Sampling Vertices Uniformly from a Graph

Flavio Chierichetti *Sapienza University*

With subsets of
Anirban Dasgupta *IIT Gandhinagar*
Shahrzad Haddadan *Sapienza University*
Silvio Lattanzi *Google Zurich*
Ravi Kumar *Google MTV*
Tamás Sarlós *Google MTV*

Social Networks

- Social Networks are “large”
- We would like to study their properties
- We need to be able to sample from them

Learning Average Opinions

Learning Average Opinions

Learning Average Opinions

Learning Average Opinions

Learning Average Opinions

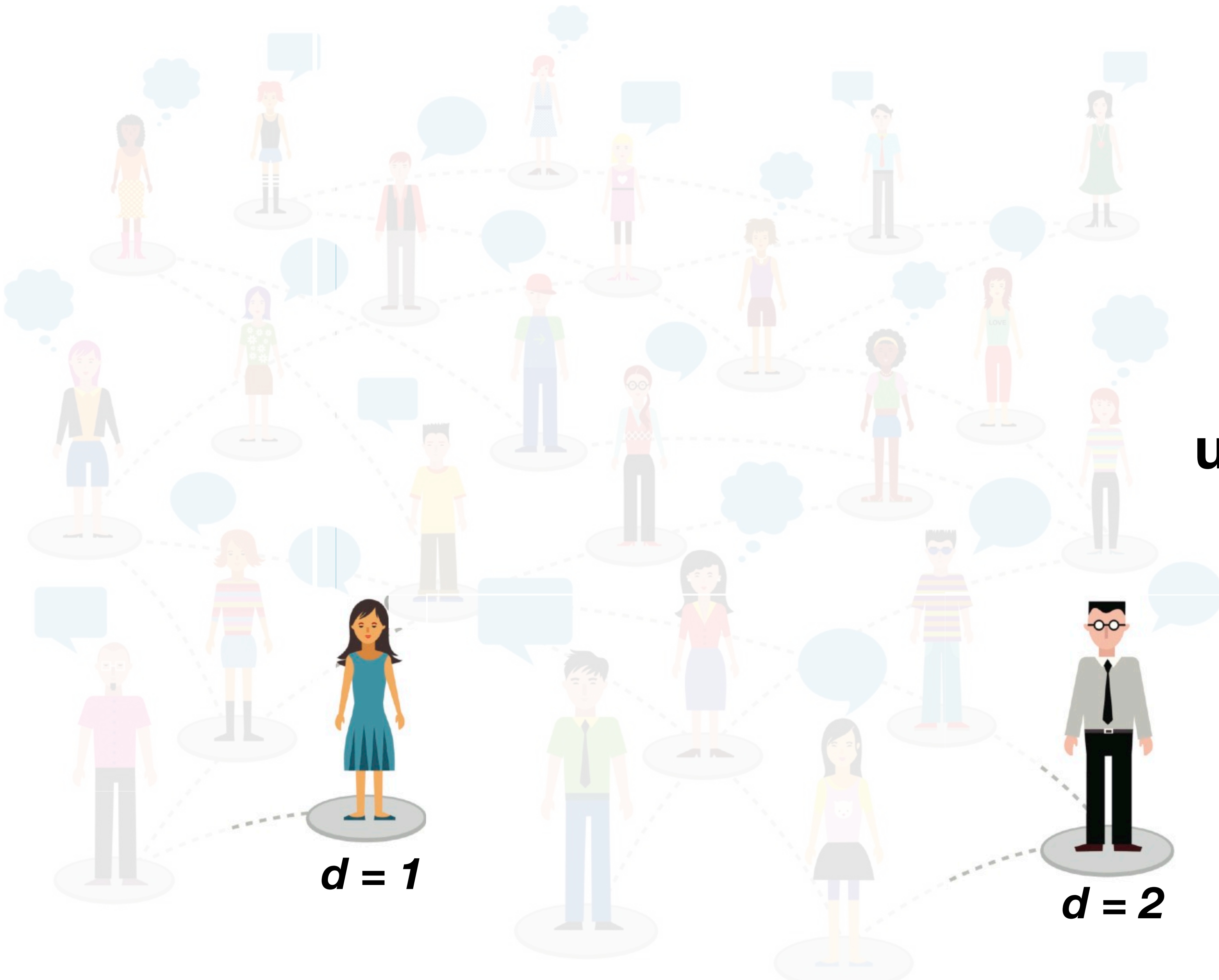
Learning Average Opinions

Asking all the users is too costly!

Learning Average Opinions

Select some people **uniformly-at-random** and ask them their opinion

Learning Average Opinions



Select some people
uniformly-at-random
and ask them
their opinion

Learning Average Opinions

Select some people **uniformly-at-random** and ask them their opinion

Learning Average Opinions

Select some people **uniformly-at-random** and ask them their opinion

The empirical average will be close to the real average

Learning Average Opinions

Learning Average Opinions

Learning Average Opinions

What is the fraction of  ?

Learning Average Opinions

Select some people
uniformly-at-random
and ask them
their opinion

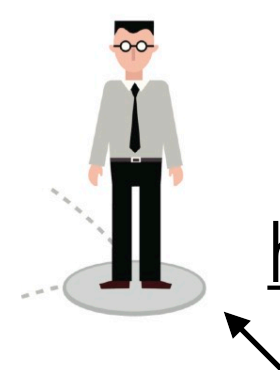
Learning Average Opinions

Select some people **uniformly-at-random** and ask them their opinion

The empirical fraction of will be close to the real fraction

How do we select uniform-at-random profiles in a Social Network?

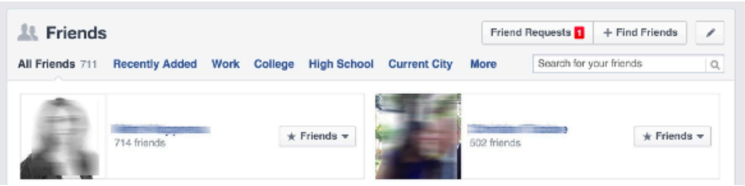
- We can access the SN through a crawling process.



http://s-n.com/001.html

How do we select uniform-at-random profiles in a Social Network?

- We can access the SN through a crawling process.

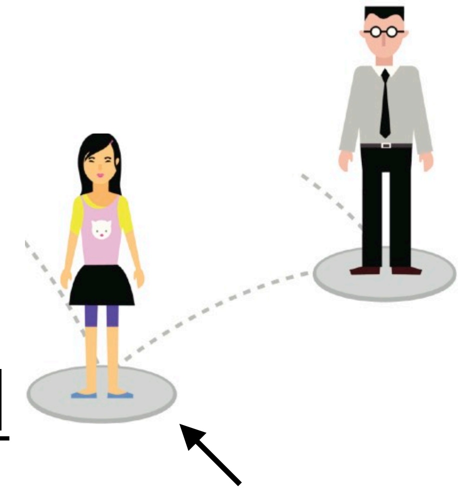


http://s-n.com/001.html

How do we select uniform-at-random profiles in a Social Network?

- We can access the SN through a crawling process.

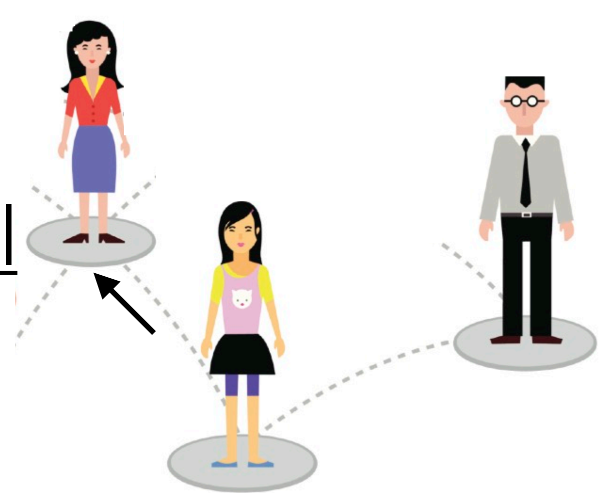
http://s-n.com/005.html



How do we select uniform-at-random profiles in a Social Network?

- We can access the SN through a crawling process.

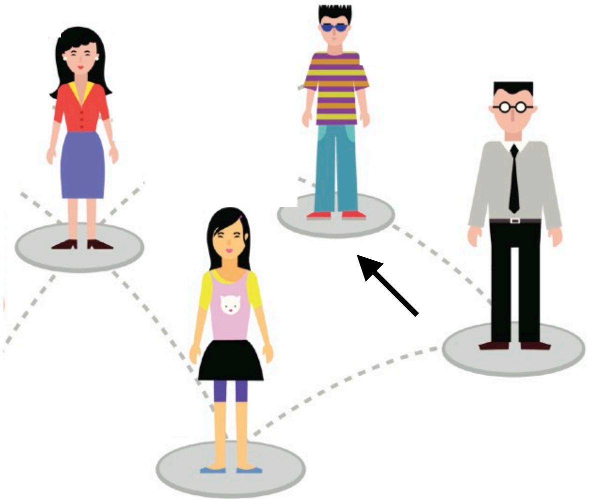
http://s-n.com/011.html



How do we select uniform-at-random profiles in a Social Network?

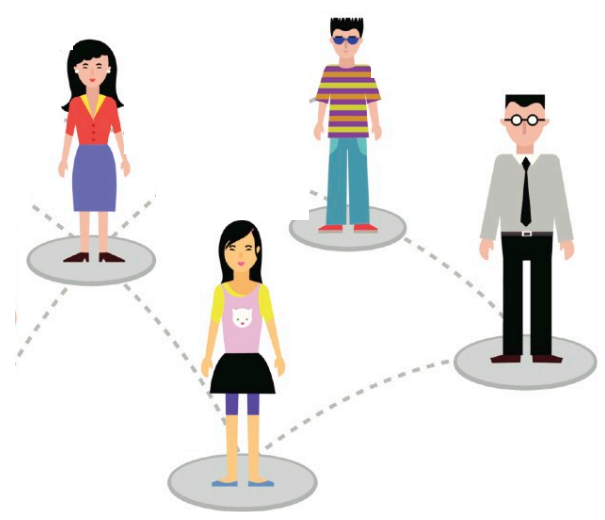
- We can access the SN through a crawling process.

http://s-n.com/012.html

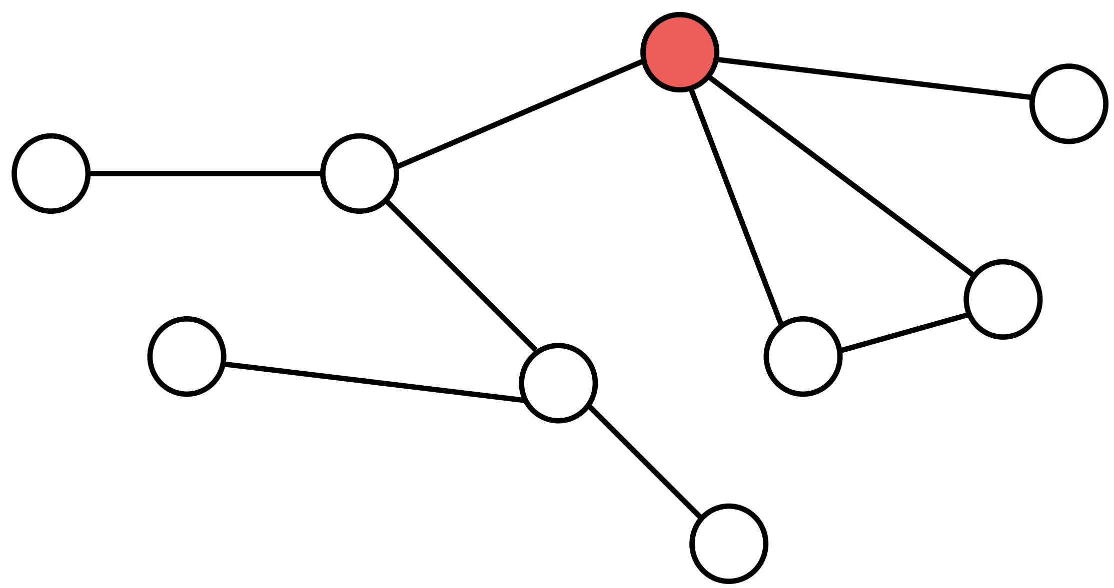


How do we select uniform-at-random profiles in a Social Network?

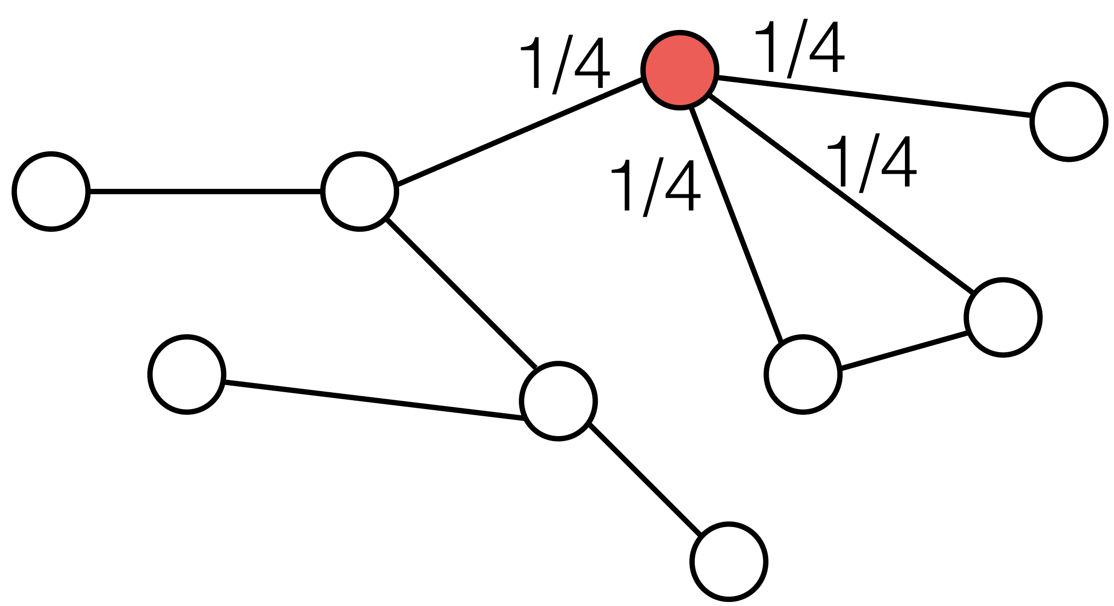
- We can access the SN through a crawling process.
- We **cannot** crawl the whole network.



