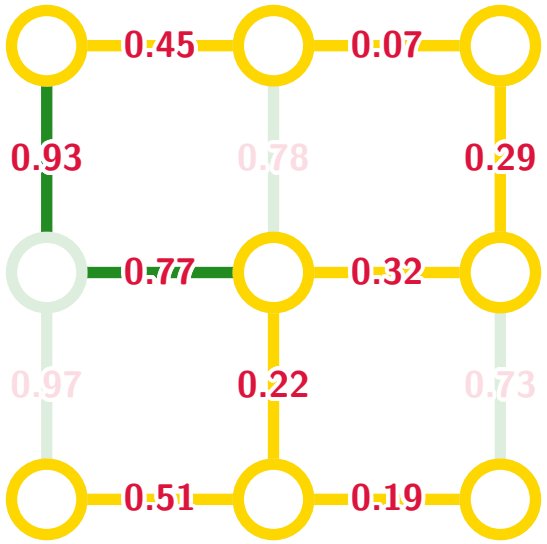
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

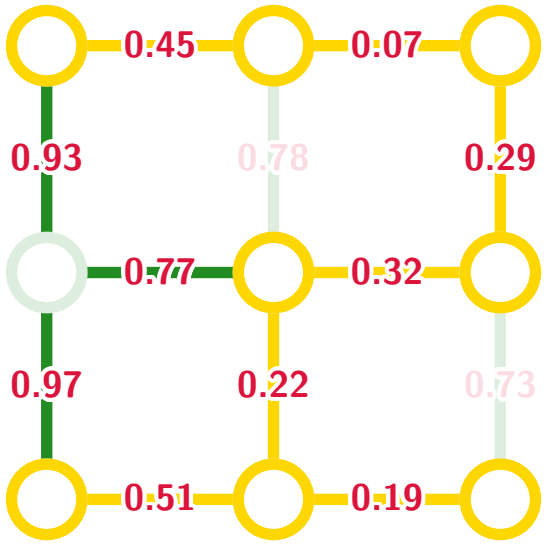
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

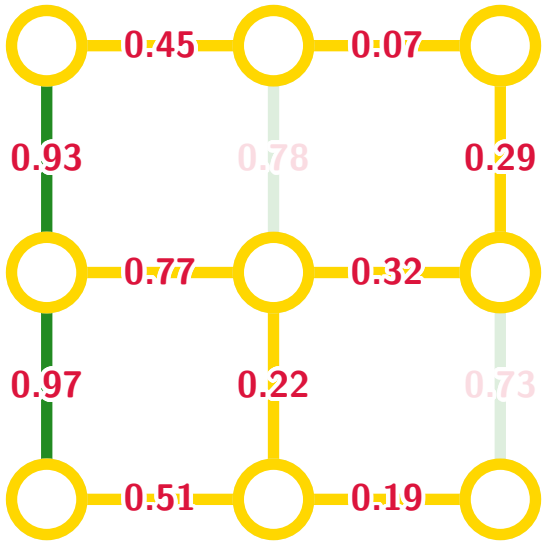
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

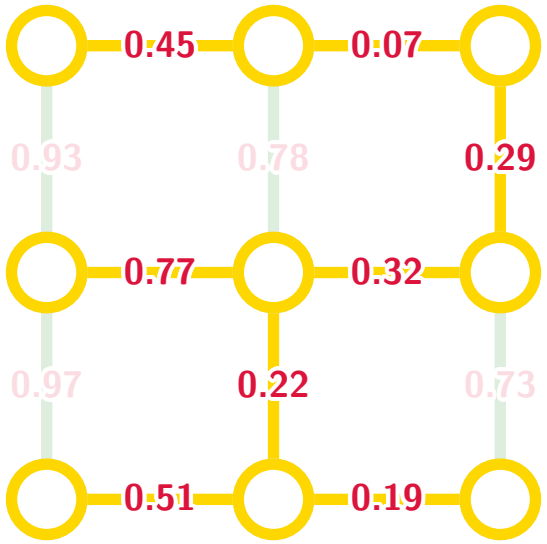
- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.



Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.

→ This algorithm works because of the following key proposition.

Prim's algorithm

Prim's algorithm is an inductive rooted methods to construct the minimum spanning tree.

- Start with a tree composed of a single node, arbitrarily chosen from the graph.
- Given the current tree, grow it by adding the edge on its boundary of smallest weight.

→ This algorithm works because of the following key proposition.

Proposition

The edges of the minimum spanning tree are exactly those that are never the heaviest on any cycle.

Prim's algorithm

Prim's algorithm

Properties of Prim's algorithms:

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree is the entire line.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree is the entire line.

→ Prim's algorithm also explores the entire line.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree is the entire line.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree is the entire line.

→ Prim's algorithm only explores the right portion of the line.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

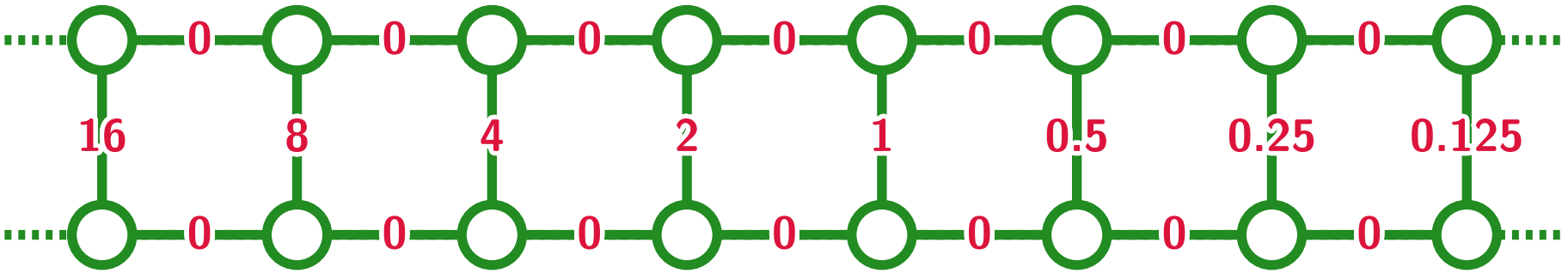
Q: What happens when G is infinite?

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

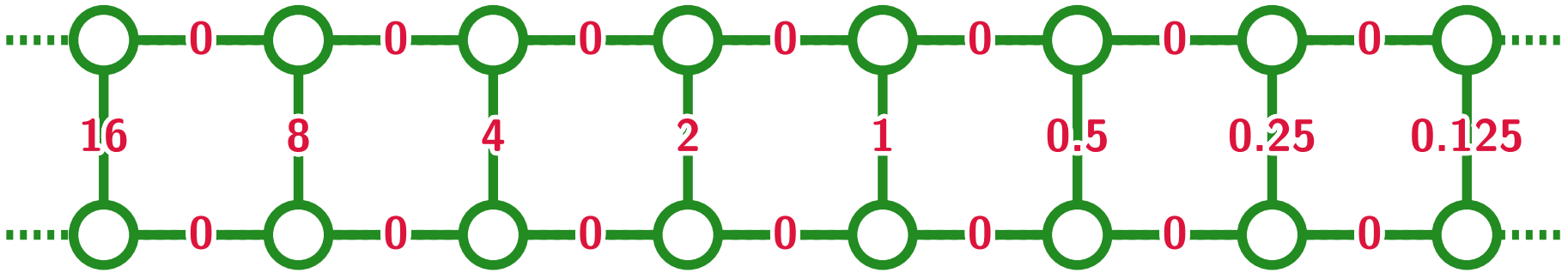


Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?



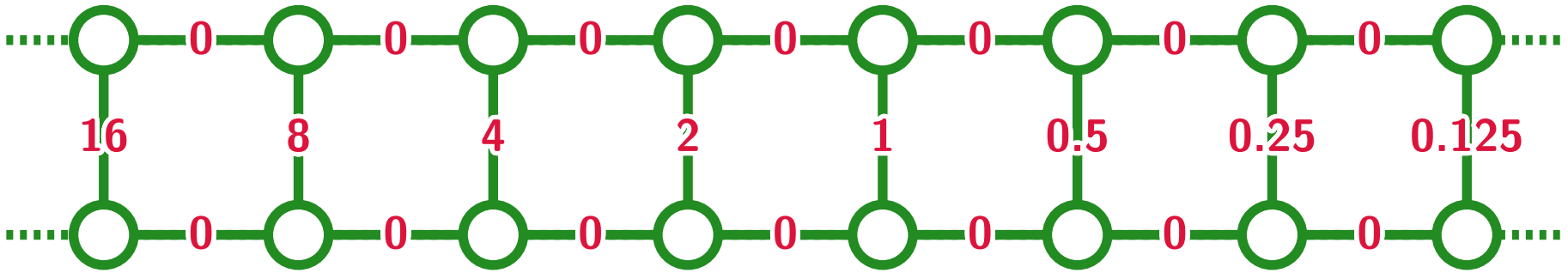
→ The minimum spanning tree is not well-defined (or is a forest).

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?



→ The minimum spanning tree is not well-defined (or is a forest).

→ Prim's algorithm only explores the line on which it starts.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree does not always exist, but the *minimum spanning forest* does.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree does not always exist, but the *minimum spanning forest* does.

→ Prim's algorithm is still well-defined, even with an infinite number of steps.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree does not always exist, but the *minimum spanning forest* does.

→ Prim's algorithm is still well-defined, even with an infinite number of steps.

→ The output of Prim's algorithm is a subtree of the minimum spanning forest.

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree does not always exist, but the *minimum spanning forest* does.

→ Prim's algorithm is still well-defined, even with an infinite number of steps.

→ The output of Prim's algorithm is a subtree of the minimum spanning forest.

- This subtree is not necessarily a tree of the minimum spanning forest. 

Prim's algorithm

Properties of Prim's algorithms:

- It constructs a sequence of trees starting from any given root.
- If the graph has n vertices, the n-th tree of the algorithm is the minimum spanning tree.

Q: What happens when G is infinite?

→ The minimum spanning tree does not always exist, but the *minimum spanning forest* does.

→ Prim's algorithm is still well-defined, even with an infinite number of steps.

→ The output of Prim's algorithm is a subtree of the minimum spanning forest.

- This subtree is not necessarily a tree of the minimum spanning forest.
- It is often referred to as the *invasion percolation cluster*.

Table of Content

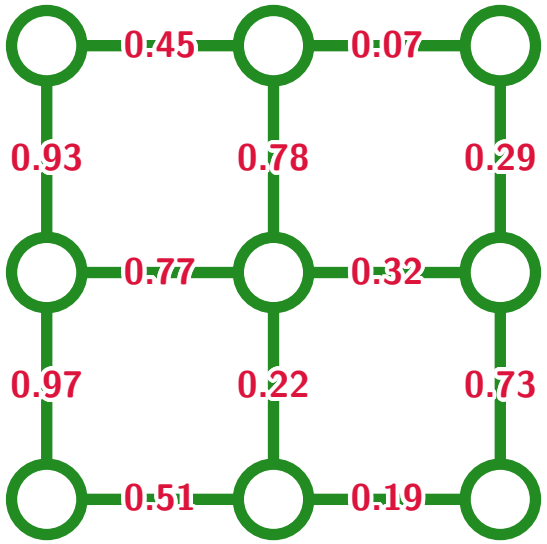
Percolation levels

Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Percolation levels

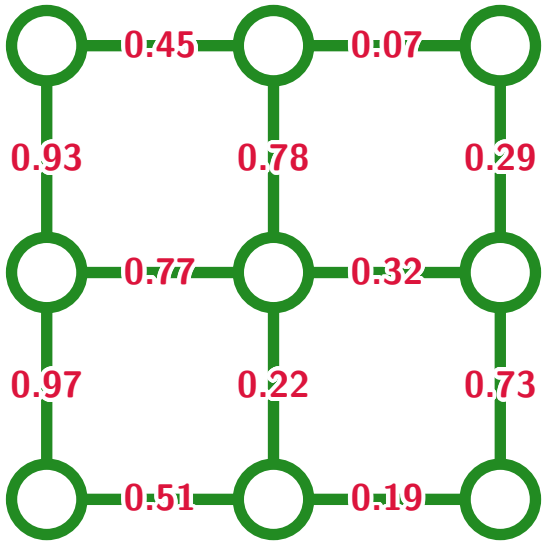
Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

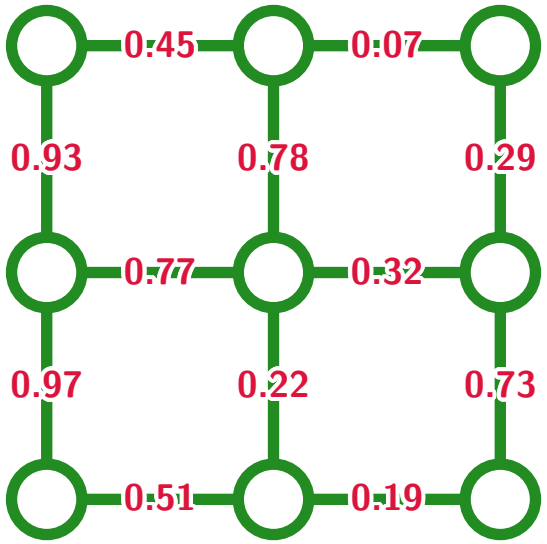
- $p = \dots$



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

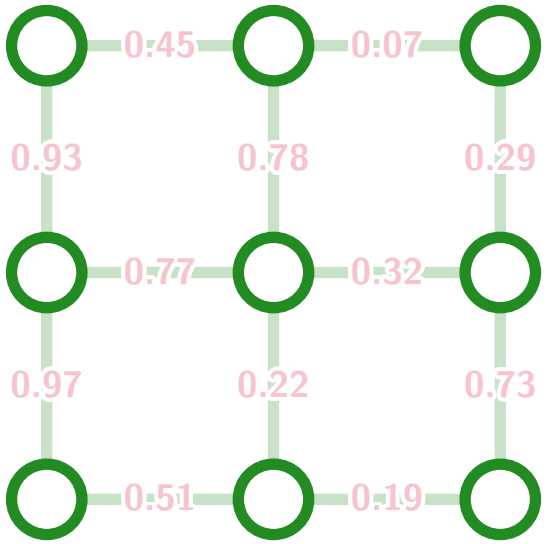
- $p \in [0, 1]$



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

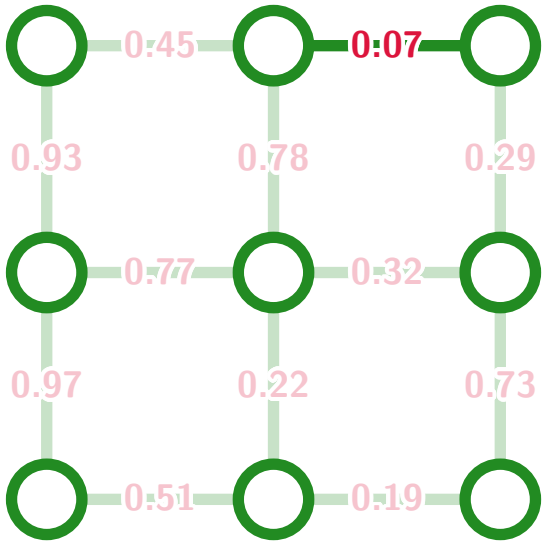
- $p = 0.00$



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

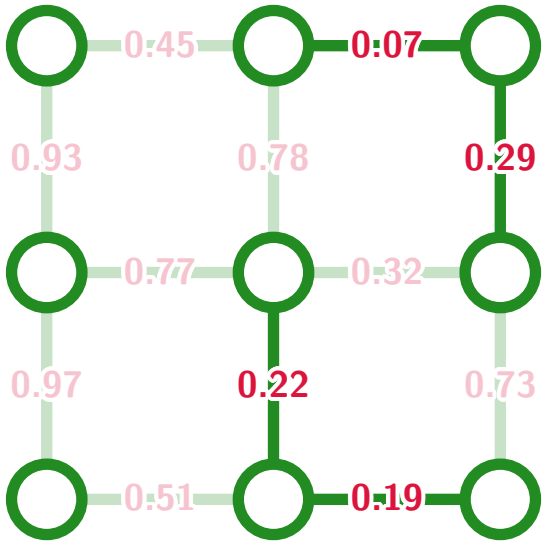
- $p = 0.07$



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

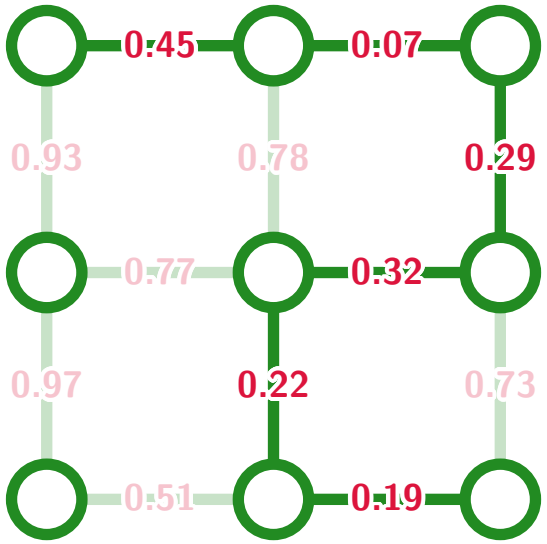
- $p = 0.30$



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

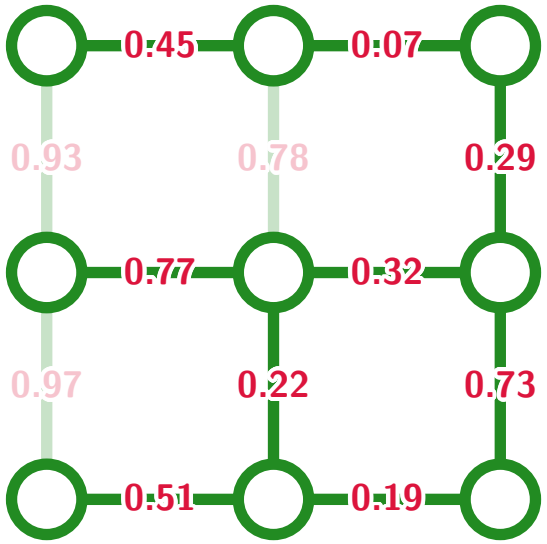
- $p = 0.50$



Percolation levels

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

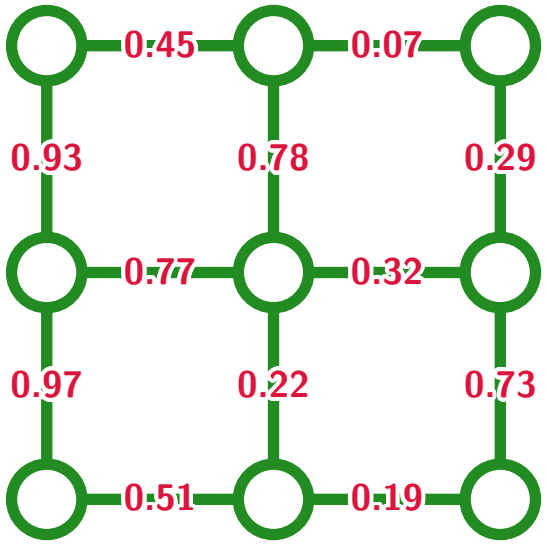
- $p = 0.77$



Percolation levels

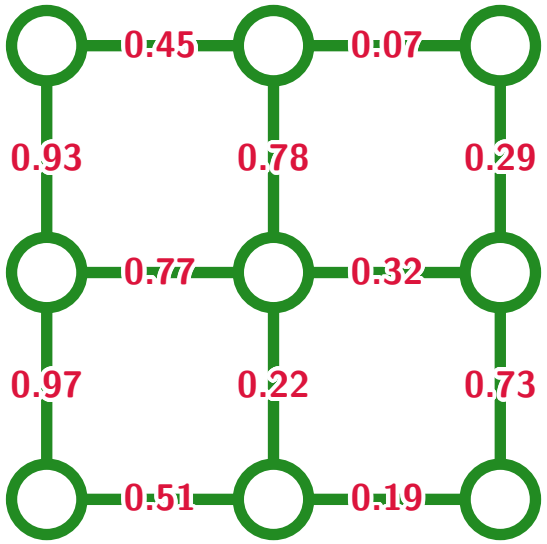
Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

- $p = 1.00$



Percolation clusters

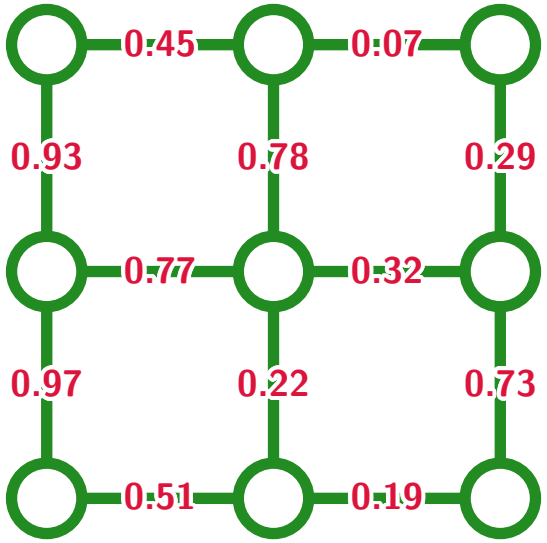
Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.



Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

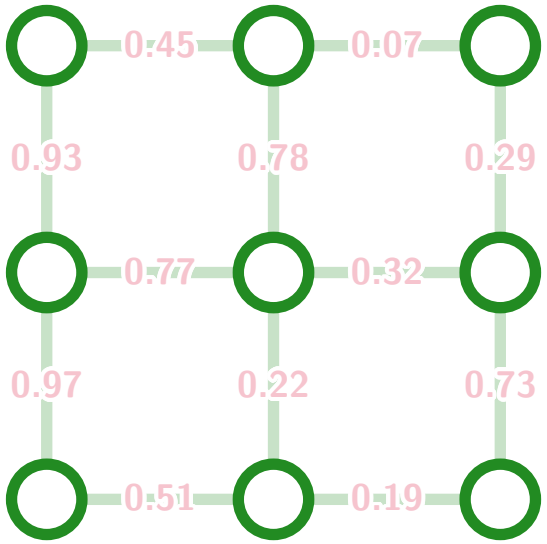


Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

- $p = 0.00$

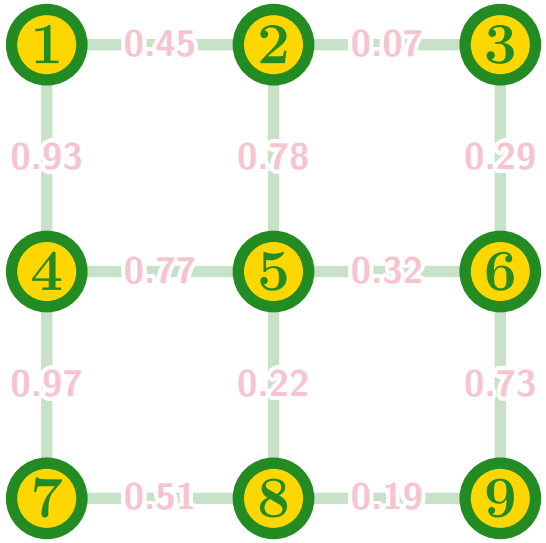


Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

- $p = 0.00$

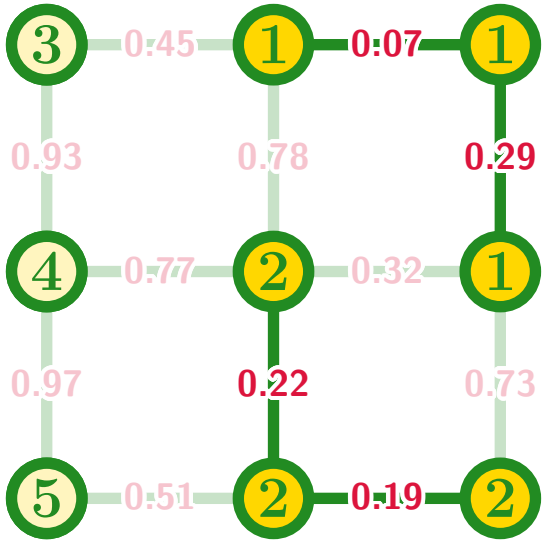


Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

- $p = 0.30$

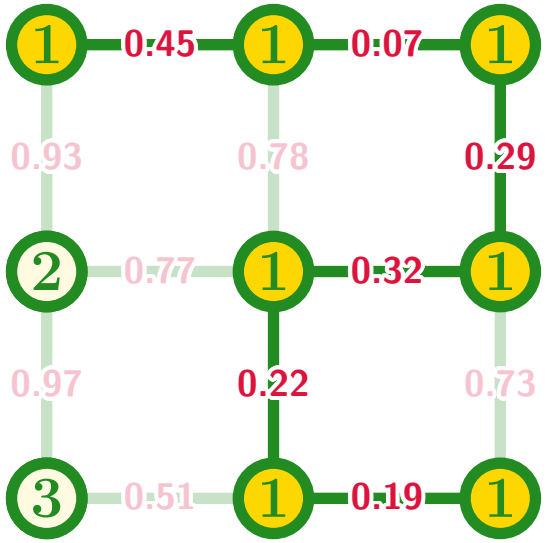


Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

- $p = 0.50$

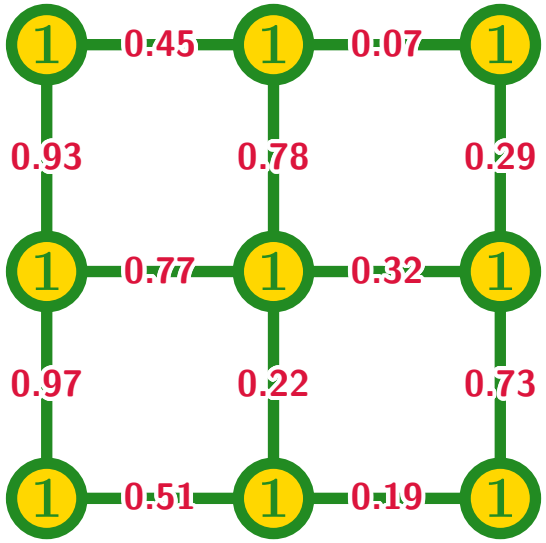


Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

- $p = 1.00$



Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

Definition (proper graph)

Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

Definition (proper graph)

A sequence of (finite) graphs $(G_n)_{n\geq 1}$ is *proper* if there exists $\theta : [0,1] \to [0,1]$ such that

$$\lim_{n\to\infty} \frac{\left|C_n^{(1)}(p)\right|}{n} = \theta(p)\,; \qquad \lim_{n\to\infty} \frac{\left|C_n^{(2)}(p)\right|}{n} = 0\,.$$

Percolation clusters

Given a weighted graph, the *percolated subgraph at level* p is the graph obained by only keeping edges whose weight is less or equal than p.

Given a percolation level p, we now consider the connected components of the graph, in decreasing order of sizes: $C^{(1)}(p), C^{(2)}(p), \ldots$

Definition (proper graph)

A sequence of (finite) graphs $(G_n)_{n\geq 1}$ is *proper* if there exists $\theta : [0,1] \to [0,1]$ such that

$$\lim_{n\to\infty} \frac{\left|C_n^{(1)}(p)\right|}{n} = \theta(p)\,; \qquad \lim_{n\to\infty} \frac{\left|C_n^{(2)}(p)\right|}{n} = 0\,.$$

Furthermore, θ should be continuous on $(p_c, 1] = \theta^{-1}((0,1])$.

Percolation clusters

Percolation clusters

Q: What happens when the graph is infinite?

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \ldots$ are usually not well-defined.

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \ldots$ are usually not well-defined.

Lucky for us, we always consider infinite graphs coupled with a root ρ.

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \dots$ are usually not well-defined.

Lucky for us, we always consider infinite graphs coupled with a root ρ.

→ Instead of ordering clusters by sizes, we only consider the cluster containing the root: $C_\rho(p)$.

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \dots$ are usually not well-defined.

Lucky for us, we always consider infinite graphs coupled with a root ρ.

→ Instead of ordering clusters by sizes, we only consider the cluster containing the root: $C_\rho(p)$.

→ We are interested in whether $C_\rho(p)$ is infinite or not.

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \dots$ are usually not well-defined.

Lucky for us, we always consider infinite graphs coupled with a root ρ.

→ Instead of ordering clusters by sizes, we only consider the cluster containing the root: $C_\rho(p)$.

→ We are interested in whether $C_\rho(p)$ is infinite or not.

→ Since the underlying graph is random, we study $\mathbb{P}(|C_\rho(p)| = \infty)$ seen as a function of $p \in [0, 1]$.

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \dots$ are usually not well-defined.

Lucky for us, we always consider infinite graphs coupled with a root ρ.

→ Instead of ordering clusters by sizes, we only consider the cluster containing the root: $C_\rho(p)$.

→ We are interested in whether $C_\rho(p)$ is infinite or not.

→ Since the underlying graph is random, we study $\mathbb{P}(|C_\rho(p)| = \infty)$ seen as a function of $p \in [0, 1]$.

Definition (proper graph)

Percolation clusters

Q: What happens when the graph is infinite?

→ The clusters $C^{(1)}(p), C^{(2)}(p), \dots$ are usually not well-defined.

Lucky for us, we always consider infinite graphs coupled with a root ρ.

→ Instead of ordering clusters by sizes, we only consider the cluster containing the root: $C_\rho(p)$.

→ We are interested in whether $C_\rho(p)$ is infinite or not.

→ Since the underlying graph is random, we study $\mathbb{P}(|C_\rho(p)| = \infty)$ seen as a function of $p \in [0, 1]$.

Definition (proper graph)

A (infinite) graph G is *proper* if $\mathbb{P}(|C_\rho(p)| = \infty) > 0$ implies that $|C_v(p)| = \infty$ for some $v \in G$.

Proper pairs

Proper pairs

Definition (proper pair)

Proper pairs

Definition (proper pair)

A pair composed of a sequence of finite graphs $(G_n)_{n\geq 1}$ and an infinite graph G is *proper* if both elements are proper and G_n converges locally towards G.

Proper pairs

Definition (proper pair)

A pair composed of a sequence of finite graphs $(G_n)_{n\geq 1}$ and an infinite graph G is *proper* if both elements are proper and G_n converges locally towards G. In that case, we further have that

$$\theta(p) := \lim_{n\to\infty} \frac{\left|C_n^{(1)}(p)\right|}{n} = \mathbb{P}\Big(\left|C_\rho(p)\right| = \infty\Big)\,.$$

Proper pairs

Definition (proper pair)

A pair composed of a sequence of finite graphs $(G_n)_{n\geq 1}$ and an infinite graph G is *proper* if both elements are proper and G_n converges locally towards G. In that case, we further have that

$$\theta(p) := \lim_{n\to\infty} \frac{\left|C_n^{(1)}(p)\right|}{n} = \mathbb{P}\Big(\left|C_\rho(p)\right| = \infty\Big)\,.$$

→ The equality states that a node chosen at random in G_n belongs to the largest component (at level p) with the same probability that we observe an infinite component (at level p) in G.

Proper pairs

Definition (proper pair)

A pair composed of a sequence of finite graphs $(G_n)_{n\geq 1}$ and an infinite graph G is *proper* if both elements are proper and G_n converges locally towards G. In that case, we further have that

$$\theta(p) := \lim_{n\to\infty} \frac{\left|C_n^{(1)}(p)\right|}{n} = \mathbb{P}\Big(\left|C_\rho(p)\right| = \infty\Big)\,.$$

→ The equality states that a node chosen at random in G_n belongs to the largest component (at level p) with the same probability that we observe an infinite component (at level p) in G.

→ Proper pairs can be seen as sequences of graphs and their limits for which the largest finite components exactly correspond to an infinite component in the limit.

Proper pairs

Proper pairs

Q: What do proper pairs and improper pairs look like?

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.
  - Geometric graphs, preferential attachment graphs, Erdős-Rényi graphs, etc.

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.
  - Geometric graphs, preferential attachment graphs, Erdős-Rényi graphs, etc.
- Graphs need to be purposefully constructed to be improper:

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.
  - Geometric graphs, preferential attachment graphs, Erdős-Rényi graphs, etc.
- Graphs need to be purposefully constructed to be improper:
  - Attaching two sequences of graphs via one edge usually provides improper (finite) graphs.

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.
  - Geometric graphs, preferential attachment graphs, Erdős-Rényi graphs, etc.
- Graphs need to be purposefully constructed to be improper:
  - Attaching two sequences of graphs via one edge usually provides improper (finite) graphs.
  - In that case, the limit could still be (infinitely) proper.

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.
  - Geometric graphs, preferential attachment graphs, Erdős-Rényi graphs, etc.
- Graphs need to be purposefully constructed to be improper:
  - Attaching two sequences of graphs via one edge usually provides improper (finite) graphs.
  - In that case, the limit could still be (infinitely) proper.
  - For an improper (infinite) graph, take the limit of two sequences of graphs with different critical percolation values.

Proper pairs

Q: What do proper pairs and improper pairs look like?

- Most "natural" pairs of graphs and their limits are proper:
  - Finite and infinite grids of any dimension.
  - Configuration models and branching processes.
  - Geometric graphs, preferential attachment graphs, Erdős-Rényi graphs, etc.
- Graphs need to be purposefully constructed to be improper:
  - Attaching two sequences of graphs via one edge usually provides improper (finite) graphs.
  - In that case, the limit could still be (infinitely) proper.
  - For an improper (infinite) graph, take the limit of two sequences of graphs with different critical percolation values.

→ Attach a 3- and a 4-regular graph together to obtain an improper pair, both finite and infinite.

Table of Content

Our result

Our result

Theorem (, Gündlach, & van der Hofstad, 2025+)

Our result

Theorem (, Gündlach, & van der Hofstad, 2025+)

Let $(G_n)_{n\geq 1}$ and G be a proper pair. Then, the output of Prim's algorithm on $(G_n)_{n\geq 1}$ after $tn + o(n)$ steps for any $t \in [0, 1]$ converges towards the union of the invasion percolation cluster on G along with the minimum spanning forest on G percolated at level $\theta^{-1}(t)$.

Moreover, this convergence occurs as a cadlag process on the space of local convergence.