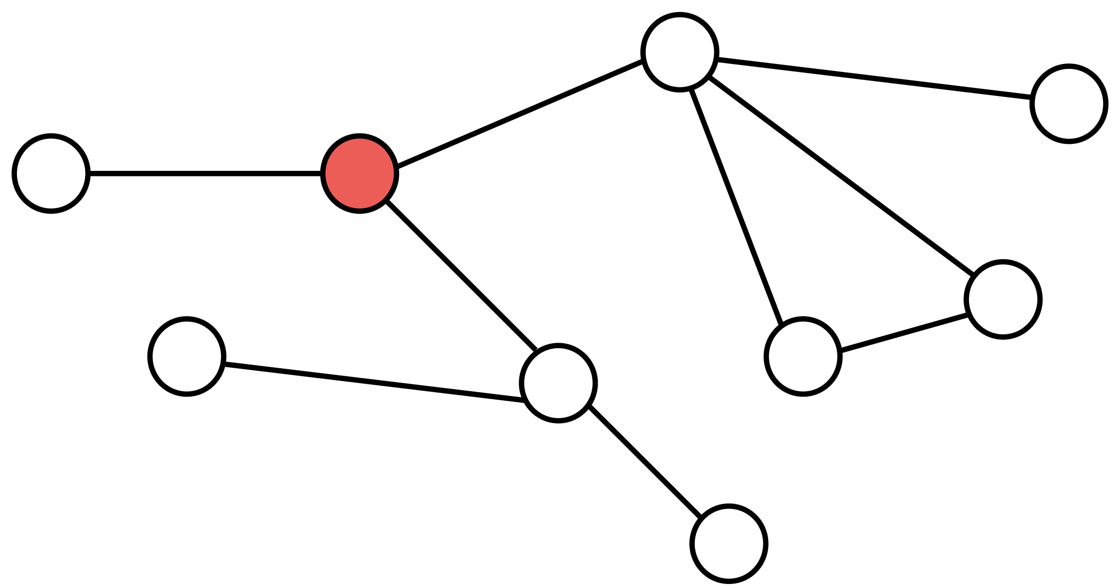
Random Walks



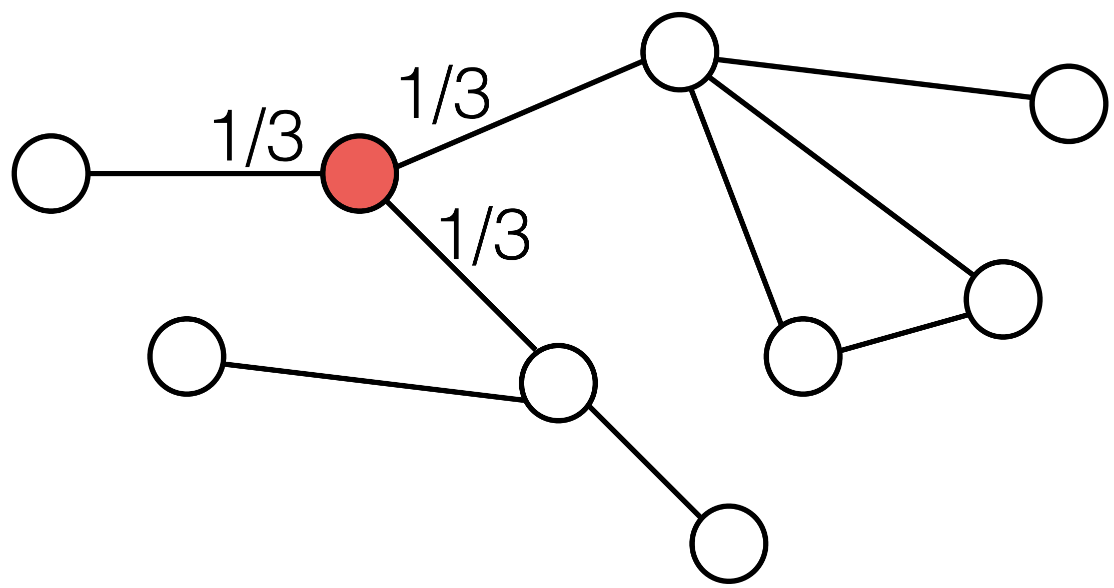
Random Walks



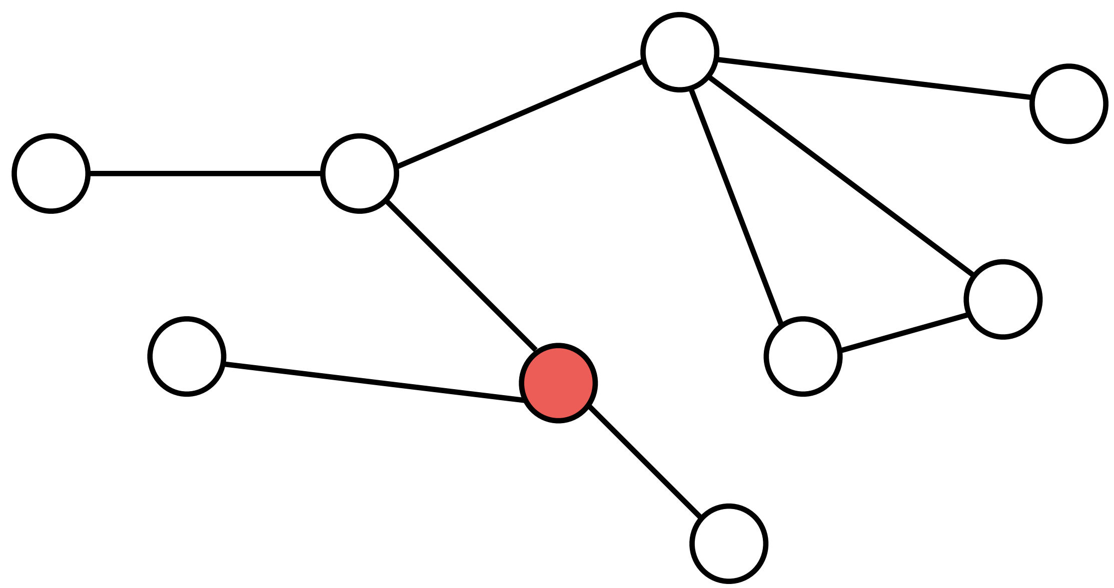
Random Walks



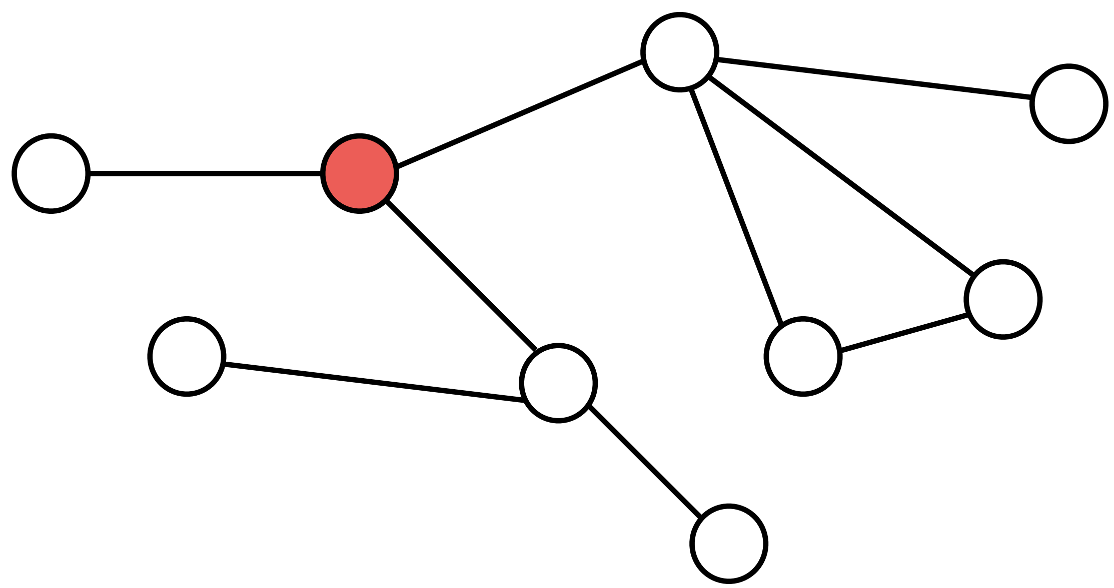
Random Walks



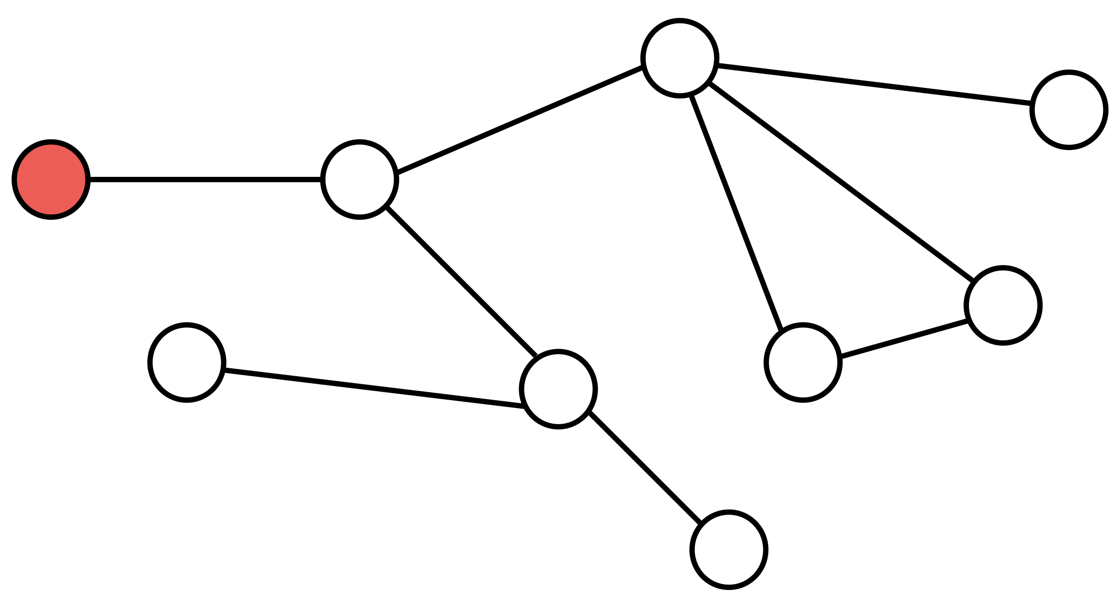
Random Walks



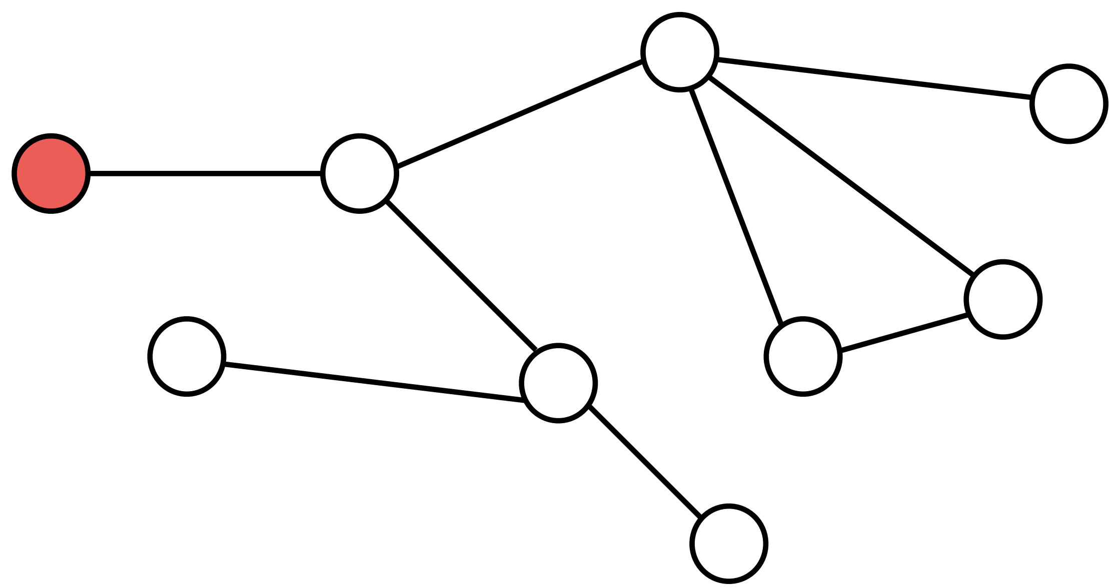
Random Walks



Random Walks

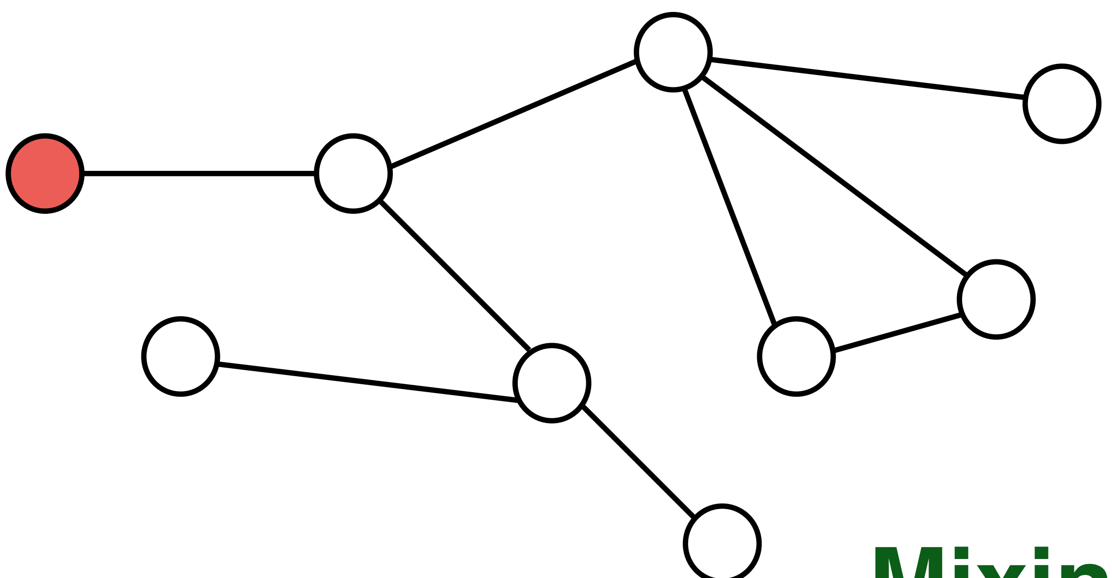


Random Walks



If the process goes on for enough many steps,
the random node it ends up on will be “random”

Random Walks

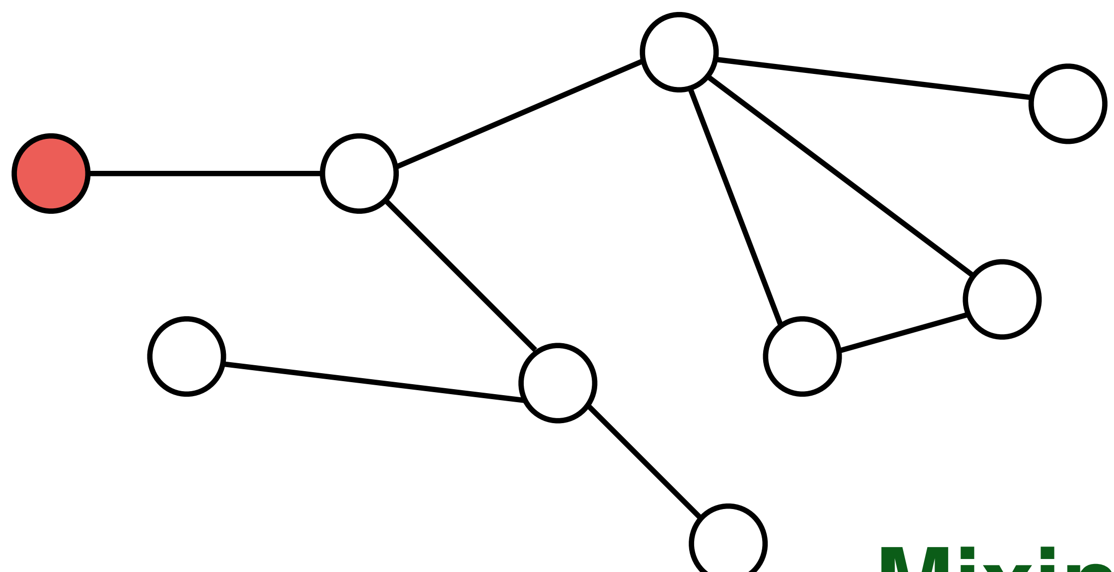


Mixing Time T(G)

If the process goes on for enough many steps,
the random node it ends up on will be “random”

Random Walks

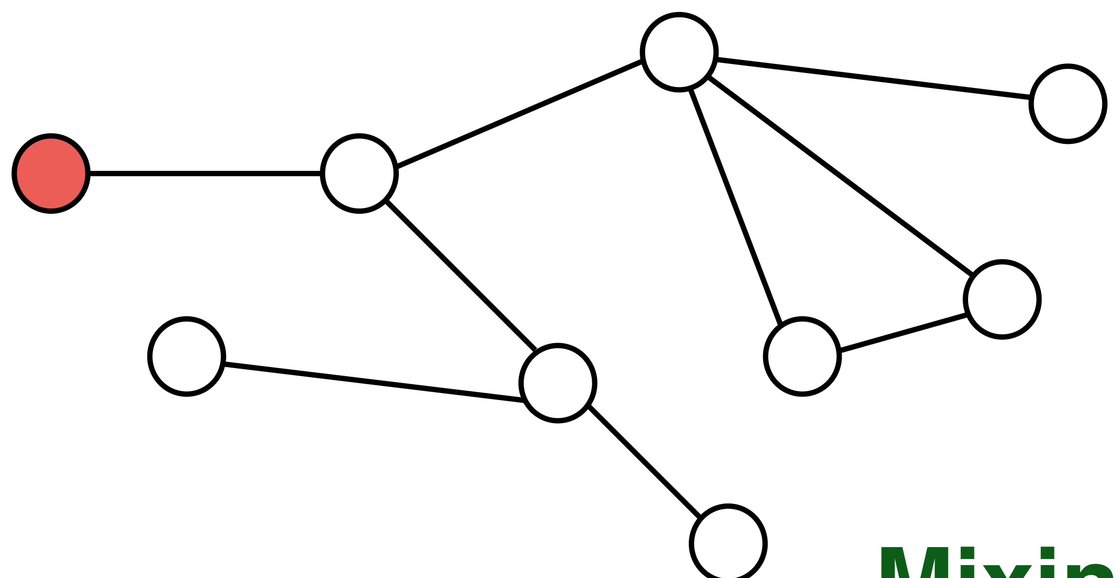
The Mixing Times of many "Social Networks" are small
[Leskovec et al, '08]



Mixing Time T(G)

If the process goes on for enough many steps,
the random node it ends up on will be "random"