Our result

Theorem (, Gündlach, & van der Hofstad, 2025+)

Let $(G_n)_{n\geq 1}$ and G be a proper pair. Then, the output of Prim's algorithm on $(G_n)_{n\geq 1}$ after $tn + o(n)$ steps for any $t \in [0, 1]$ converges towards the union of the invasion percolation cluster on G along with the minimum spanning forest on G percolated at level $\theta^{-1}(t)$.

Moreover, this convergence occurs as a cadlag process on the space of local convergence.

→ After $\theta(p)n$ steps, Prim's algorithm corresponds to the infinite percolation cluster and the percolated minimum spanning forest.

Our result

Theorem (, Gündlach, & van der Hofstad, 2025+)

Let $(G_n)_{n\geq 1}$ and G be a proper pair. Then, the output of Prim's algorithm on $(G_n)_{n\geq 1}$ after $tn + o(n)$ steps for any $t \in [0, 1]$ converges towards the union of the invasion percolation cluster on G along with the minimum spanning forest on G percolated at level $\theta^{-1}(t)$.

Moreover, this convergence occurs as a cadlag process on the space of local convergence.

→ After $\theta(p)n$ steps, Prim's algorithm corresponds to the infinite percolation cluster and the percolated minimum spanning forest.

→ Let me now explain this more instinctively.

Theorem (, Gündlach, & van der Hofstad, 2025+)

Let $(G_n)_{n\geq 1}$ and G be a proper pair. Then, the output of Prim's algorithm on $(G_n)_{n\geq 1}$ after $tn + o(n)$ steps for any $t \in [0, 1]$ converges towards the union of the invasion percolation cluster on G along with the minimum spanning forest on G percolated at level $\theta^{-1}(t)$.

Moreover, this convergence occurs as a cadlag process on the space of local convergence.

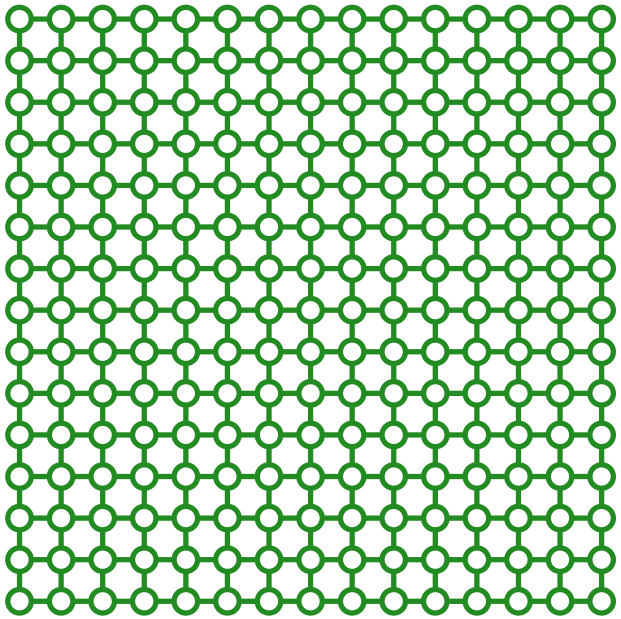
→ After $\theta(p)n$ steps, Prim's algorithm corresponds to the infinite percolation cluster and the percolated minimum spanning forest.

→ Let me now explain this more instinctively.

Our result

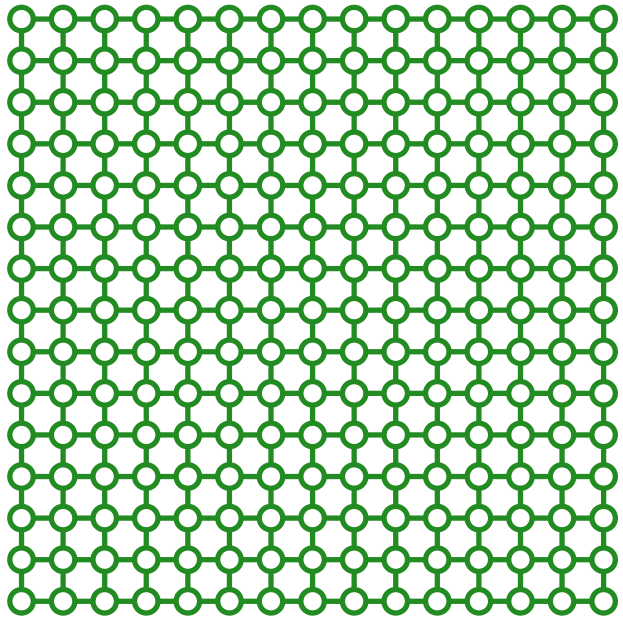
$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

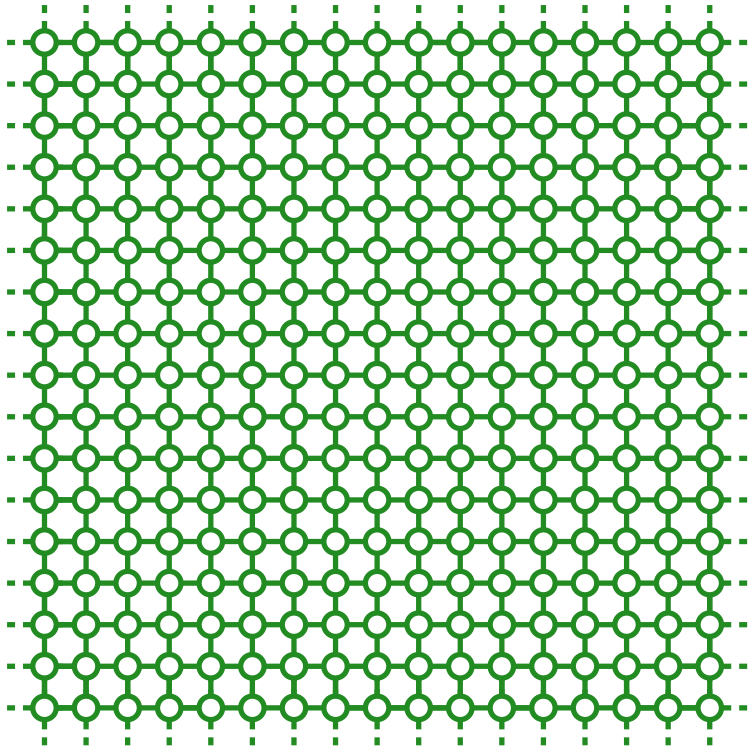
- Take a graph



$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

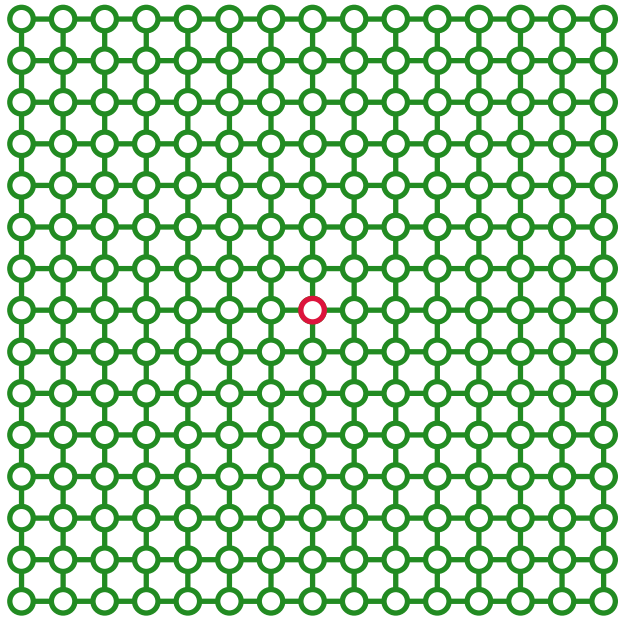
- Take a graph and its local limit

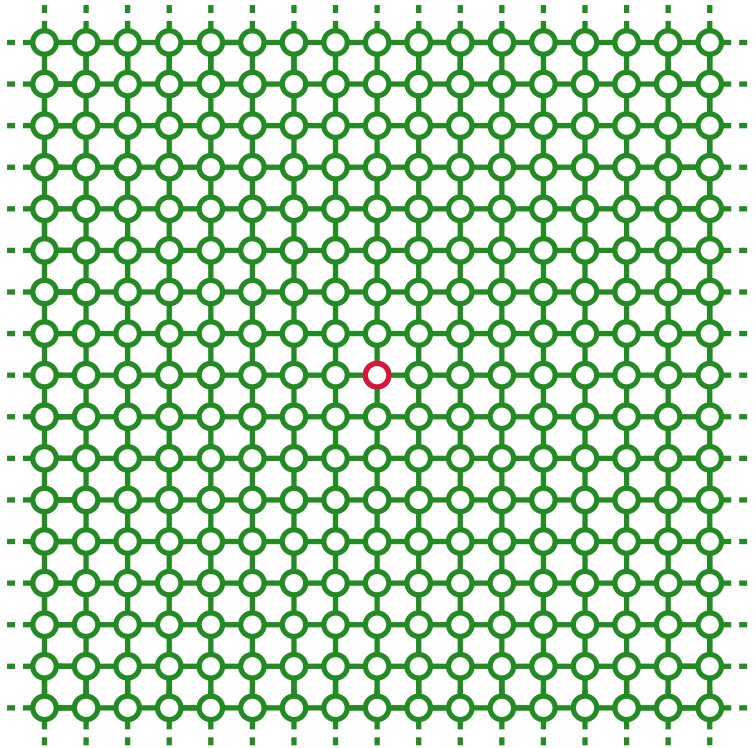




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

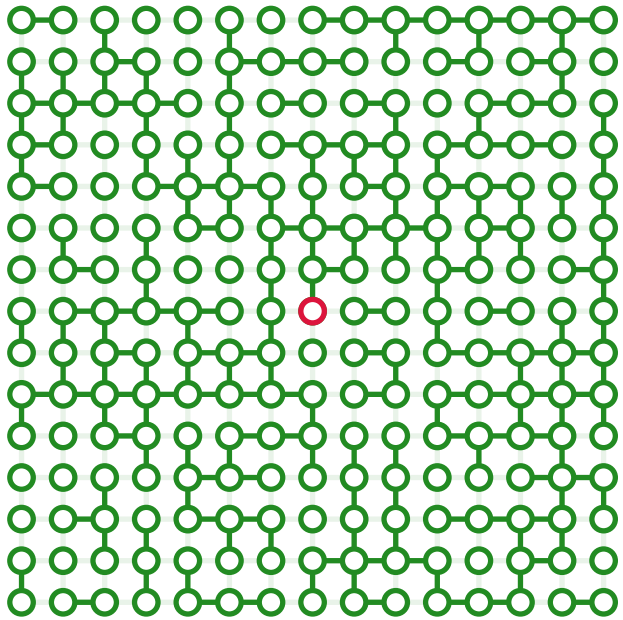
- Take a graph and its local limit both rooted at some node.