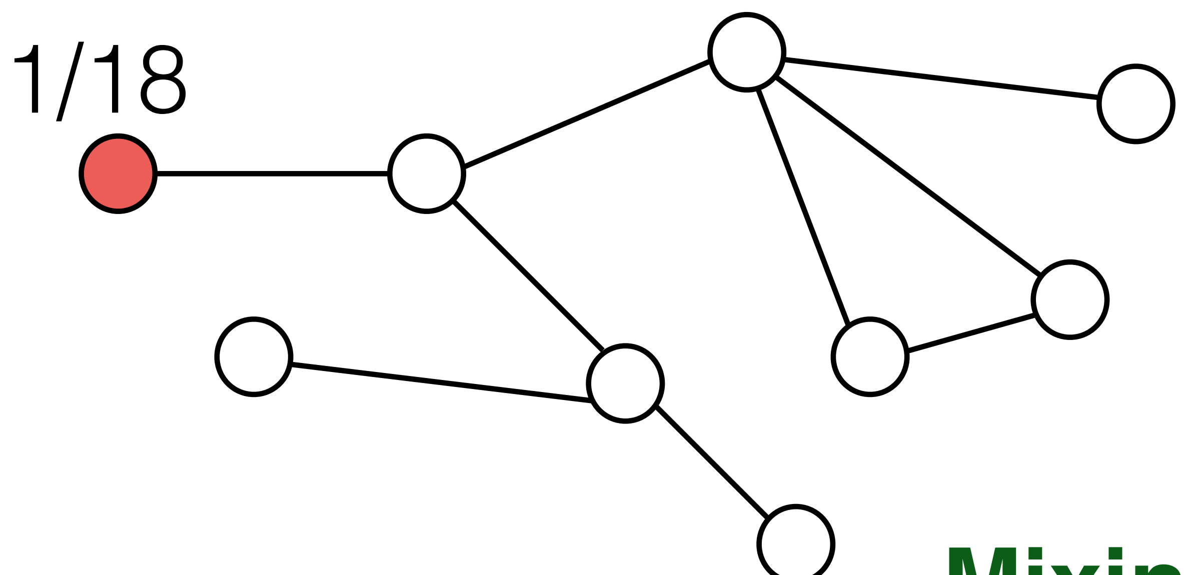
Random Walks



Mixing Time T(G)

If the process goes on for enough many steps,
the random node it ends up on will be “random”,
chosen with probability proportional to its degree

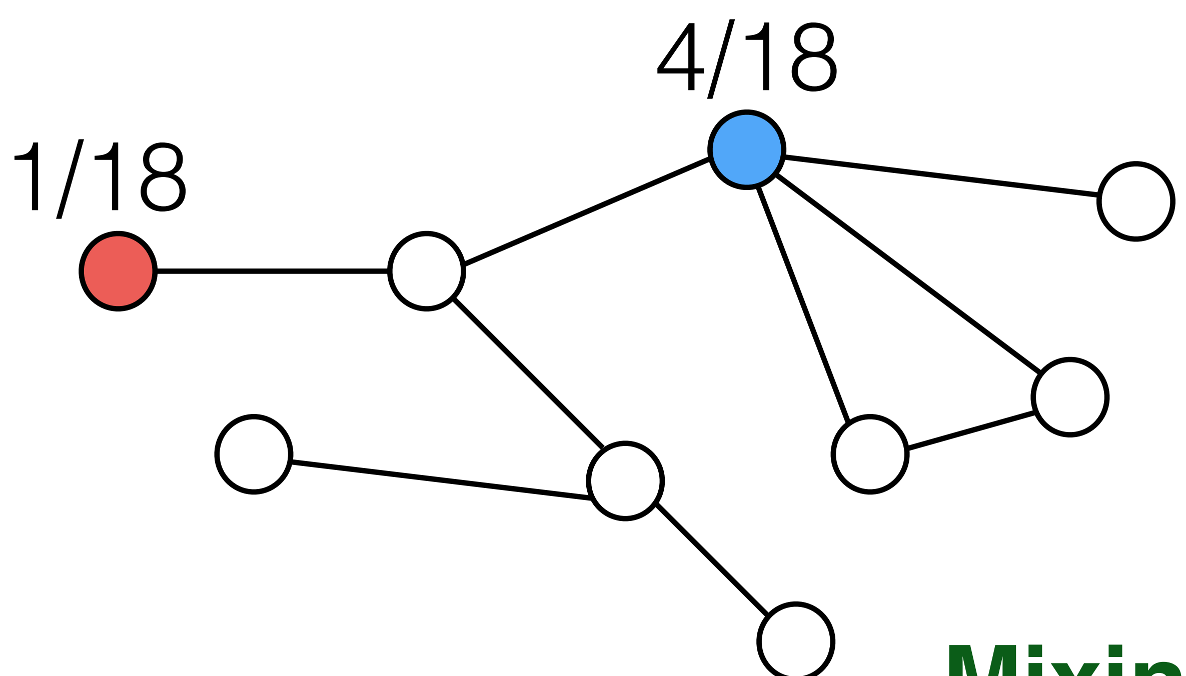
Random Walks



Mixing Time T(G)

If the process goes on for enough many steps, the random node it ends up on will be "random", *chosen with probability proportional to its degree*

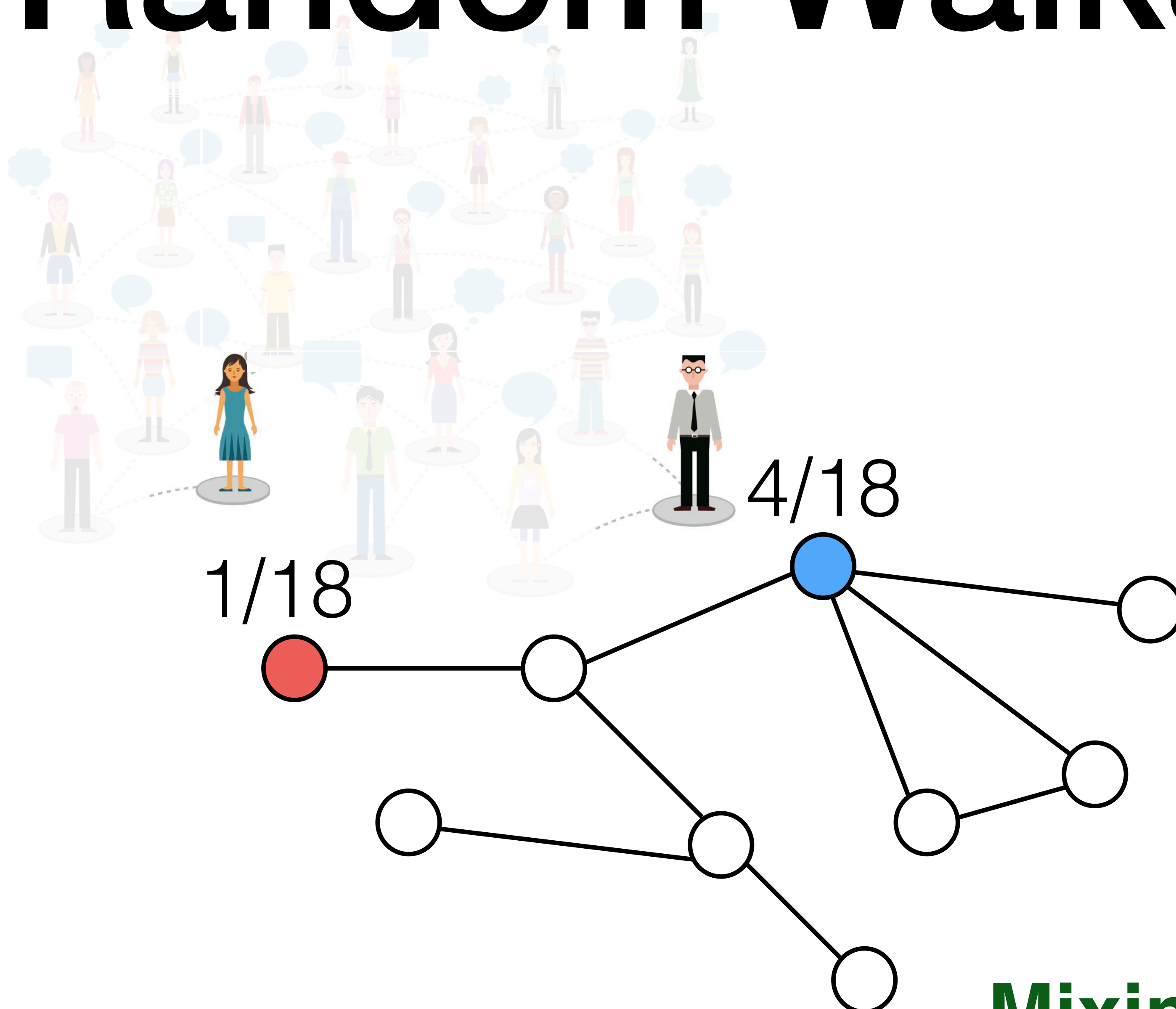
Random Walks



Mixing Time T(G)

If the process goes on for enough many steps, the random node it ends up on will be "random", *chosen with probability proportional to its degree*

Random Walks



Mixing Time T(G)

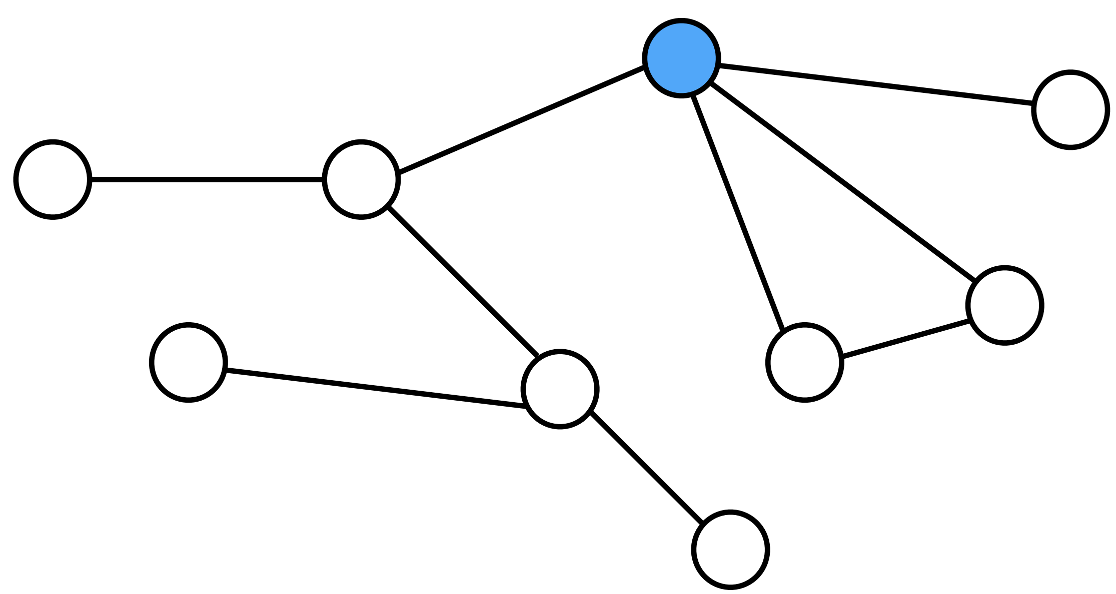
If the process goes on for enough many steps, the random node it ends up on will be “random”, *chosen with probability proportional to its degree*

A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).

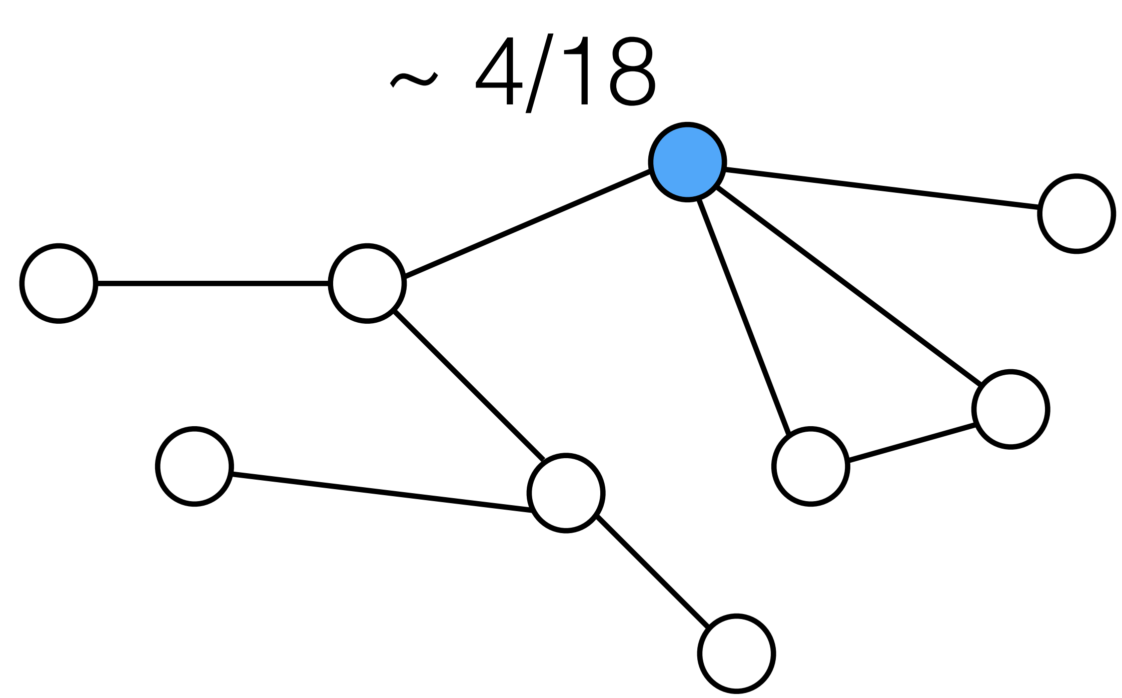
A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).



A Folklore Algorithm

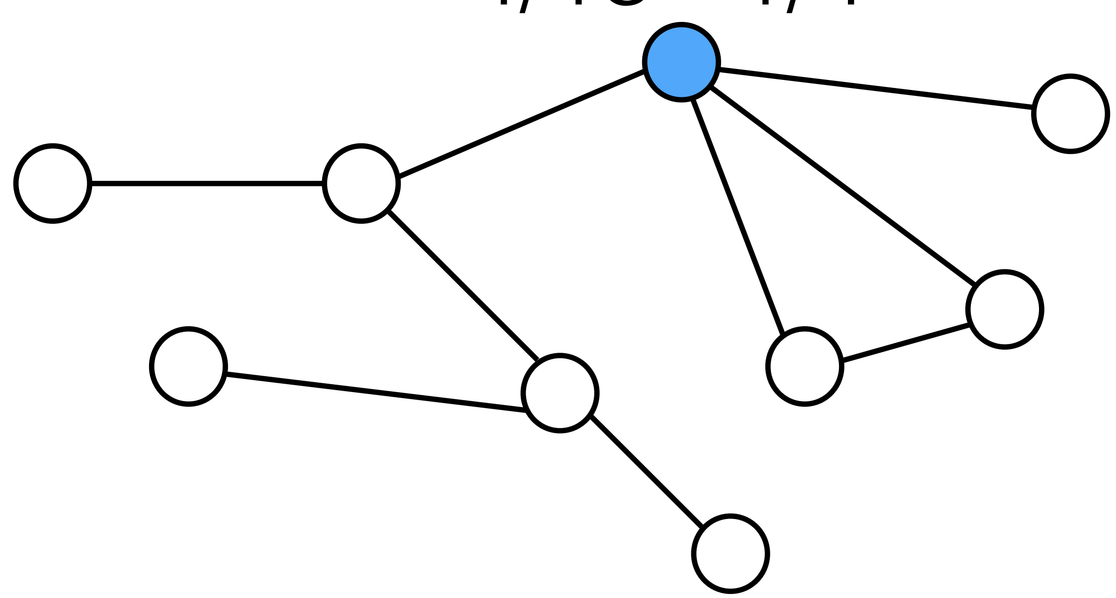
- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).



A Folklore Algorithm

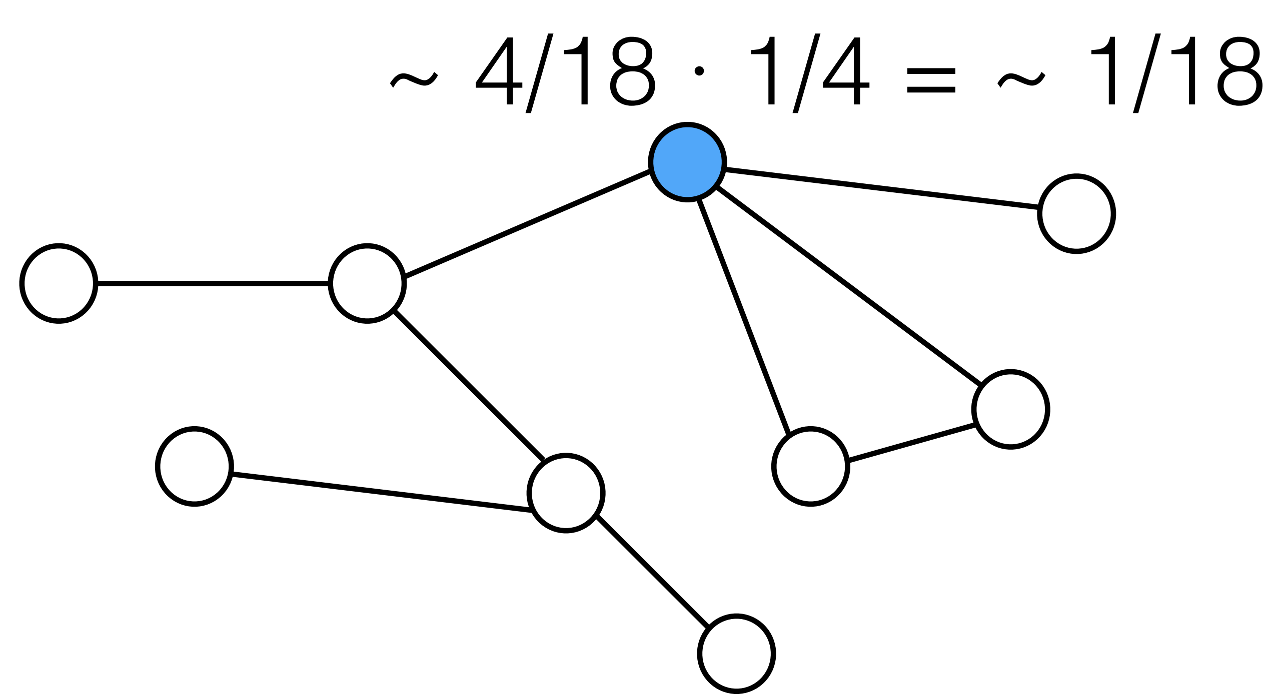
- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).

~ 4/18 · 1/4



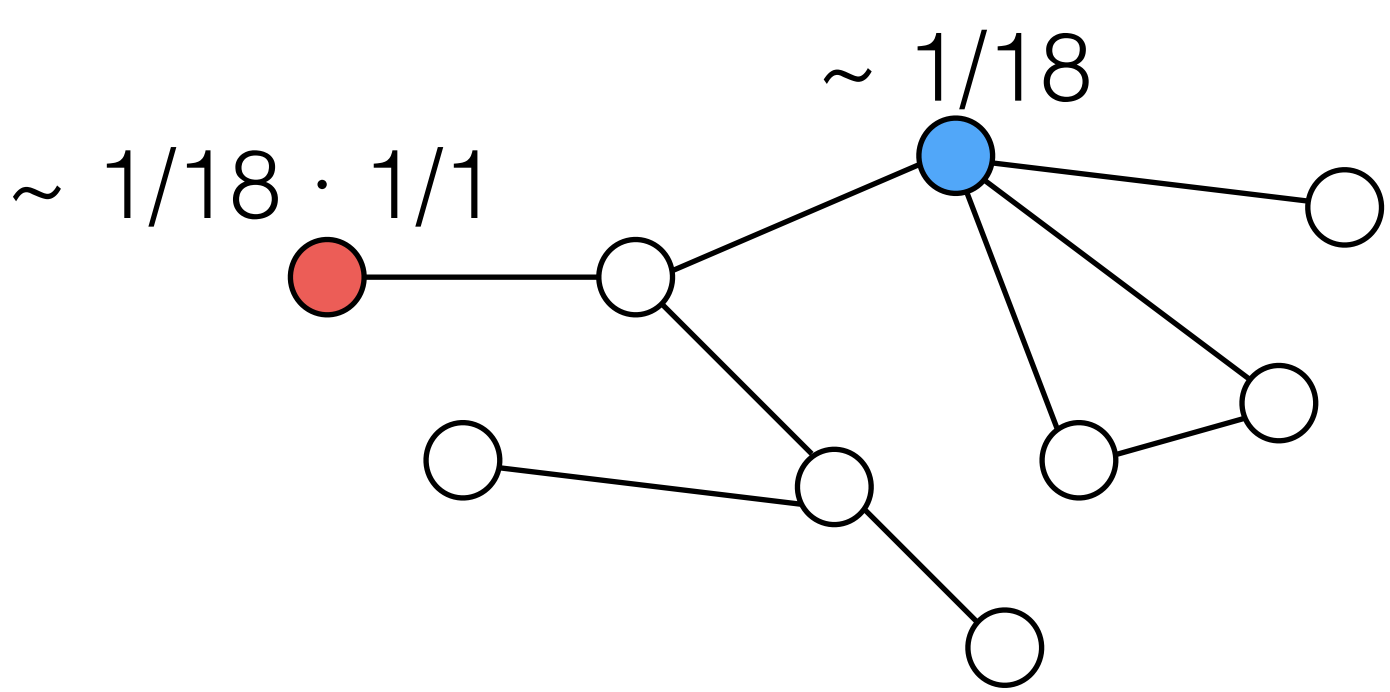
A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).



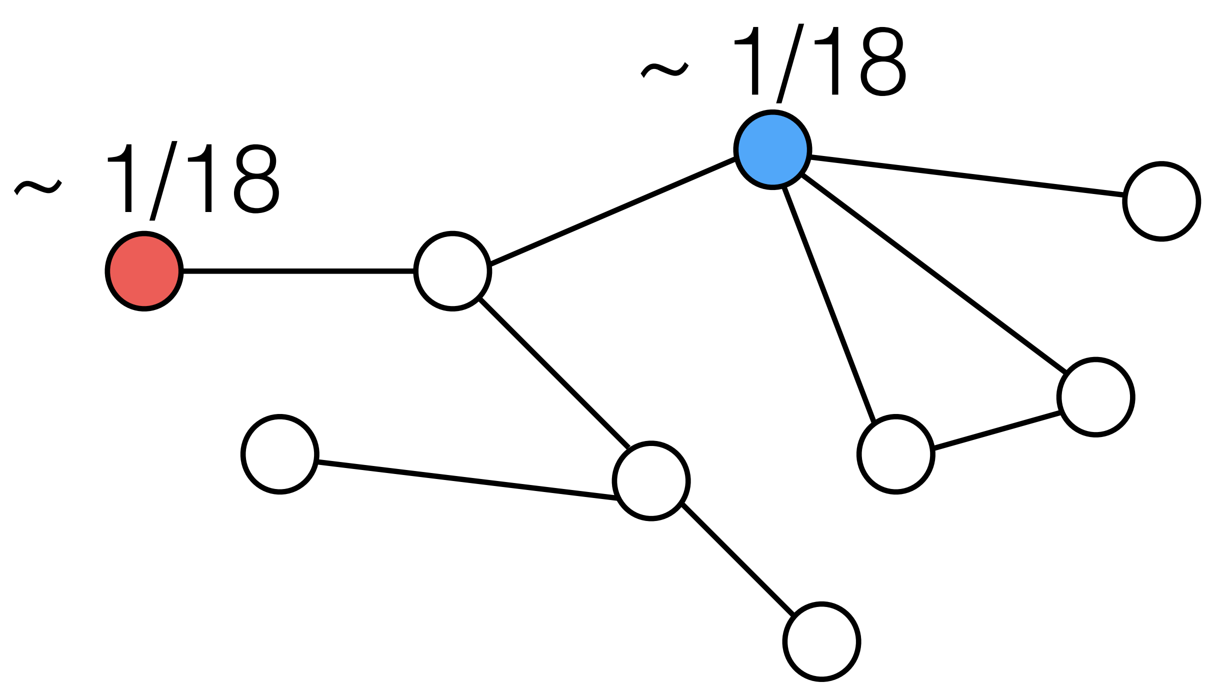
A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).



A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).



A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).

This algorithm returns a node chosen *(arbitrarily close to)* **uniformly at random**

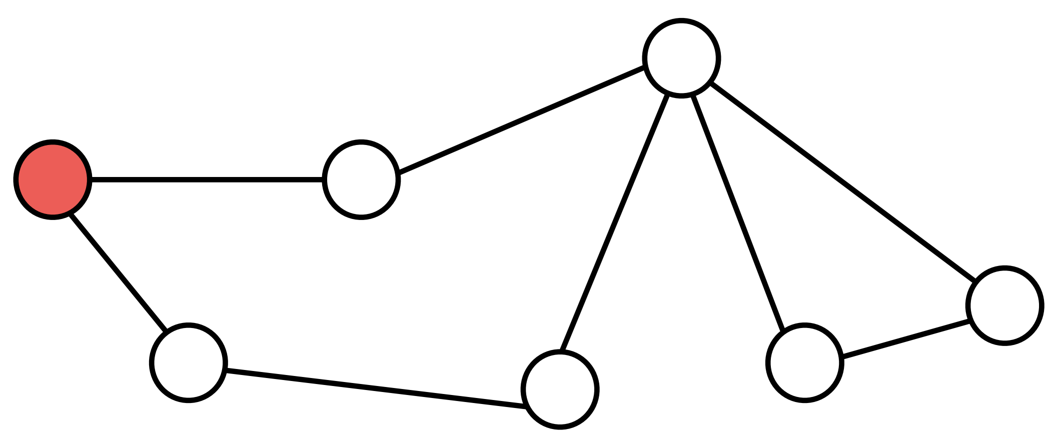
A Folklore Algorithm

- **While** True:
  - run the random walk for T(G) steps;
  - suppose it ends on the node v;
  - **return** v with probability 1/deg(v).

One can easily show that this algorithm **downloads**, with high probability, at most *O(T(G) · AvgDeg(G))* nodes from the network

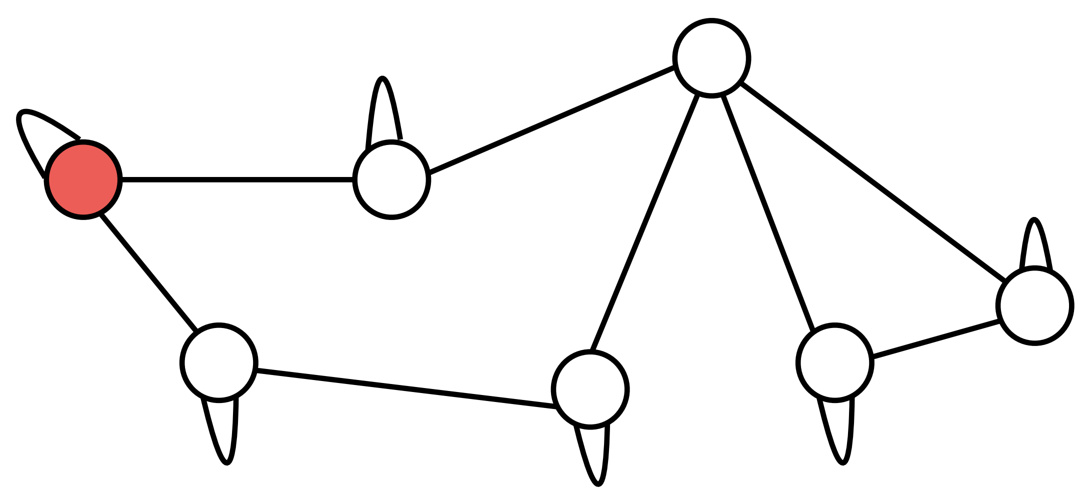
The Max-Degree Algorithm

- Let D be the max-degree of G.
- Add self-loops to G in order to make it D-regular.
- Run the random walk for $D \cdot T(G)$ steps.
- **return** the node on which it ends.



The Max-Degree Algorithm

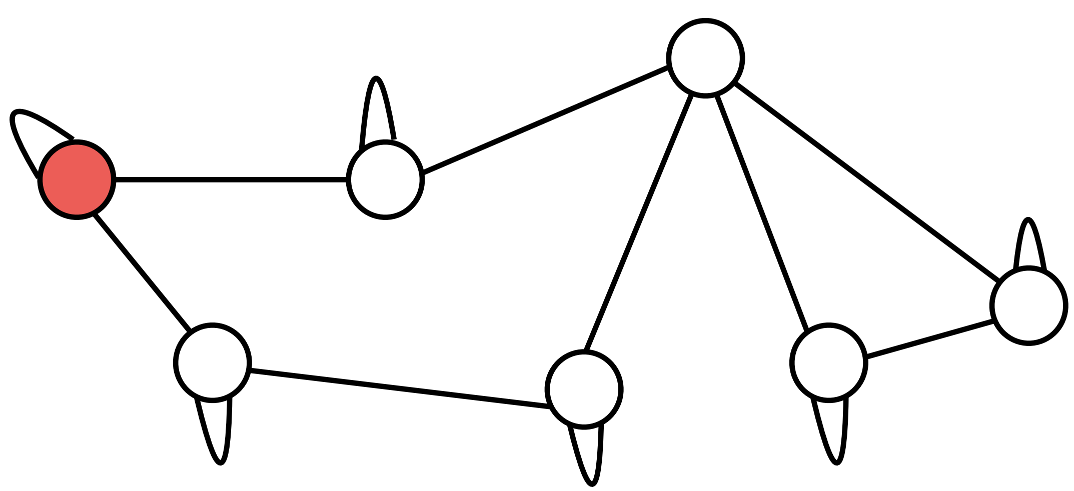
- Let D be the max-degree of G.
- Add self-loops to G in order to make it D-regular.
- Run the random walk for D · T(G) steps.
- **return** the node on which it ends.



The Max-Degree Algorithm

- Let D be the max-degree of G.
- Add self-loops to G in order to make it D-regular.
- Run the random walk for D · T(G) steps.
- **return** the node on which it ends.

Running Time: D · T(G)

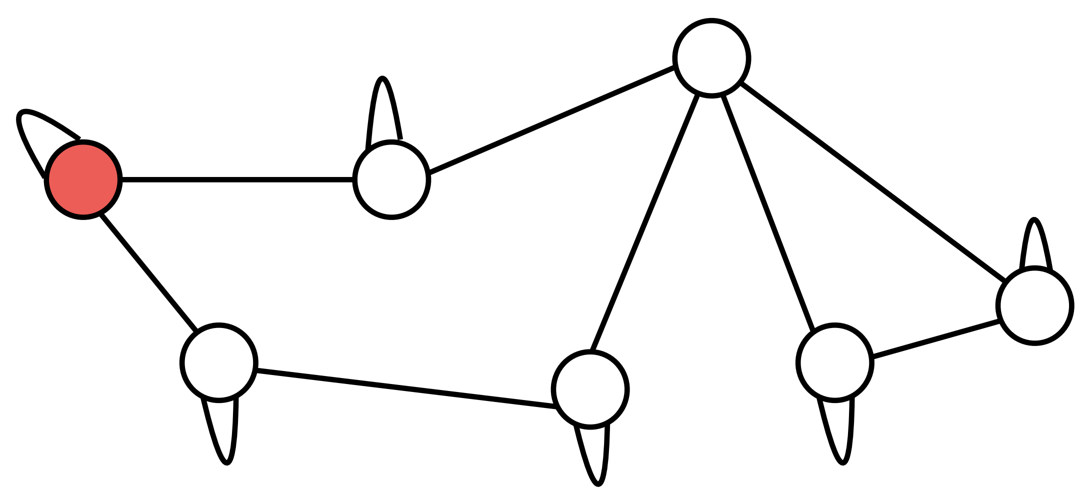


The Max-Degree Algorithm

- Let D be the max-degree of G.
- Add self-loops to G in order to make it D-regular.
- Run the random walk for D · T(G) steps.
- **return** the node on which it ends.

Running Time: D · T(G)

# of Downloaded Vertices ≤ AvgDeg(G) · T(G)



Can one do better?

- In *[C., Dasgupta, Kumar, Lattanzi, Sarlós,’16]* we analyzed various algorithms for selecting a UAR node.
- Some of them were on-par with the Folklore Algorithm, some of them were worse.

Can one do better?

- In *[C., Dasgupta, Kumar, Lattanzi, Sarlós,'16]* we analyzed various algorithms for selecting a UAR node.

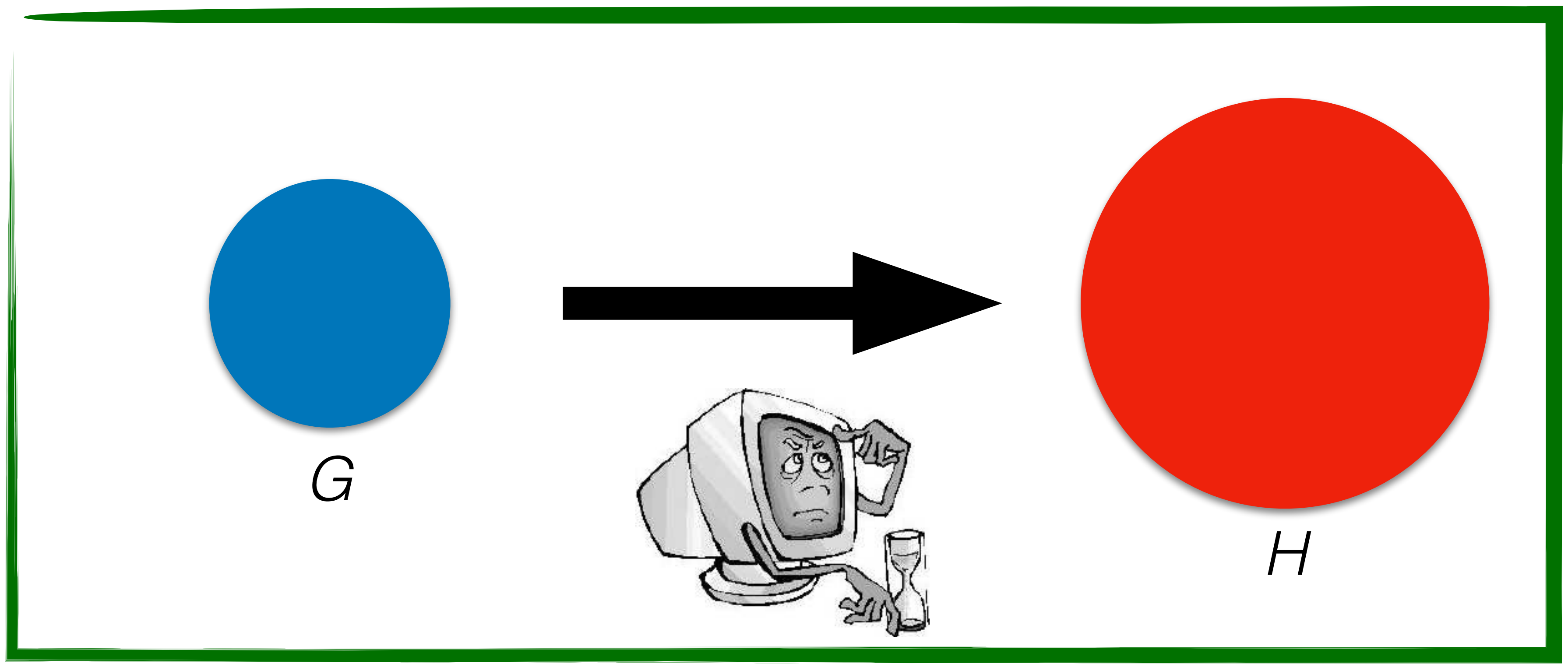
- Some of them were on-par with the Folklore Algorithm, some of them were worse.