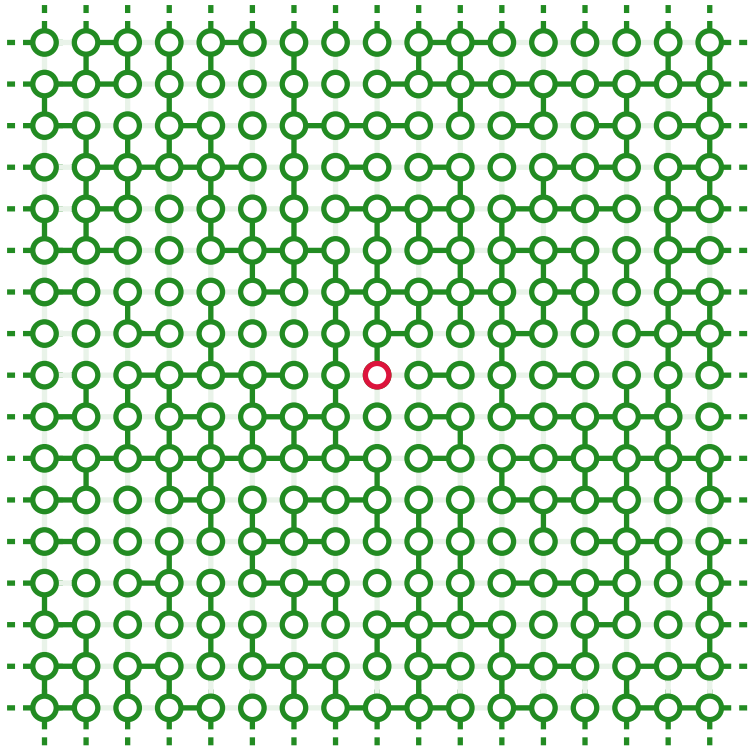




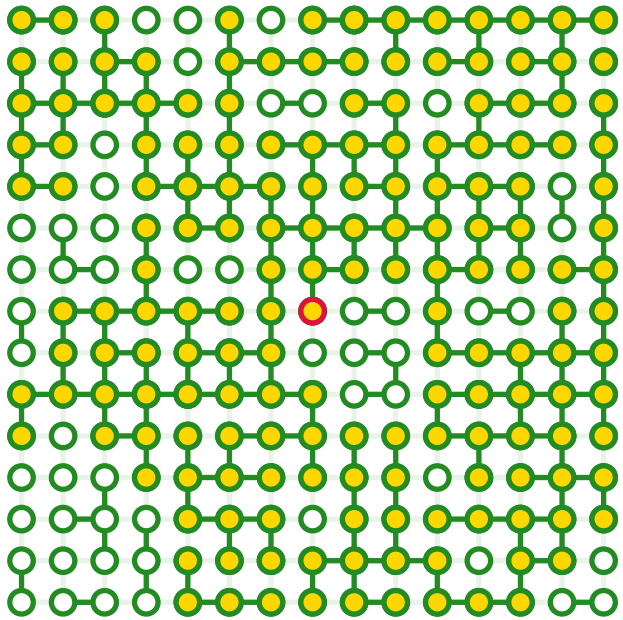
$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

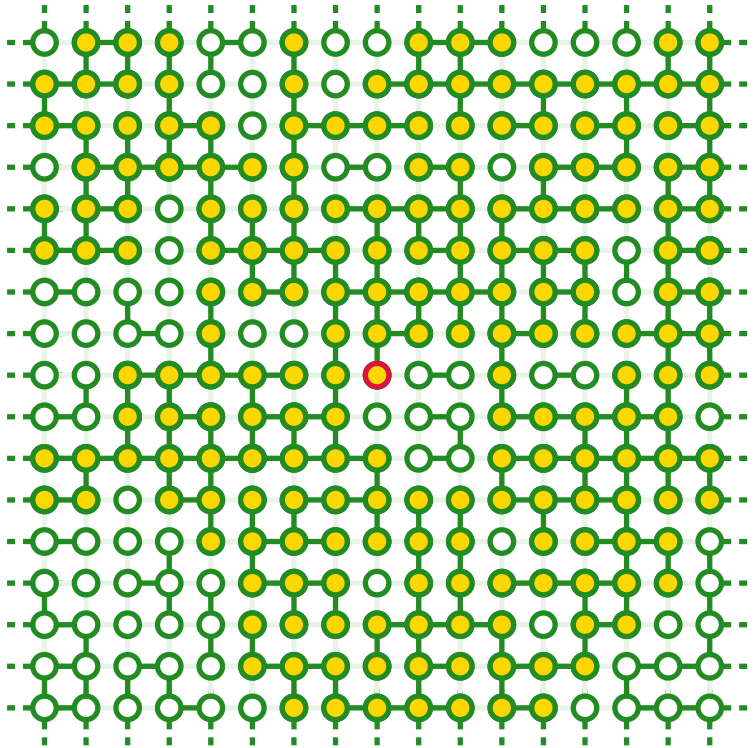
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$





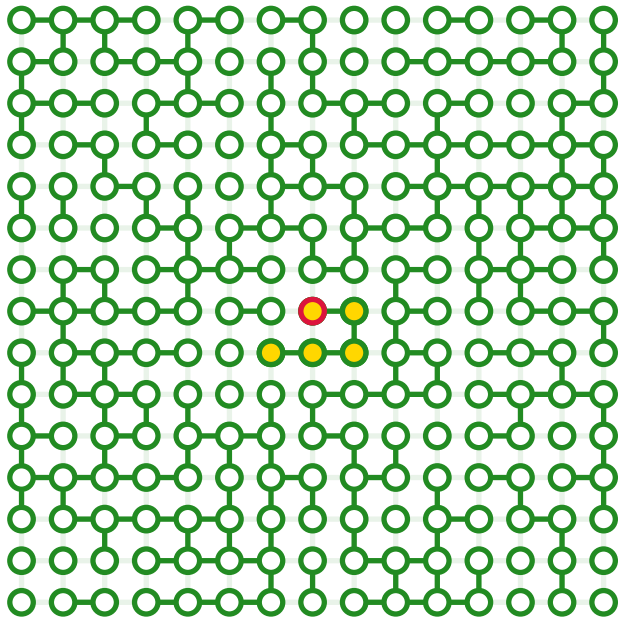
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.

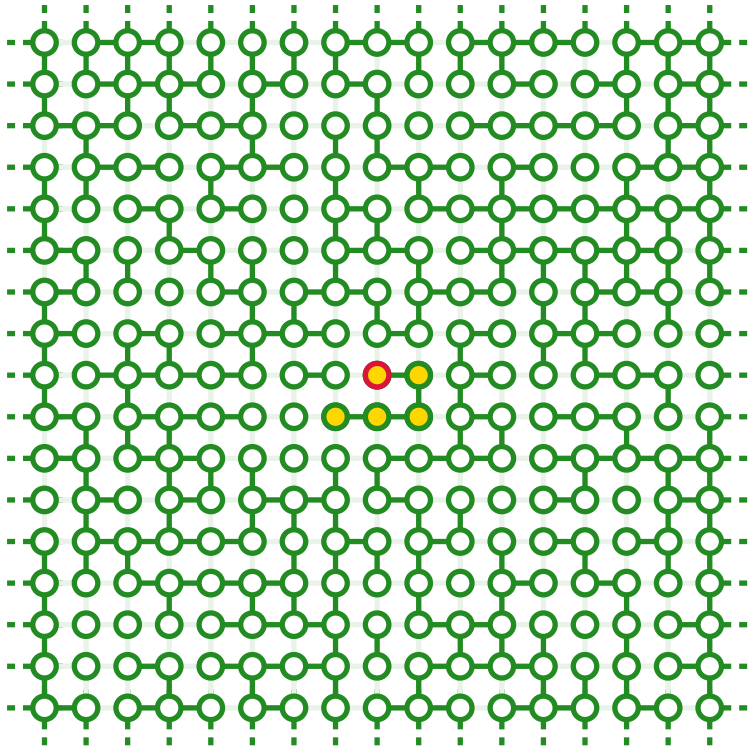




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

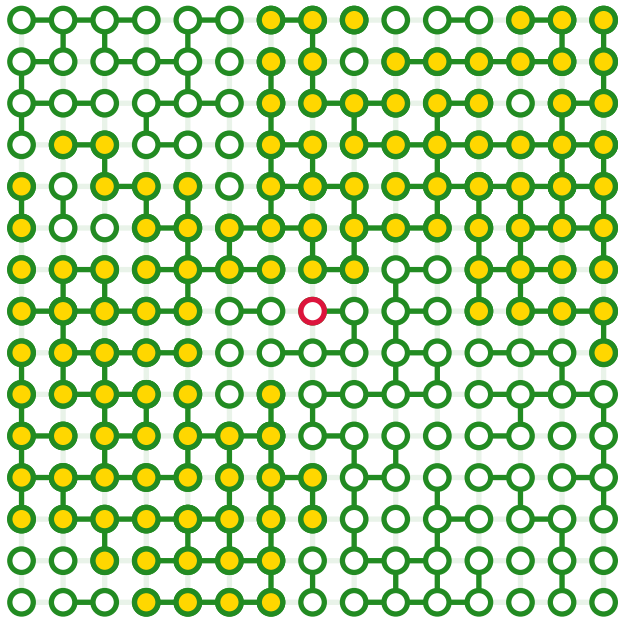
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.

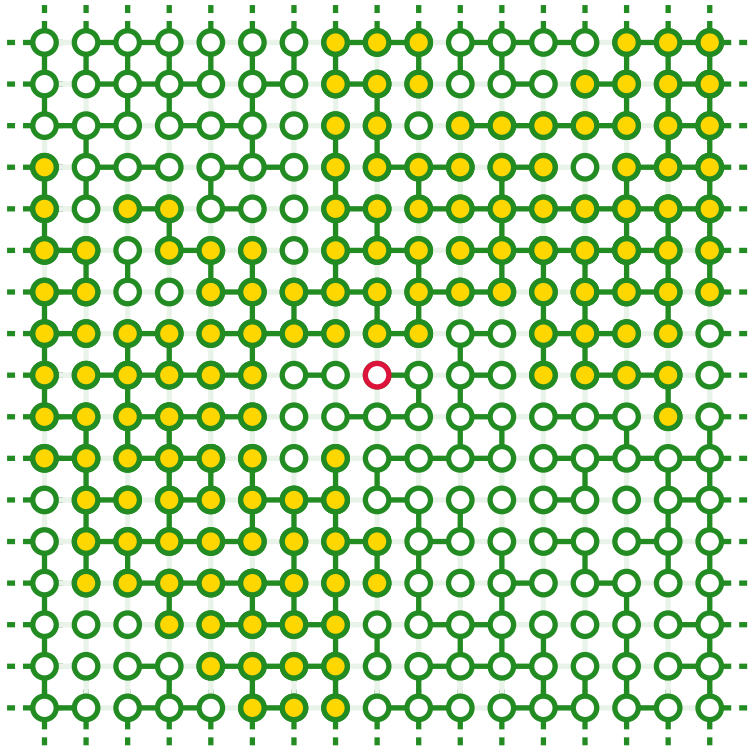




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

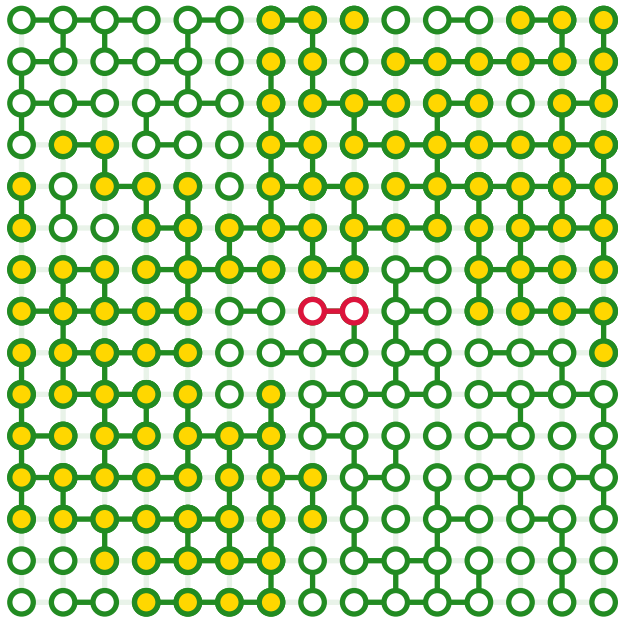
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.