- In *[C., Haddadan, '18],* we show that if an algorithm downloads < *o(T(G) AvgDeg(G))* nodes from the network, then it cannot return anything close to a uniform-at-random node.

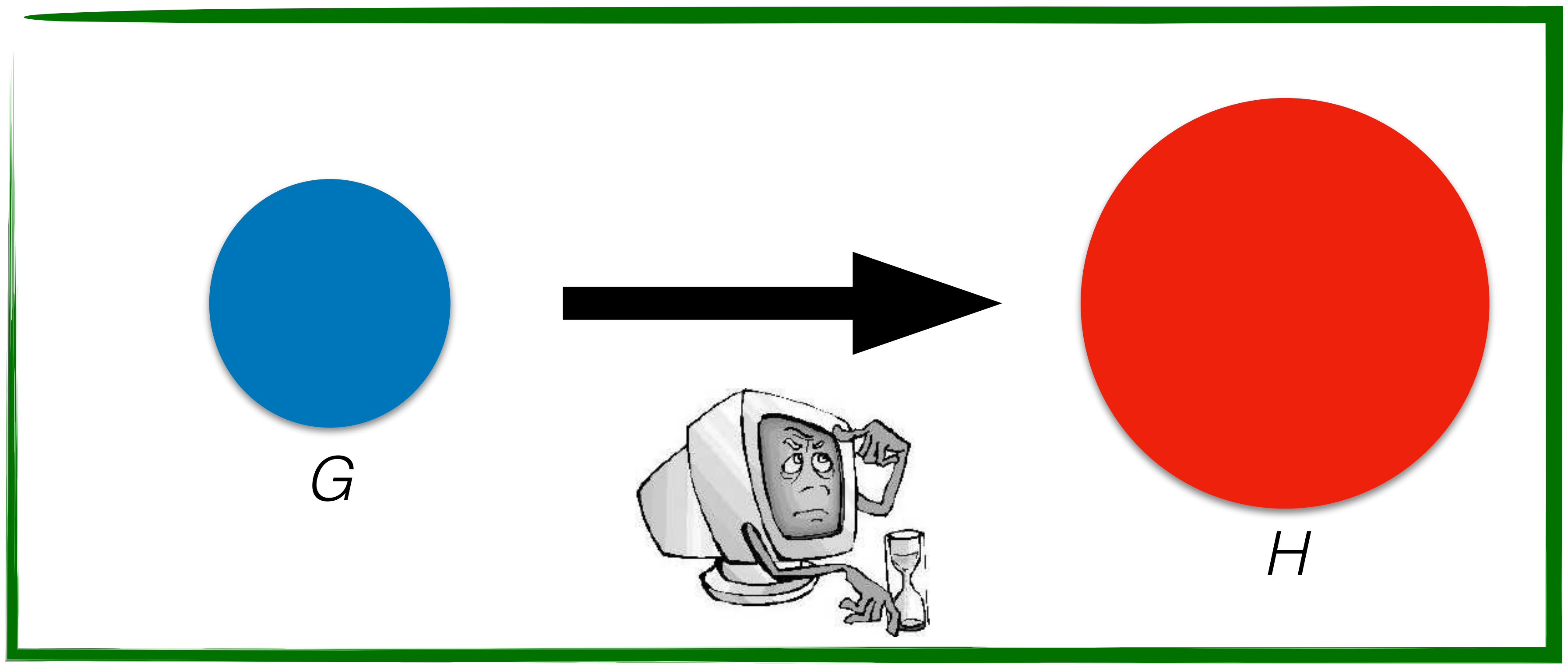
- That is, the Folklore algorithm is optimal.

Two Main Ingredients

Two Main Ingredients



Two Main Ingredients

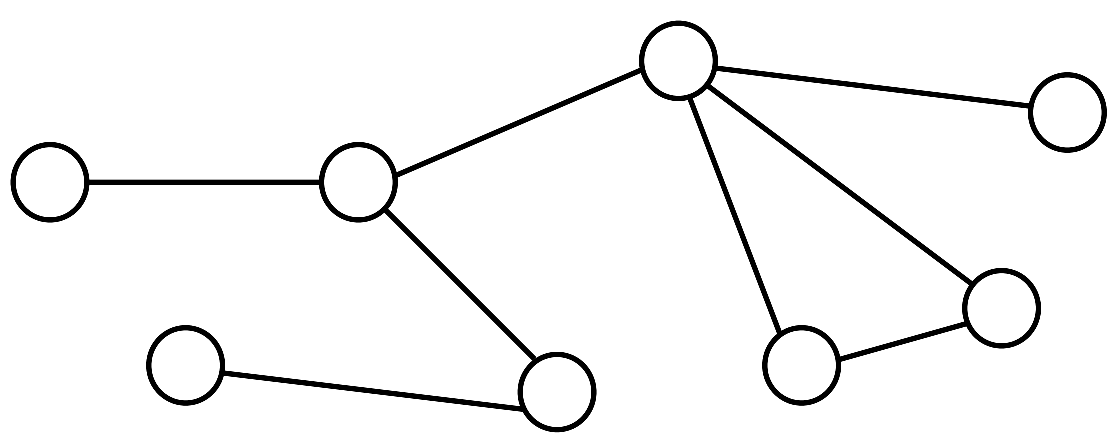


A distribution over graphs *G*

Decoration Construction

[C., Haddadan,'18]

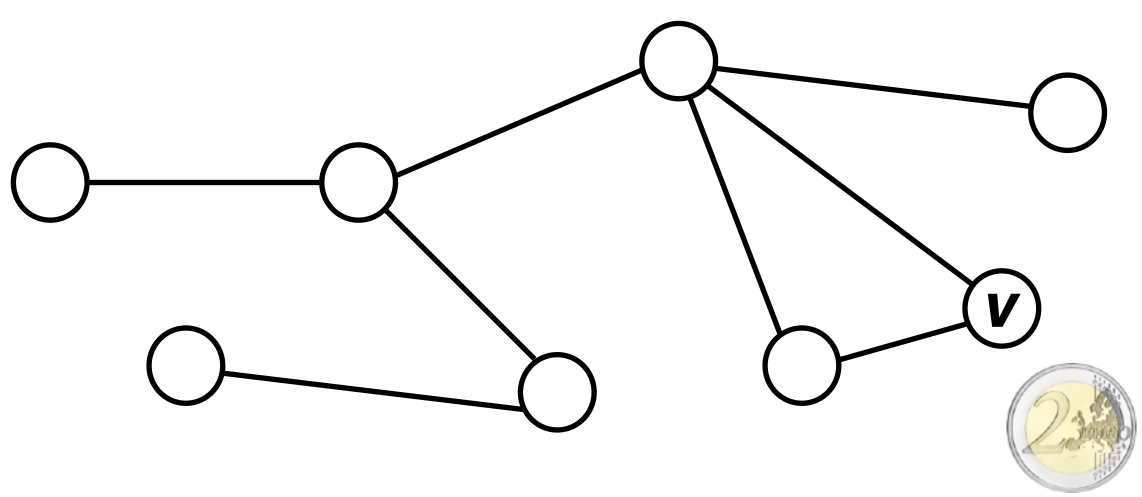
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

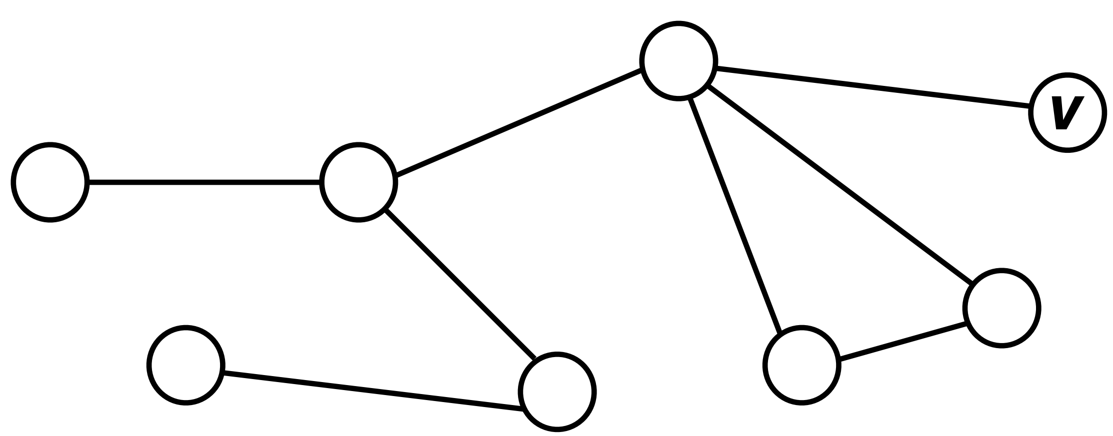
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

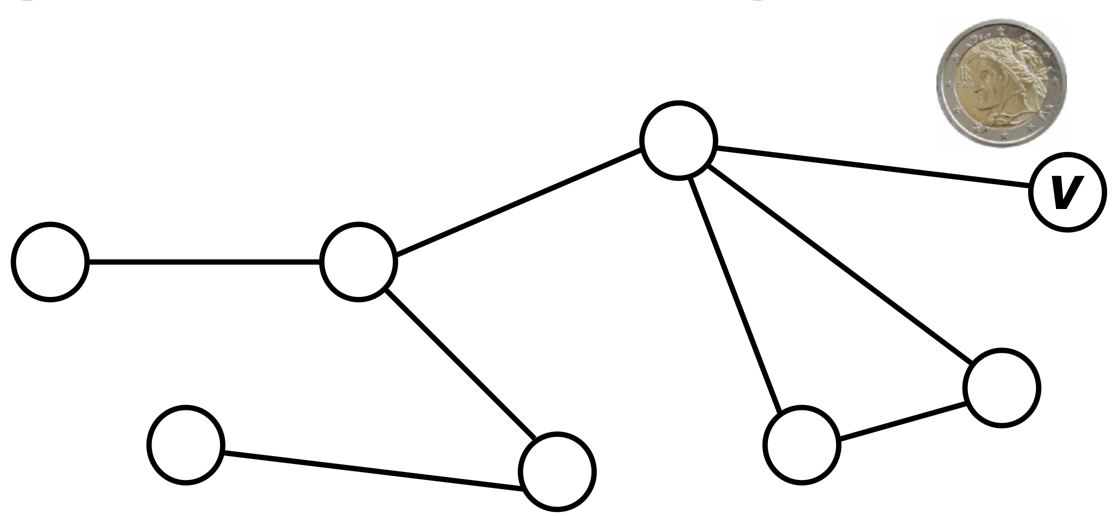
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
    - for each v in V, flip an iid coin: with probability $1/T$,
        - **mark** node v;
        - create a new node v', and cT new nodes v'_i
        - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

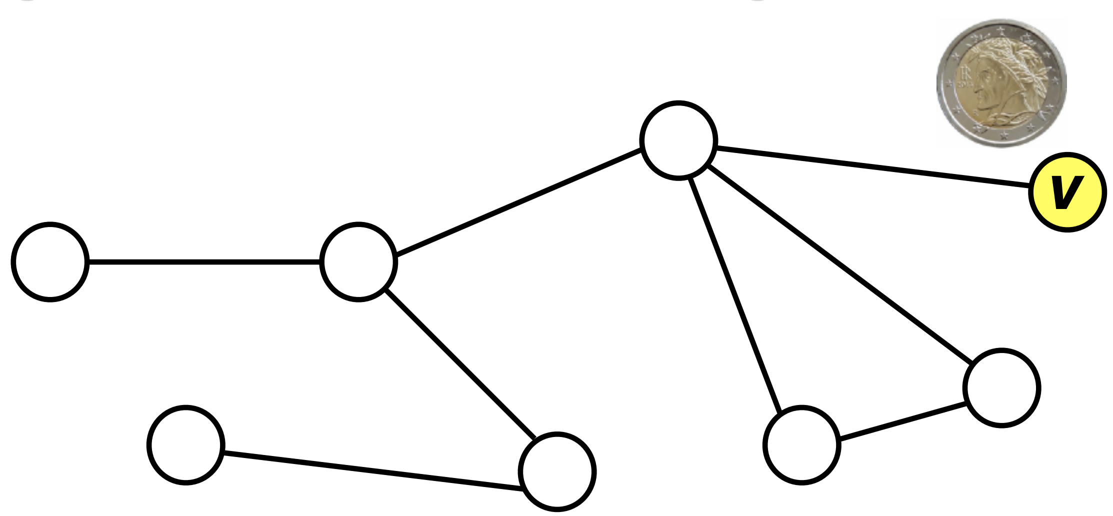
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

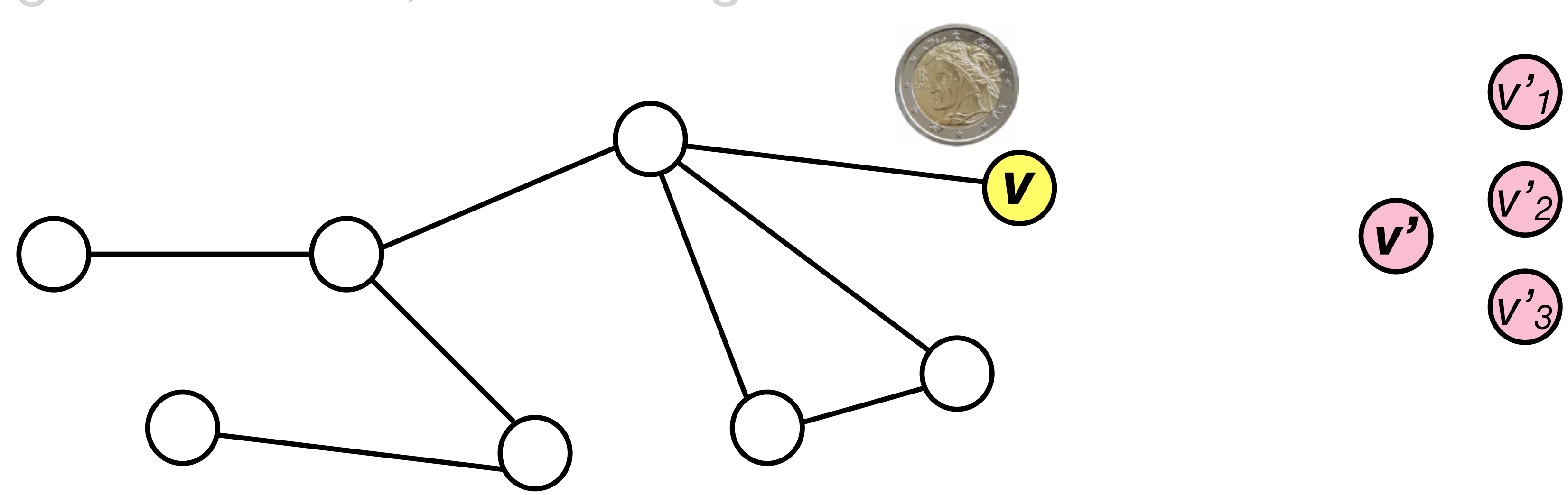
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

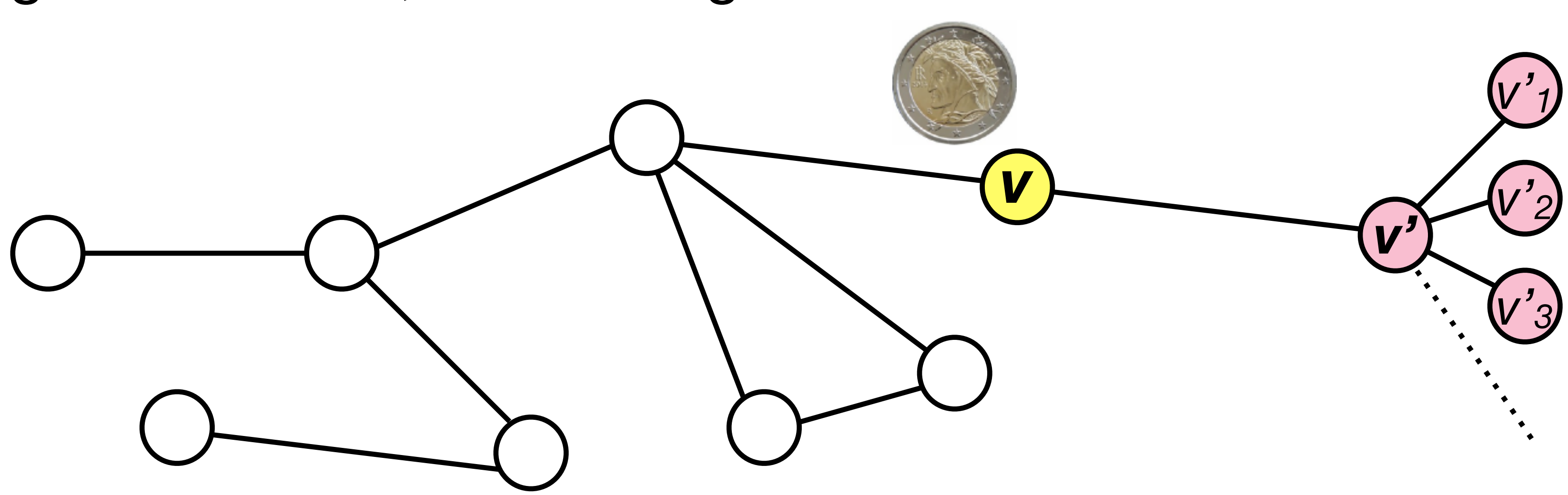
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

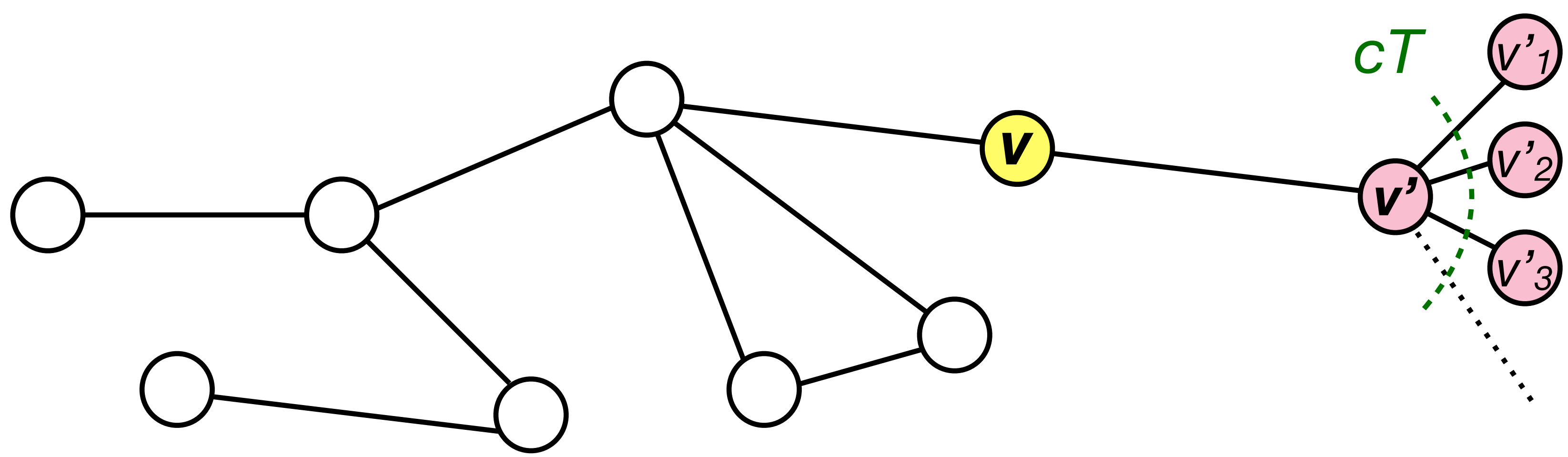
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

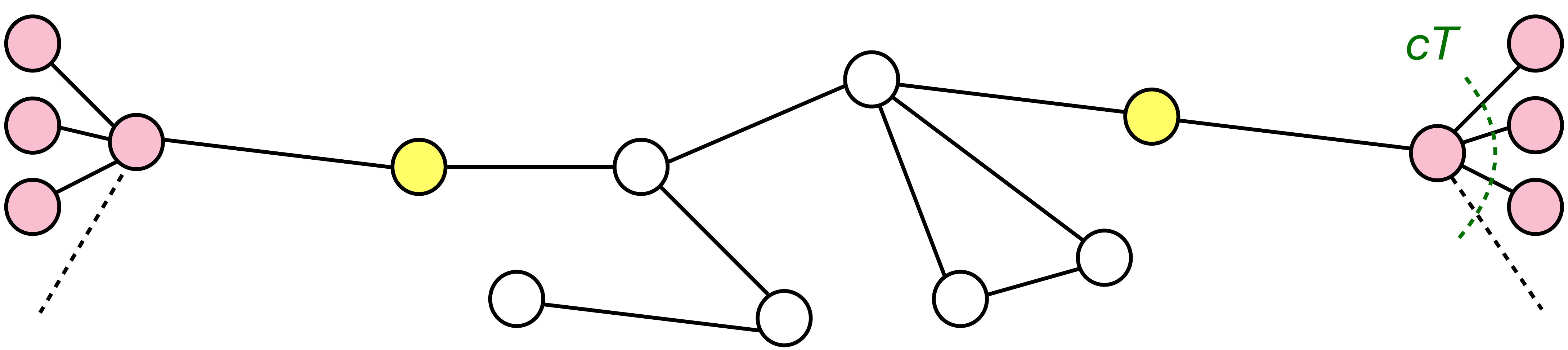
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

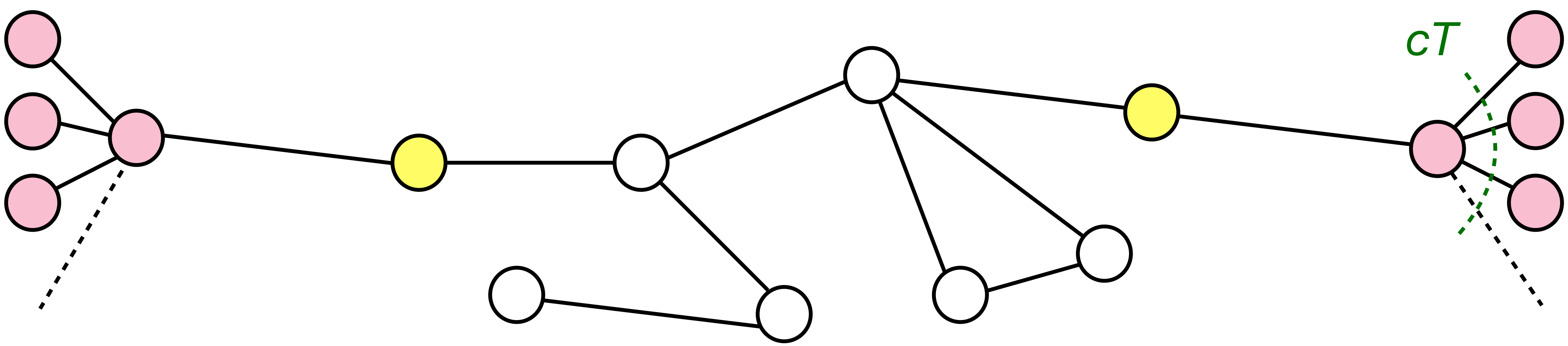
- Let $G = (V,E)$ be a graph, with mixing time T.
- The (random) decoration of G is a super-graph H of G constructed as follows:
  - for each v in V, flip an iid coin: with probability $1/T$,
    - **mark** node v;
    - create a new node v', and cT new nodes v'_i
    - add an edge from v to v', and an edge to v' to each v'_i



Decoration Construction

[C., Haddadan,'18]

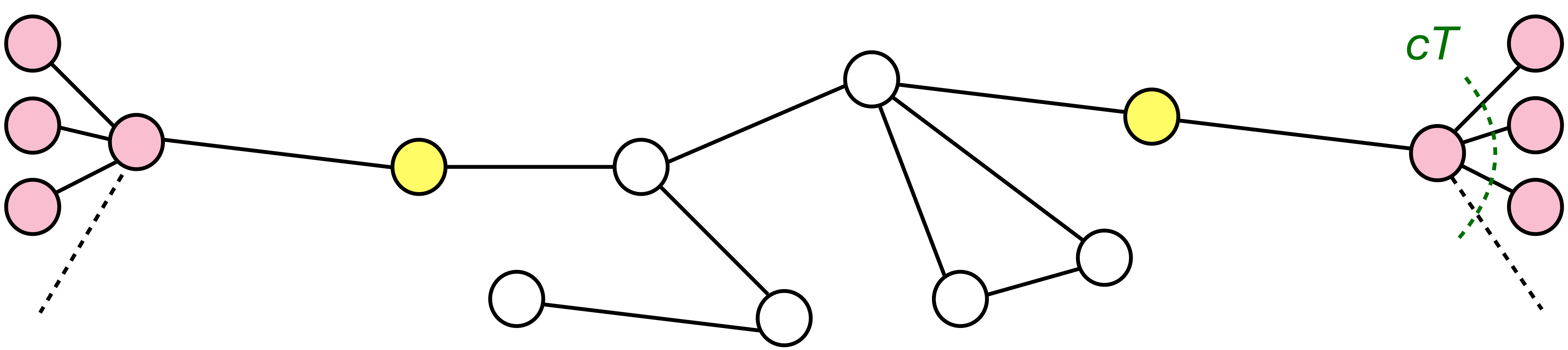
- Let $G = (V,E)$ be a graph, with mixing time $T < o(|V|)$ and average degree $d > \omega(1)$.
- Let H be a random decoration of G.



Decoration Construction

[C., Haddadan,'18]

- Let $G = (V,E)$ be a graph, with mixing time $T < o(|V|)$ and average degree $d > \omega(1)$.
- Let H be a random decoration of G.
- Then, with probability $1\text{-}o(1)$, the mixing time S of H satisfies $\alpha\, T < S < \alpha'\, T$, for constants $\alpha = \alpha(c)$ and $\alpha'=\alpha'(c)$.



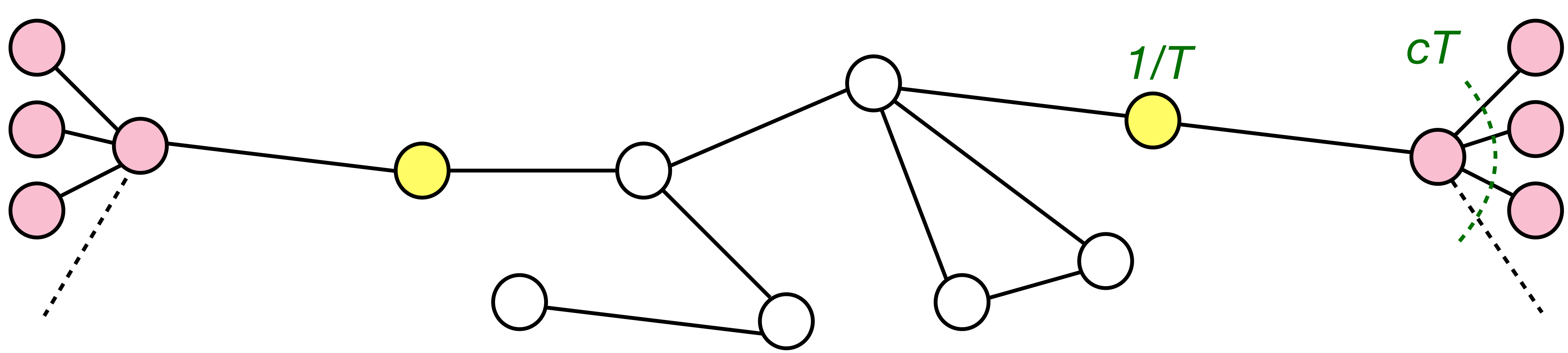
Decoration Construction

[C., Haddadan,'18]

- Let $G = (V,E)$ be a graph, with mixing time $T < o(|V|)$ and average degree $d > \omega(1)$.

- Let H be a random decoration of G.

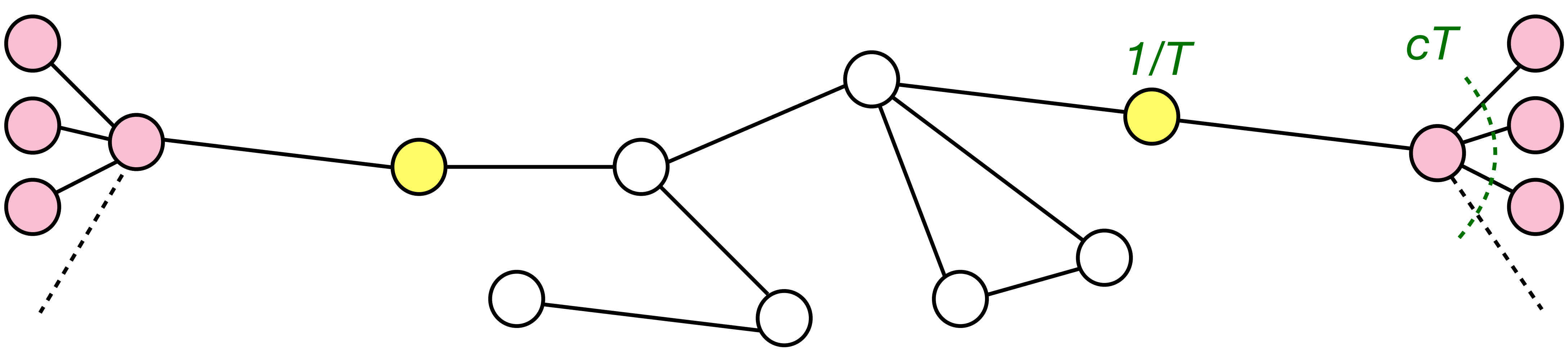
- Moreover, with probability $1 - o(1)$, the number of nodes increases by a factor of $1 + \Theta(c)$



Decoration Construction

[C., Haddadan,'18]

- Let $G = (V,E)$ be a graph, with mixing time $T < o(|V|)$ and average degree $d > \omega(1)$.
- Let H be a random decoration of G.
- Moreover, with probability $1 - o(1)$, the average degree decreases by a factor of $1 + \Theta(c)$.

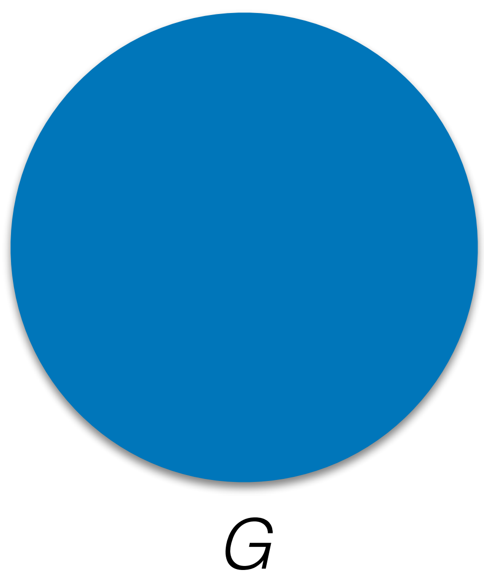


Decoration Construction

[C., Haddadan,'18]

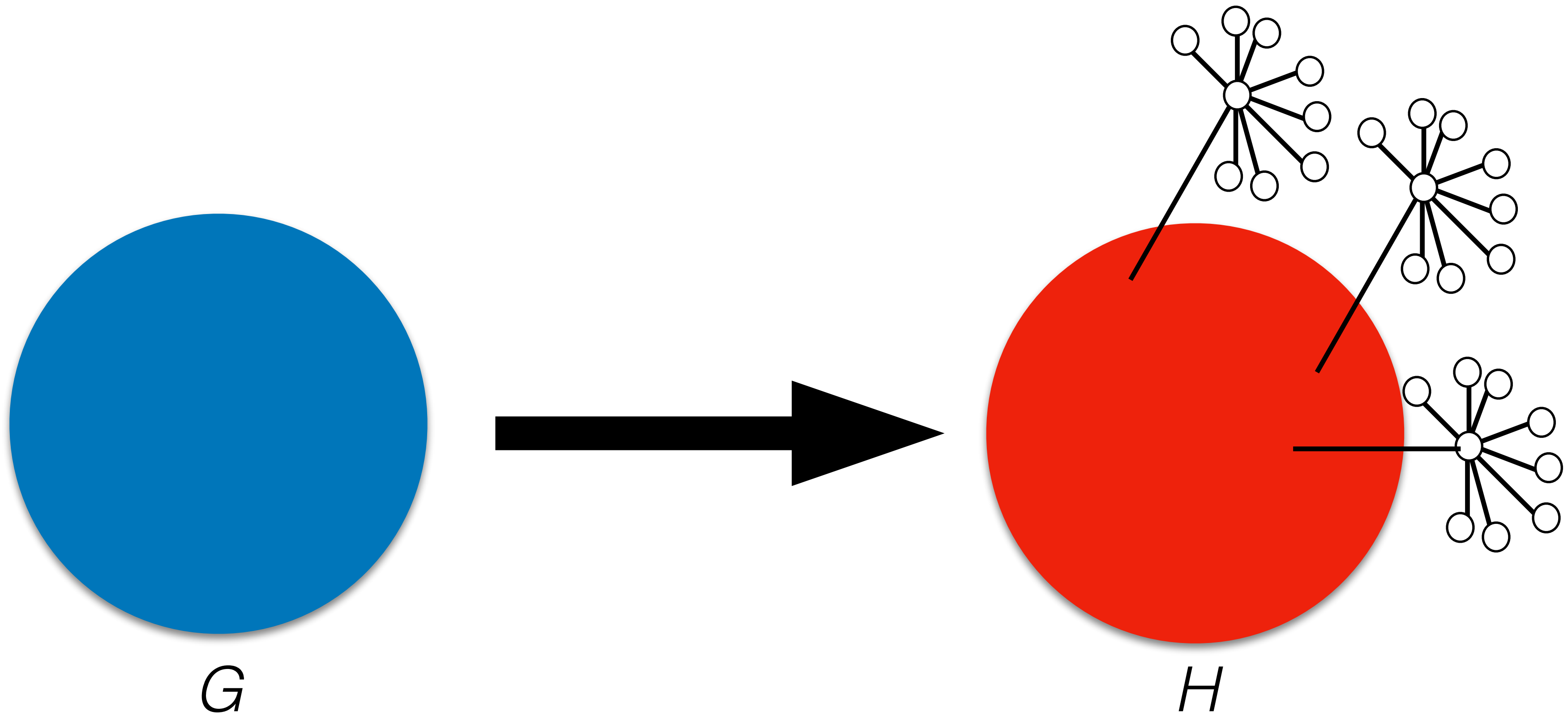
- Let $G = (V,E)$ be a graph, with mixing time $T < o(|V|)$ and average degree $d > \omega(1)$.
- Let H be a random decoration of G.
- Then, with probability $1 - o(1)$:
  - the mixing time S of H satisfies $S = \Theta(T)$,
  - the number of nodes increases by a factor of $1 + \Theta(c)$,
  - the average degree decreases by a factor of $1 + \Theta(c)$.