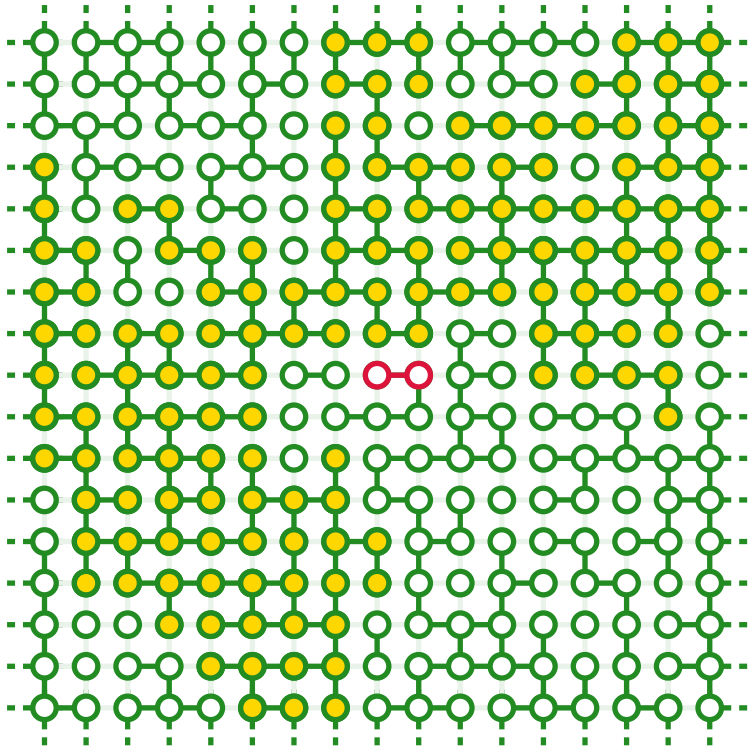




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

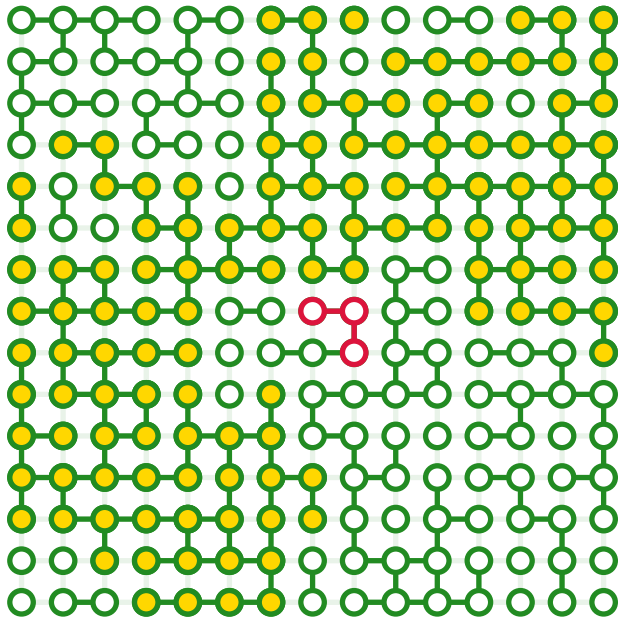
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.

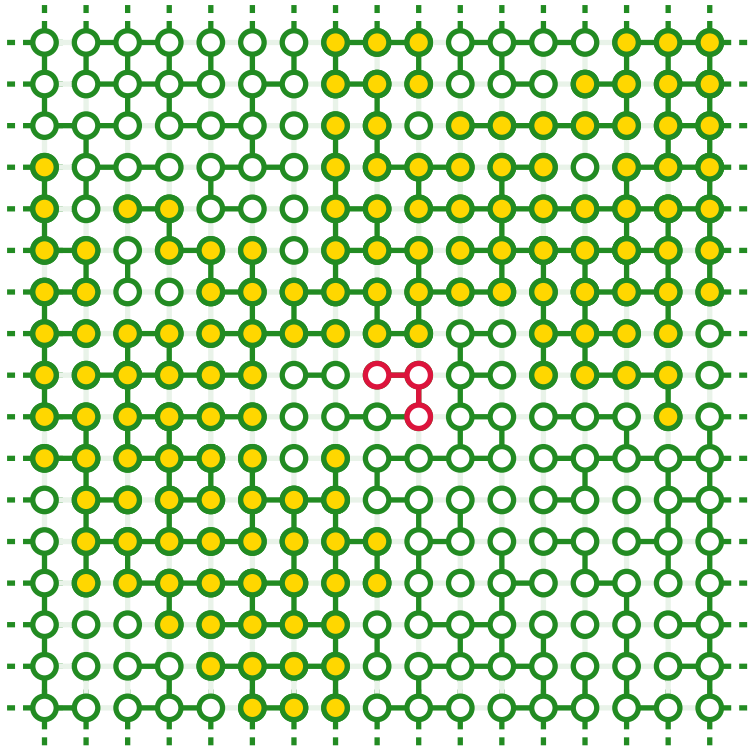




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

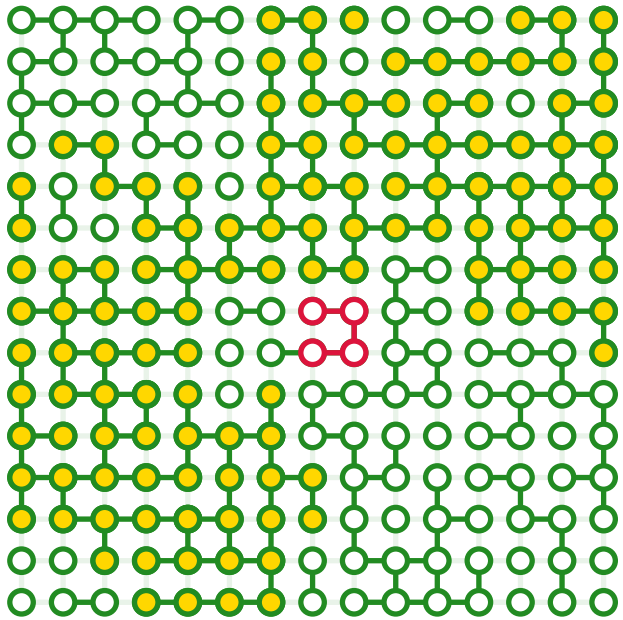
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.

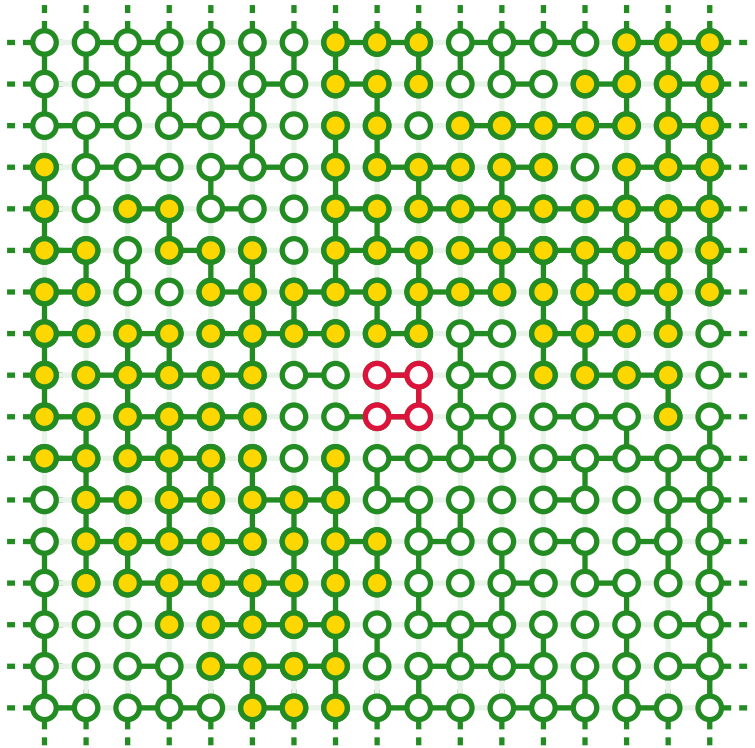




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

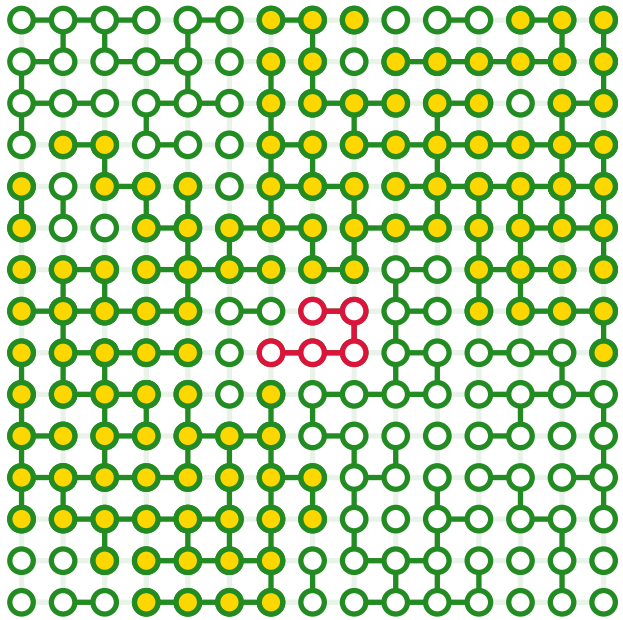
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.

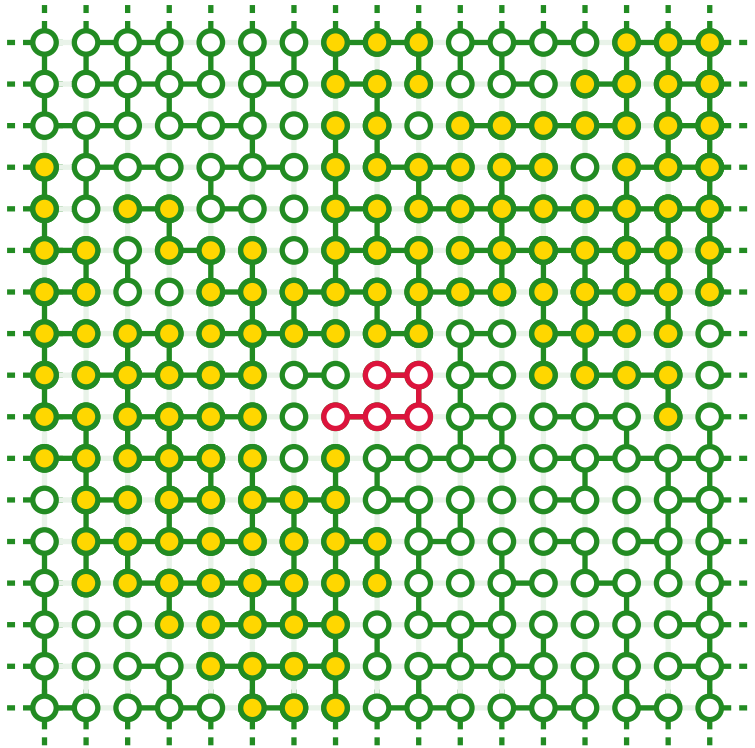




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

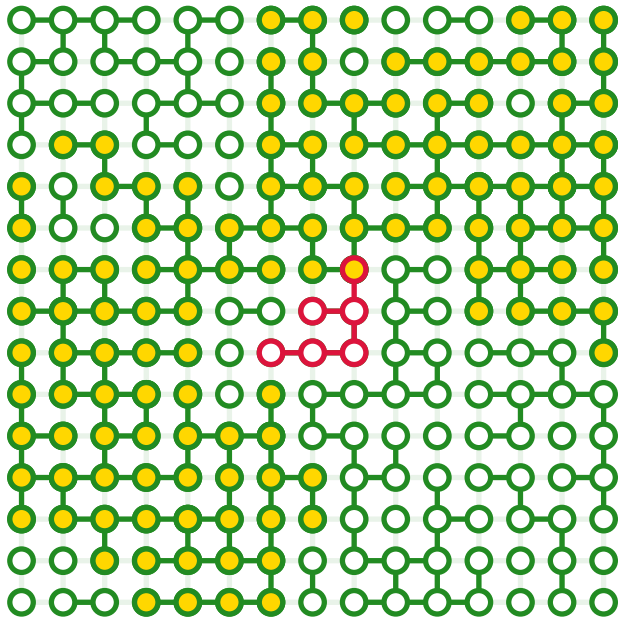
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.