How to Use The Lemma



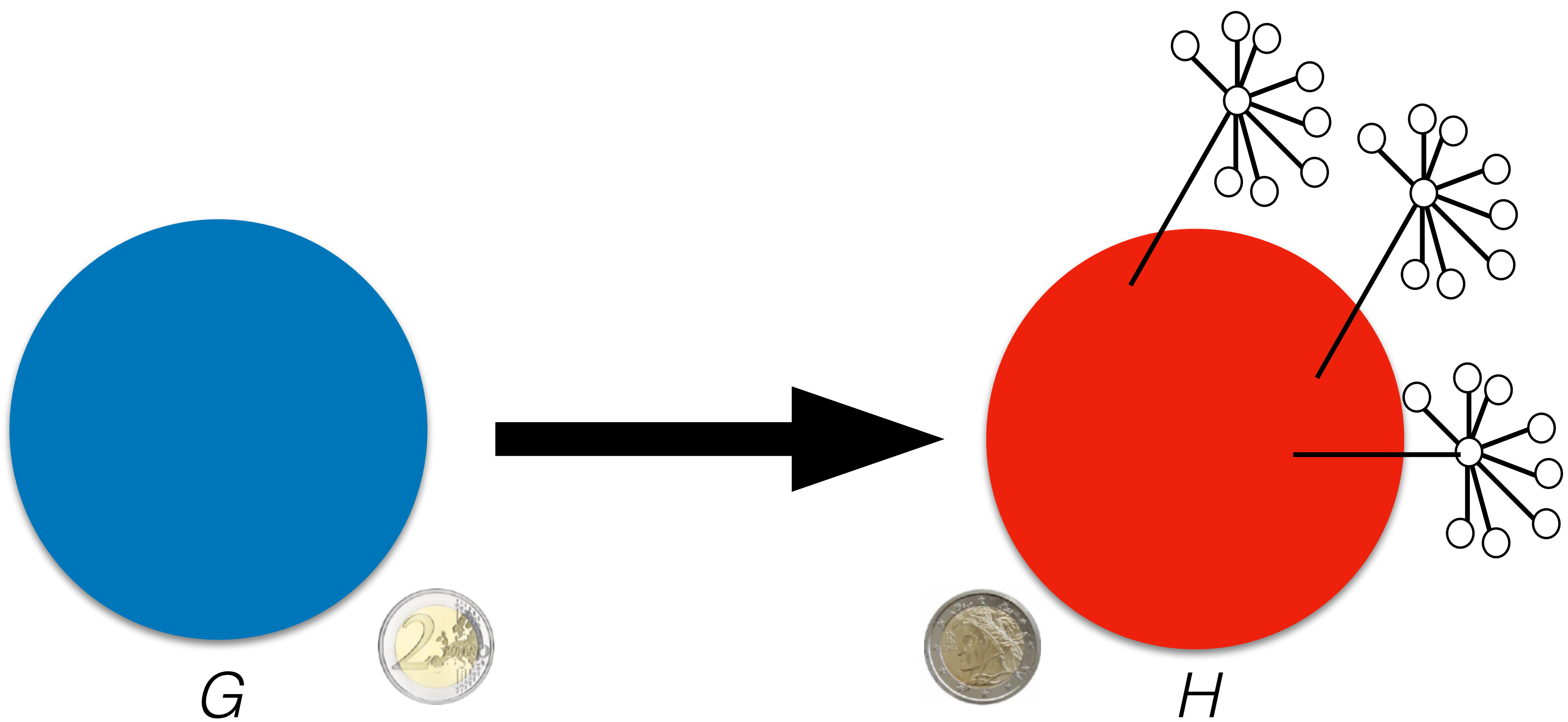
Let *G* be some (random) graph

How to Use The Lemma



Let *G* be some (random) graph, and let *H* a (random) decoration of *G*

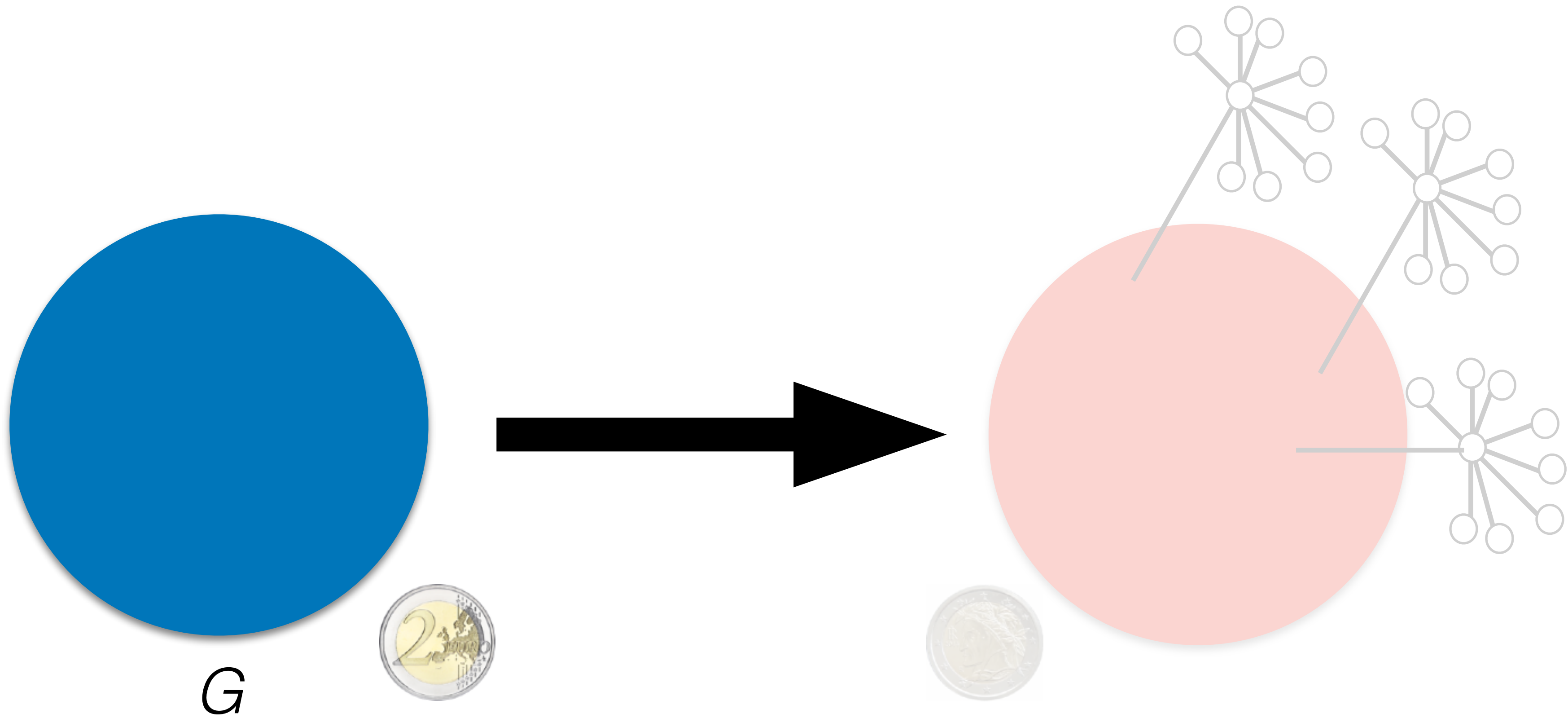
How to Use The Lemma



Let *G* be some (random) graph, and
let *H* a (random) decoration of *G*

We flip a fair coin, and run the (generic) algorithm on one of the two graphs

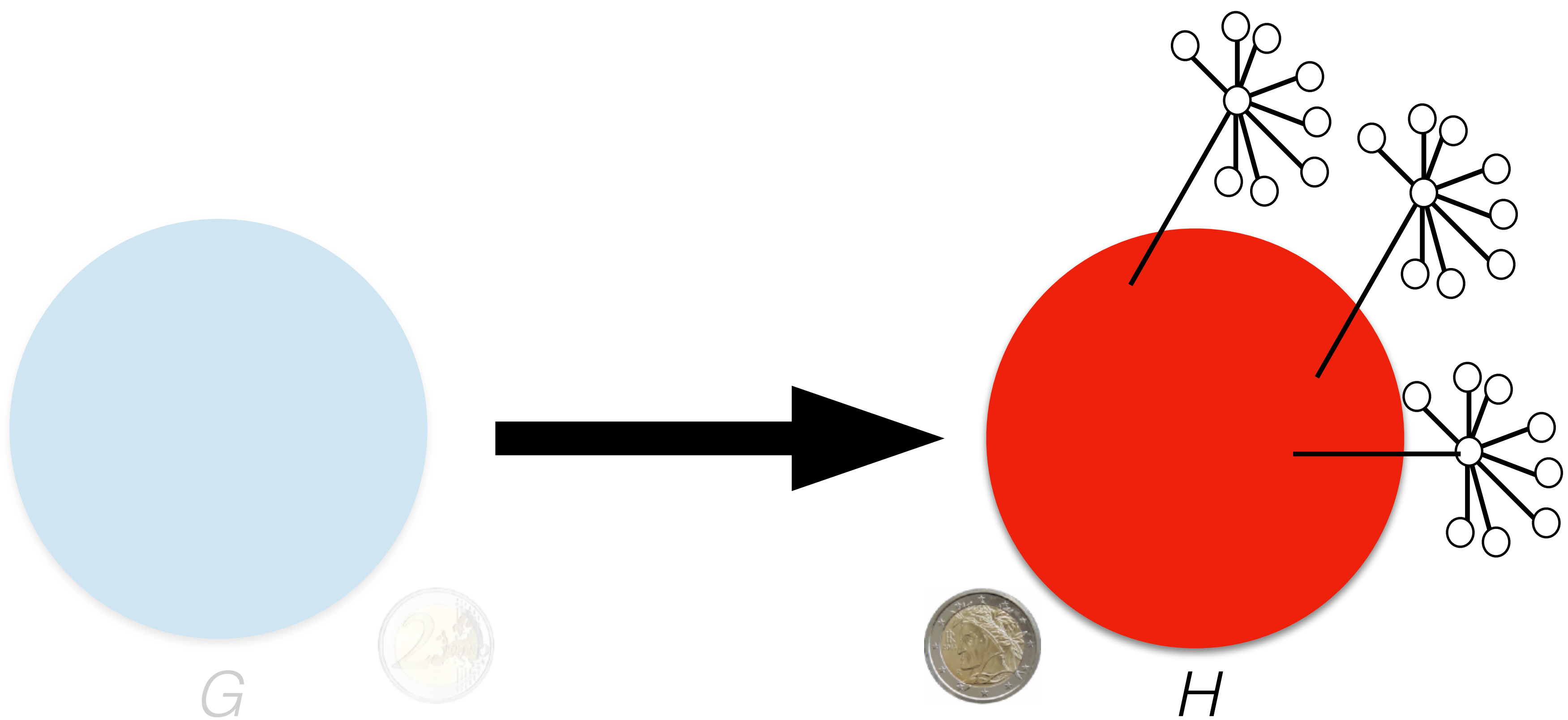
How to Use The Lemma



Let *G* be some (random) graph, and let *H* a (random) decoration of *G*

We flip a fair coin, and run the (generic) algorithm on one of the two graphs

How to Use The Lemma



Let *G* be some (random) graph, and let *H* a (random) decoration of *G*

We flip a fair coin, and run the (generic) algorithm on one of the two graphs

By showing that the algorithm cannot detect whether it is running on G or H, we prove that the algorithm cannot solve a number of problems.

The Graph G

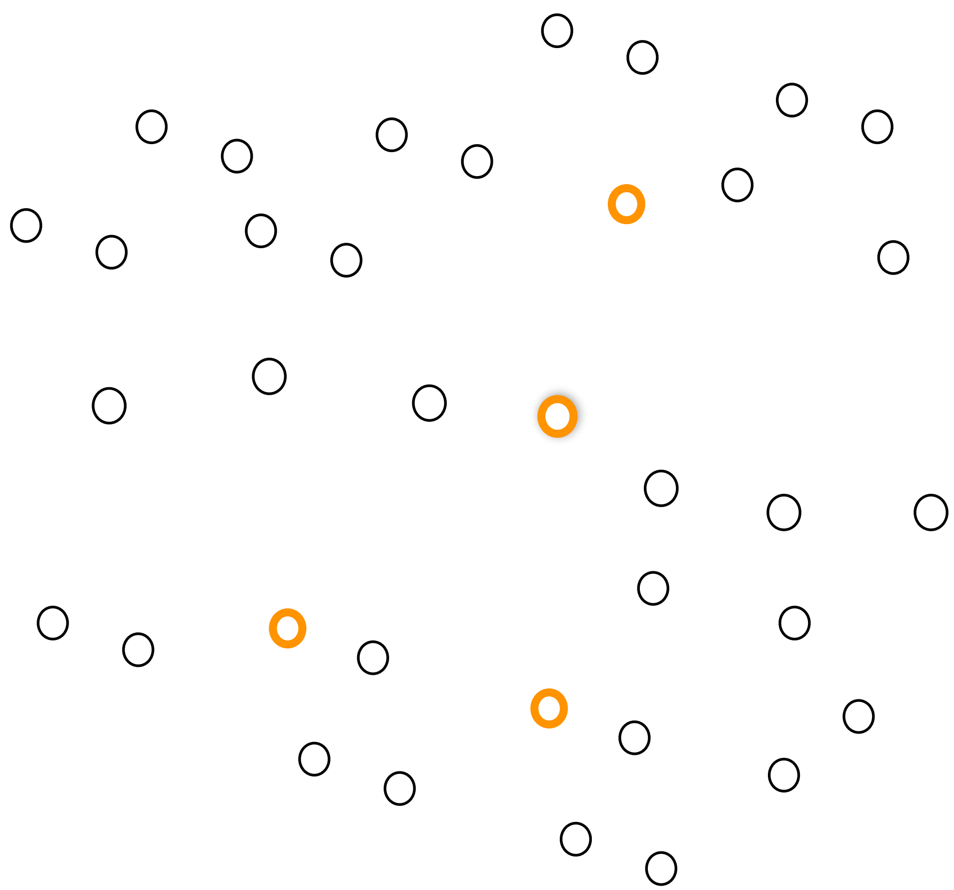
[C., Haddadan,'18]

- This approach can be made to work with *G* being a *G(n, d/n)* graph, with $d \sim \log n$

The Graph G

[C., Haddadan,'18]

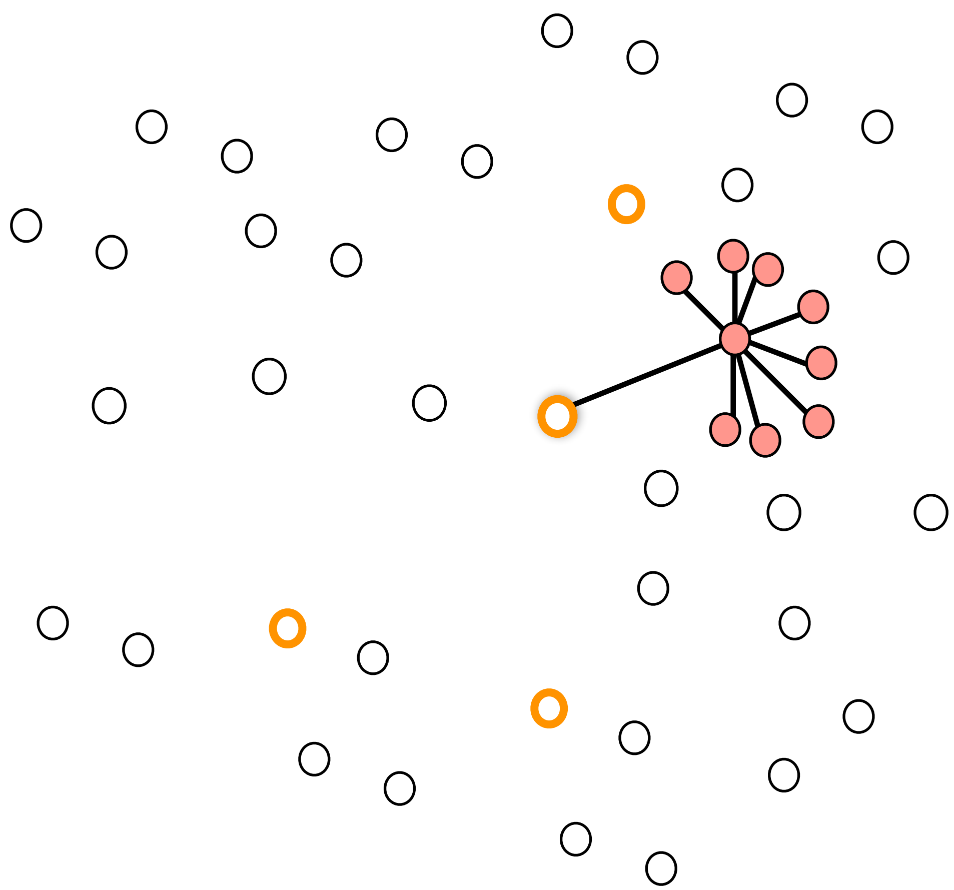
- This approach can be made to work with *G* being a *G(n, d/n)* graph, with *d* ~ *log n*