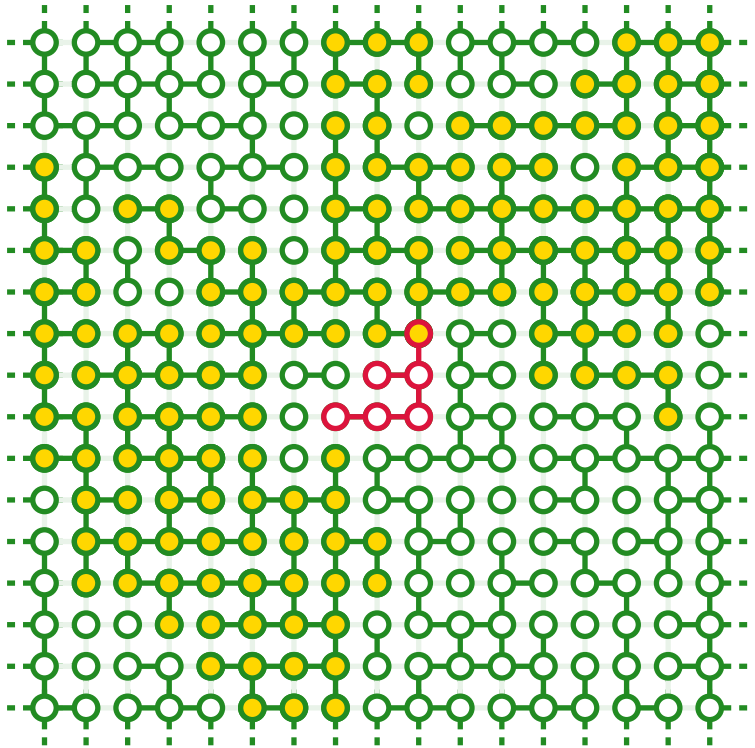




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

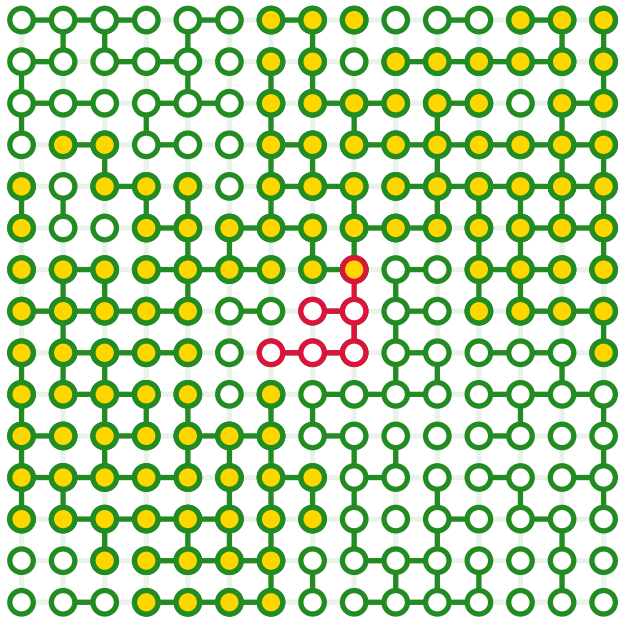
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.

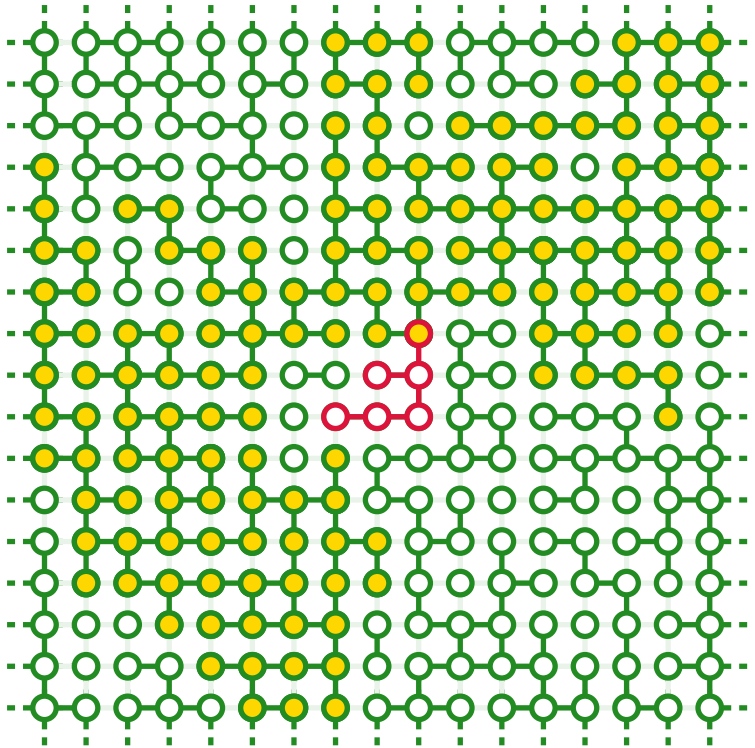




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

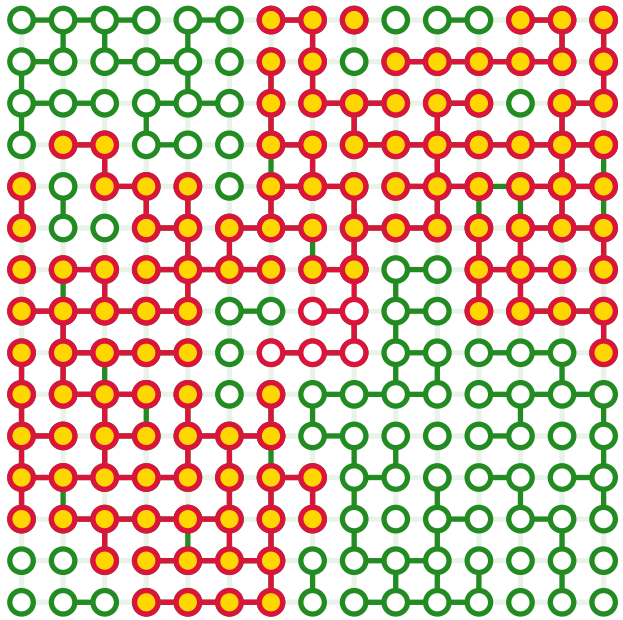
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.
- After $\simeq \theta(p)n$ steps:

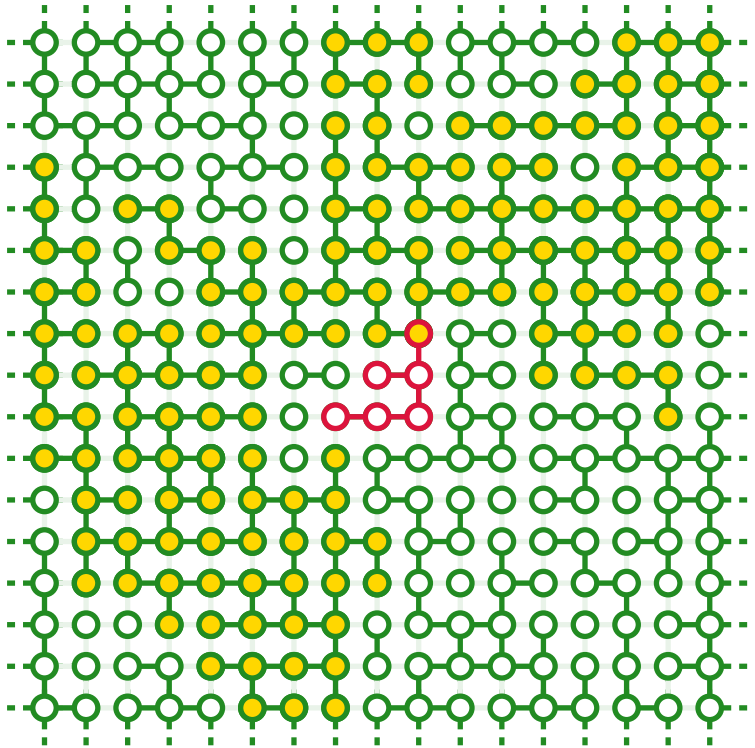




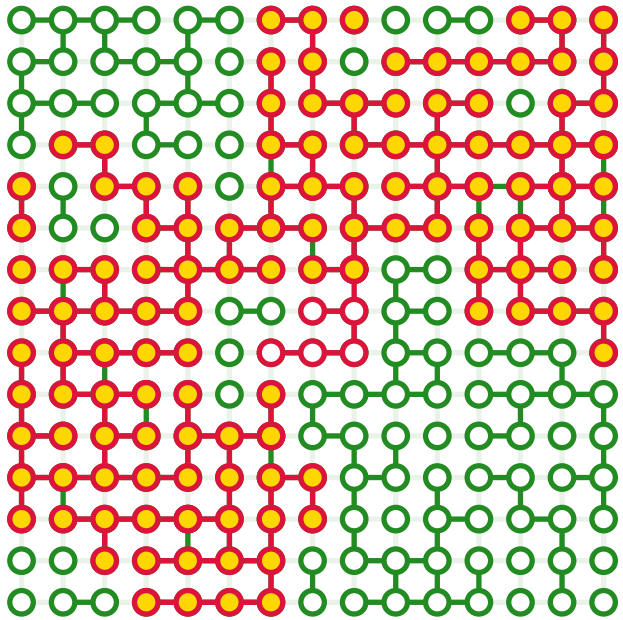
$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

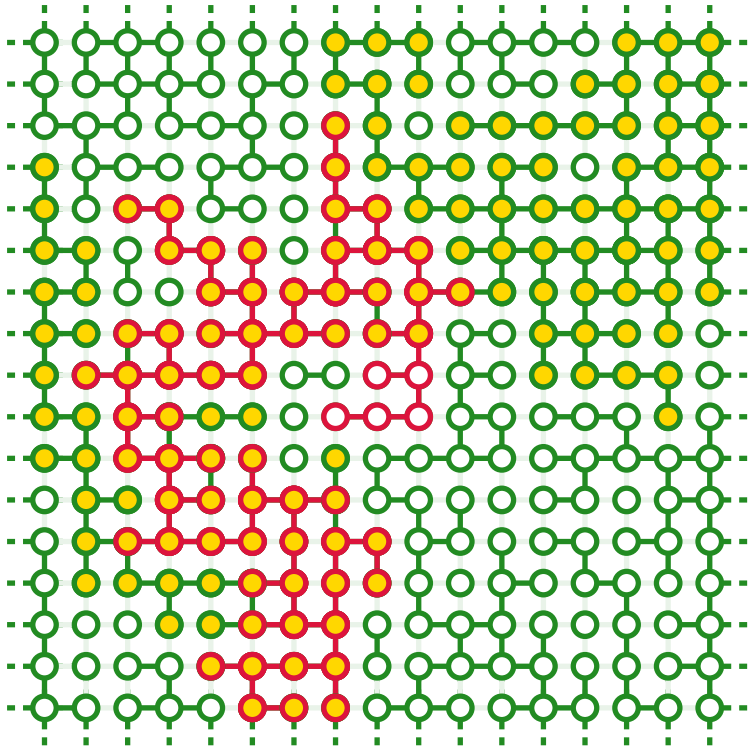
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.
- After $\simeq \theta(p)n$ steps:





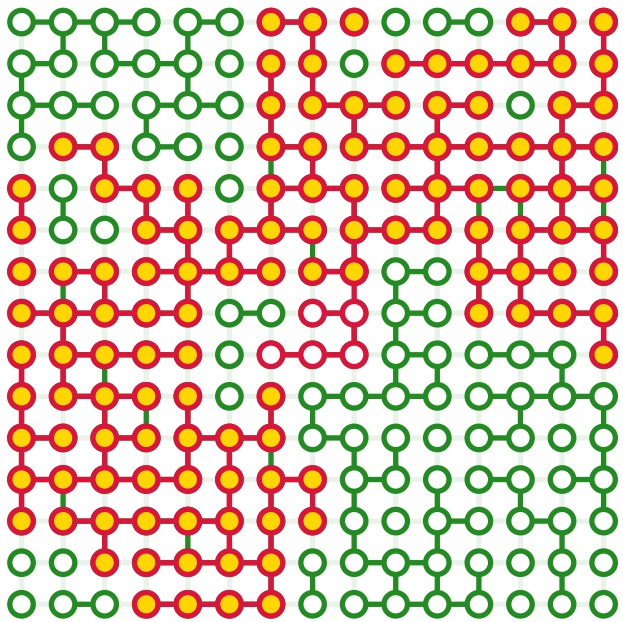
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.
- After $\simeq \theta(p)n$ steps:

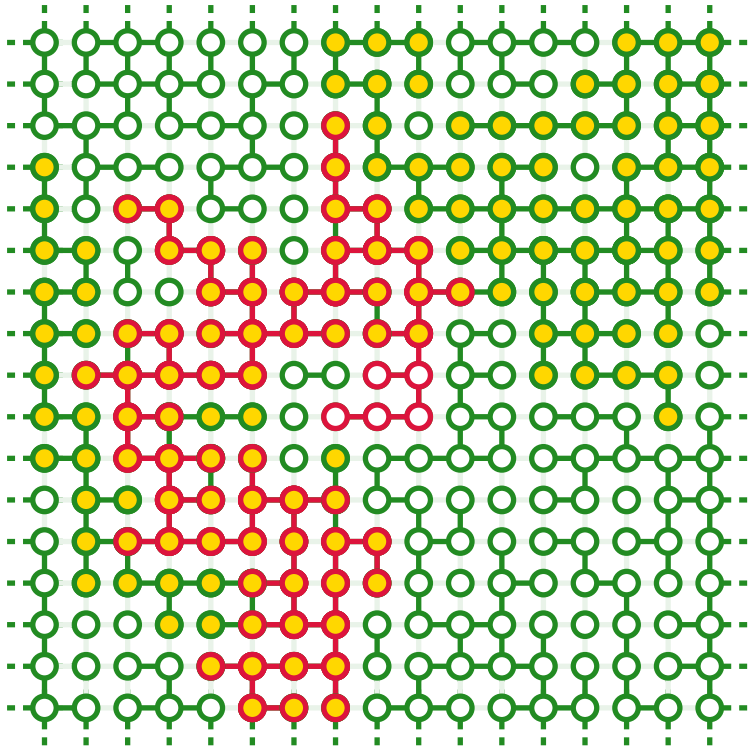




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

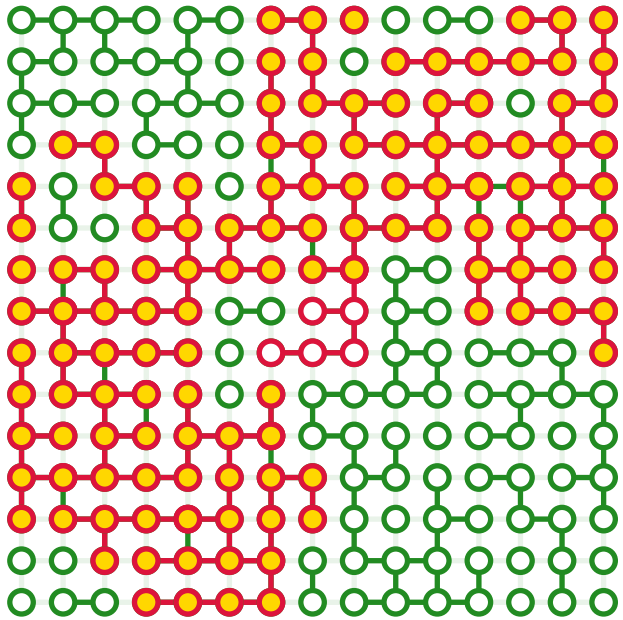
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0, 1]$ and identify the large components.
- Start running Prim's algorithm on both sides.
- After $\simeq \theta(p)n$ steps:
  - finite and infinite look different;

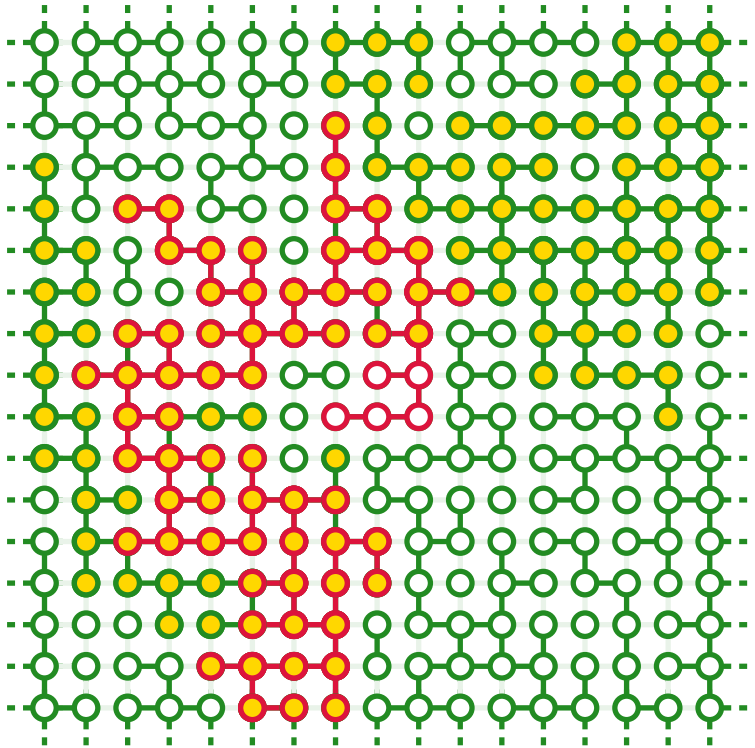




$\mathrm{Prim}\big(\theta(p)n\big) \longrightarrow \mathrm{IPC} + \mathrm{MST}(p)$

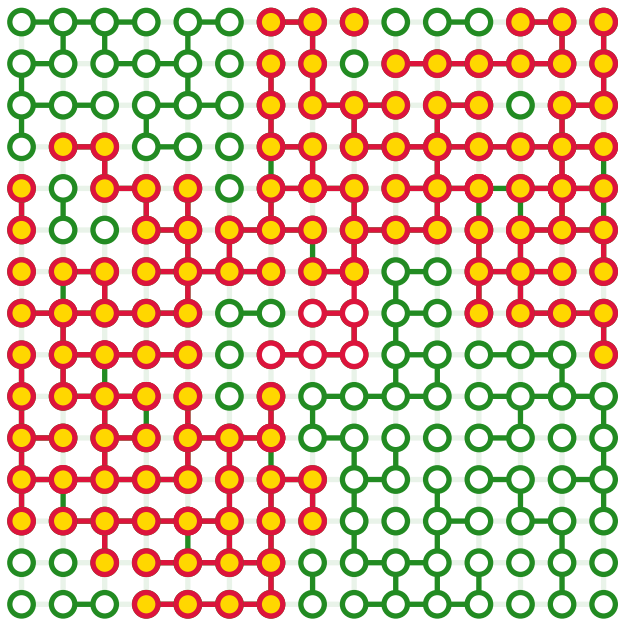
- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0,1]$ and identify the large components.
- Start running Prim's algorithm on both sides.
- After $\simeq \theta(p)n$ steps:
  - finite and infinite look different;
  - but finite Prim equals infinite Prim combined with percolation.

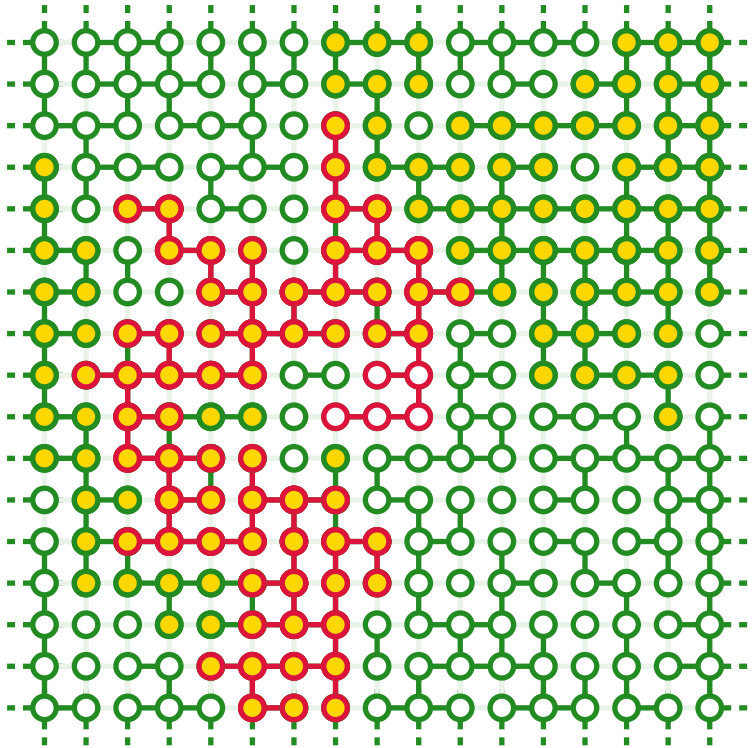




- Take a graph and its local limit both rooted at some node.
- Apply percolation for some $p \in [0,1]$ and identify the large components.
- Start running Prim's algorithm on both sides.
- After $\simeq \theta(p)n$ steps:
  - finite and infinite look different;
  - but finite Prim equals infinite Prim combined with percolation.

→ CQFD 👍





References

- Aldous, D., & Steele, J. M. (2004). **The objective method: probabilistic combinatorial optimization and local weak convergence.** *Probability on Discrete Structures, 110, 1-72.*
- Corsini, B., Gündlach, R., & van der Hofstad, R. (2025+). **Local limit of Prim's algorithm.** *preprint.*
- van der Hofstad, R. (2024). **Random Graphs and Complex Networks, Volume 2.** *Cambridge University Press.*
- Prim, R. C. (1957). **Shortest connection networks and some generalizations.** *Bell System Technical Journal, 36(6), 1389-1401.*

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement No. 101034253 .


Thank you!
Thank you!
Thank you!
Thank you!
Thank you!
Thank you!
Thank you!