The Graph G

[C., Haddadan,'18]

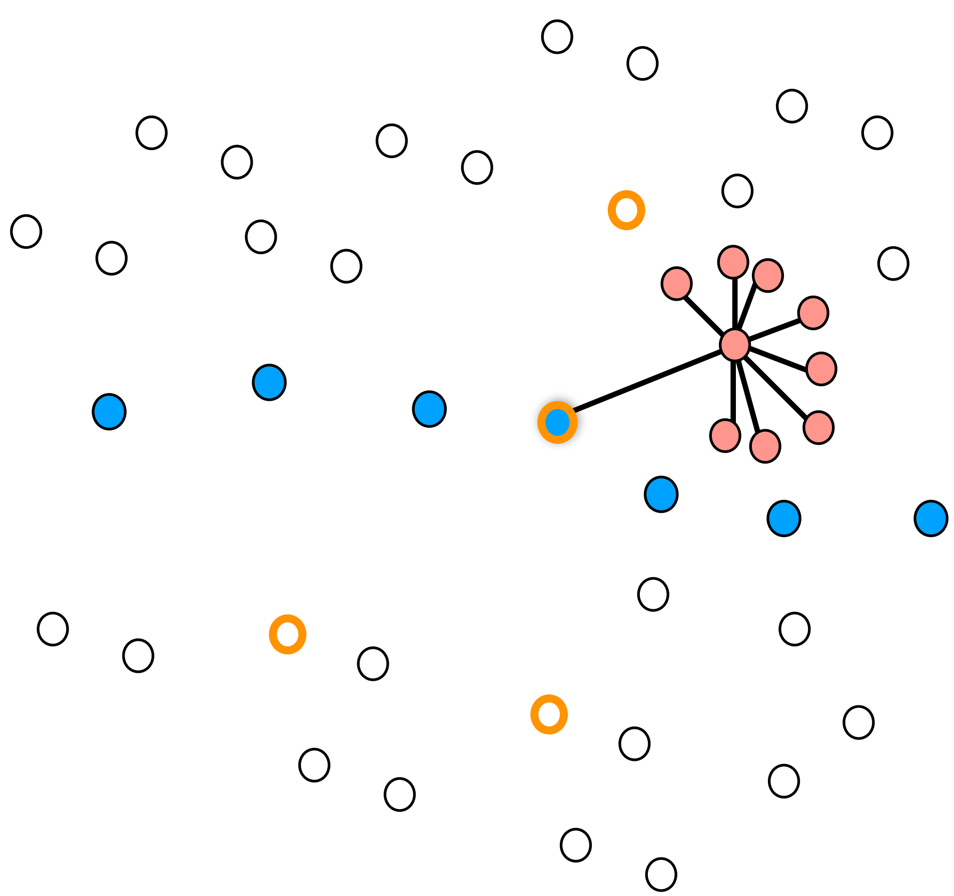
- This approach can be made to work with *G* being a *G(n, d/n)* graph, with $d \sim \log n$



The Graph G

[C., Haddadan,'18]

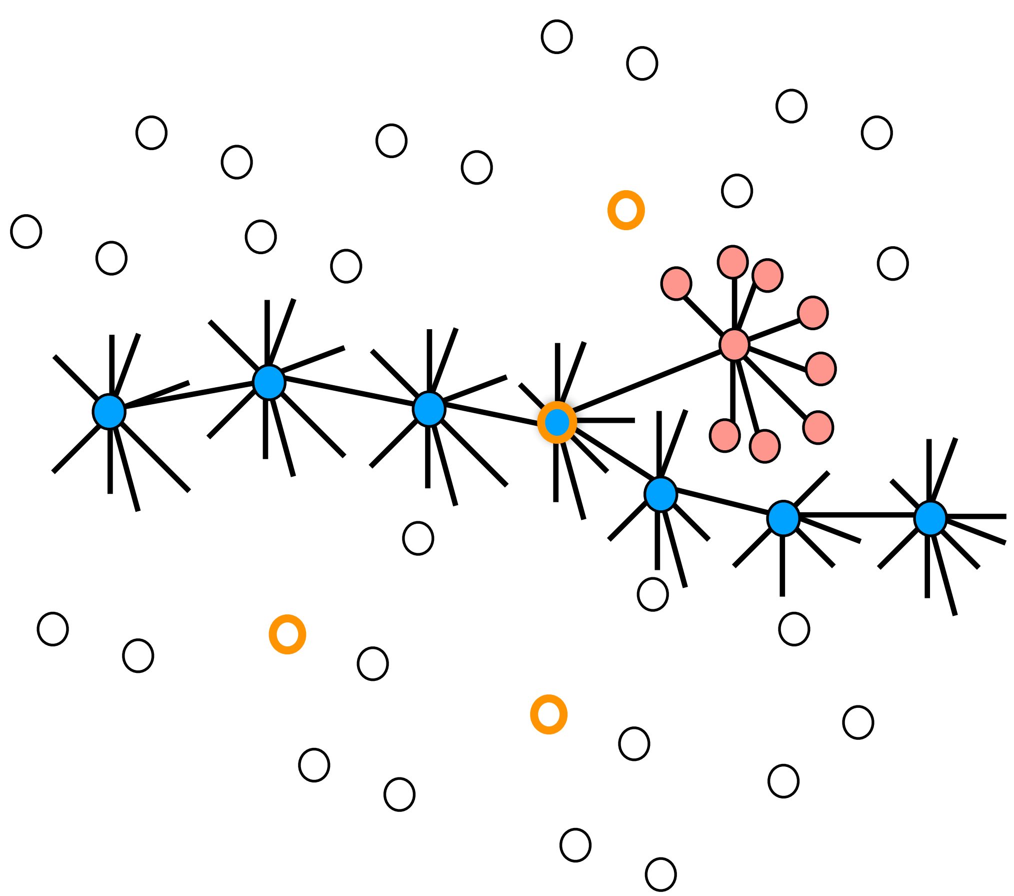
- This approach can be made to work with *G* being a *G(n, d/n)* graph, with *d* ~ *log n*



The Graph G

[C., Haddadan,'18]

- This approach can be made to work with *G* being a *G(n, d/n)* graph, with *d* ~ *log n*

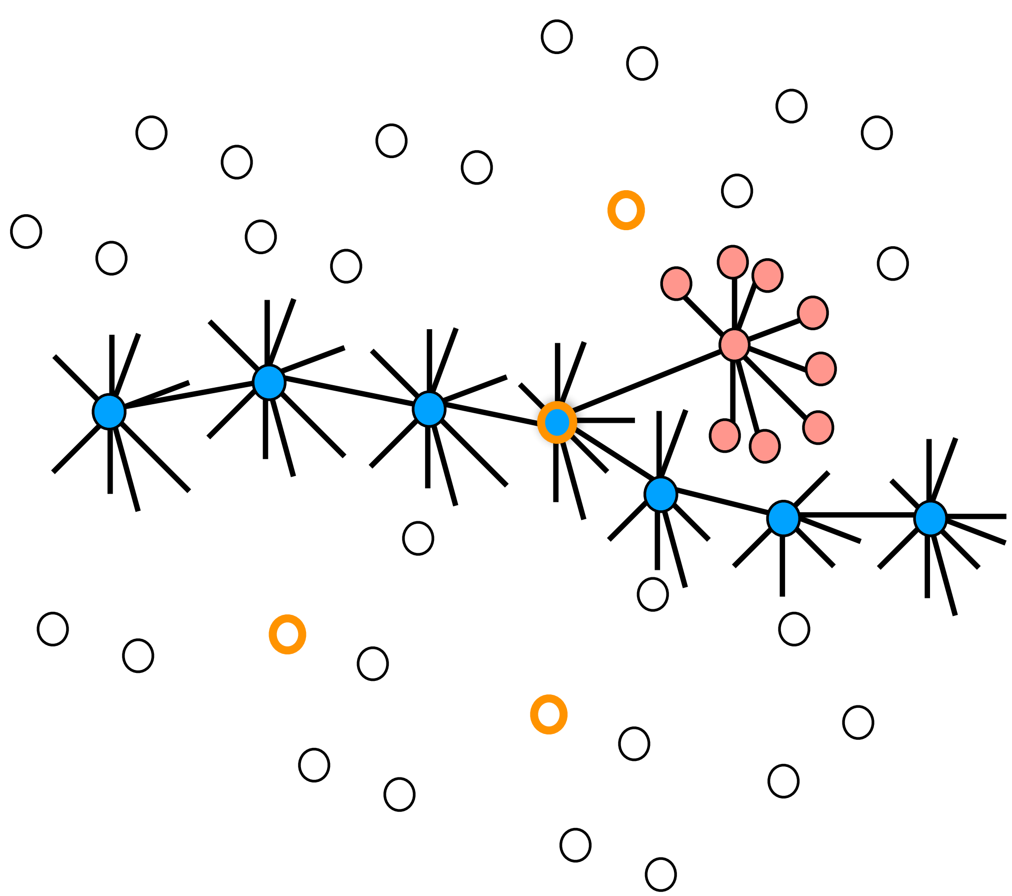


The edges towards stars will make up a 1 / (T d) fraction of the visited edges

The Graph G

[C., Haddadan,'18]

- This approach can be made to work with *G* being a *G(n, d/n)* graph, with *d ~ log n,*
  - with such a G, though, the mixing time *T* is going to be *~ log n.*

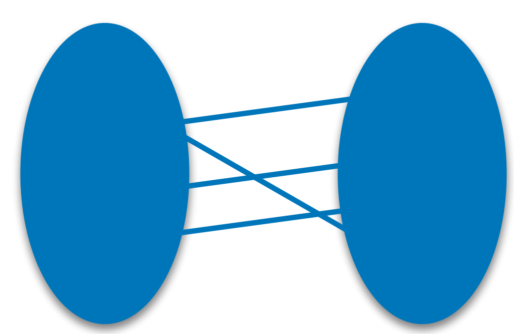


The edges towards stars will make up a 1 / (T d) fraction of the visited edges

The Graph G

[C., Haddadan,'18]

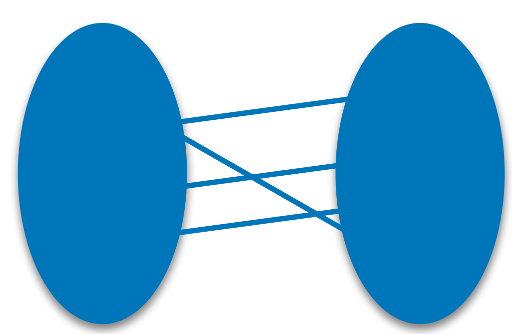
- This approach can be made to work with G being a $G(n, d/n)$ graph, with $d \sim \log n$,
  - with such a G, though, the mixing time T is going to be $\sim \log n$.
- Therefore, we pick two independent $G(n/2, p)$'s, and join them with a random matching of $< n / 2$ edges



The Graph G

[C., Haddadan,'18]

- This approach can be made to work with *G* being a *G(n, d/n)* graph, with $d \sim \log n$,
    - with such a G, though, the mixing time *T* is going to be $\sim \log n$.
- Therefore, we pick two independent *G(n/2, p)*'s, and join them with a random matching of $< n / 2$ edges,
    - the number of edges allows us to control the mixing time *T* of the resulting *G*.



The Graph G

[C., Haddadan,'18]

- Let n be a large integer. Pick T and d so that
    - $T \geq d > \omega(\log n)$, and
    - $T d^2 < o(n)$.

The Graph G

[C., Haddadan,'18]

- Let n be a large integer. Pick T and d so that
  - $T \geq d > \omega(\log n)$, and
  - $T d^2 < o(n)$.
- Then, there exists a distribution over graphs G of $\Theta(n)$ nodes, having average degree $\Theta(d)$ and mixing time $\Theta(T)$ such that***, no algorithm* accessing *o(T d)*** nodes of G can
  - return a random node of G with a distribution *o(1)-far* from the uniform one in ℓ_1 distance

The Graph G

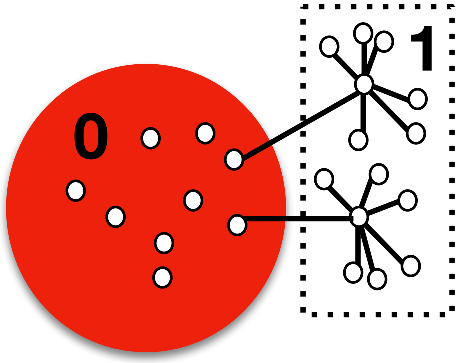
[C., Haddadan,'18]

- Let n be a large integer. Pick T and d so that
  - $T \geq d > \omega(\log n)$, and
  - $T d^2 < o(n)$.
- Then, there exists a distribution over graphs G of $\Theta(n)$ nodes, having average degree $\Theta(d)$ and mixing time $\Theta(T)$ such that***, no algorithm* accessing *o(T d)*** nodes of G can
  - return a random node of G with a distribution *o(1)-far* from the uniform one in ℓ_1 distance,
  - approximate the average value of a bounded function on the nodes to an o(1) error

The Graph G

[C., Haddadan,'18]

- Let n be a large integer. Pick T and d so that
  - $T \geq d > \omega(\log n)$, and
  - $T d^2 < o(n)$.
- Then, there exists a distribution over graphs G of $\Theta(n)$ nodes, having average degree $\Theta(d)$ and mixing time $\Theta(T)$ such that, ***no algorithm* accessing *o(T d)*** nodes of G can
  - return a random node of G with a distribution *o(1)-far* from the uniform one in ℓ_1 distance,
  - approximate the average value of a bounded function on the nodes to an o(1) error



The Graph G

[C., Haddadan,'18]

- Let n be a large integer. Pick T and d so that
  - $T \geq d > \omega(\log n)$, and
  - $T d^2 < o(n)$.
- Then, there exists a distribution over graphs G of $\Theta(n)$ nodes, having average degree $\Theta(d)$ and mixing time $\Theta(T)$ such that***, no algorithm* accessing *o(T d)*** nodes of G can
  - return a random node of G with a distribution *o(1)-far* from the uniform one in ℓ_1 distance,
  - approximate the average value of a bounded function on the nodes to an o(1) error,
  - approximate the number of nodes of G to *any given constant,*
  - approximate the average degree of G to *any given constant.*

Applications

	Upper Bound
Average of a Bounded Function	$O(\mathrm{t_{mix}}\, \mathrm{d_{avg}} \log(\delta^{-1})\epsilon^{-2})$ **Max-Degree**
Uniform Sample	$O(\mathrm{t_{mix}}\, \mathrm{d_{avg}} \log(\epsilon^{-1}))$ **Max-Degree/Rejection-Sampling**

Applications

	Upper Bound	Lower Bound
Average of a Bounded Function	$O(\mathrm{t}_{\mathrm{mix}}\, \mathrm{d}_{\mathrm{avg}} \log(\delta^{-1})\epsilon^{-2})$ **Max-Degree**	$\Omega(\mathrm{t}_{\mathrm{mix}}\, \mathrm{d}_{\mathrm{avg}} \log(\delta^{-1})\epsilon^{-2})$
Uniform Sample	$O(\mathrm{t}_{\mathrm{mix}}\, \mathrm{d}_{\mathrm{avg}} \log(\epsilon^{-1}))$ **Max-Degree/Rejection-Sampling**	$\Omega(\mathrm{t}_{\mathrm{mix}}\, \mathrm{d}_{\mathrm{avg}})$

Applications

	Upper Bound	Lower Bound
Average of a Bounded Function	$O(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\delta^{-1})\epsilon^{-2})$ **Max-Degree**	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\delta^{-1})\epsilon^{-2})$
Uniform Sample	$O(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\epsilon^{-1}))$ **Max-Degree/Rejection-Sampling**	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$
Number of Vertices	$O(\mathrm{t_{mix}}\max\{\mathrm{d_{avg}}, \lvert\Pi^1\rvert_2^{-1}\}\log(\delta^{-1})\log(\epsilon^{-1})\epsilon^{-2})$ **[Katzir et al.]**	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$

Open Questions

- What is the minimum number of *node queries* to approximate the number of nodes of *G?*
- Can the lower bound, and/or the algorithm of *[Katzir et al]*, be improved?

Open Questions

- In *[C., Dasgupta, Kumar, Lattanzi, Sarlós,'16]* we also studied the number of node accesses to return a node with probability proportional to some power of its degree.

- Can one obtain tight lower and upper bounds for this problem?