Generation and Counting

Brendan McKay
Australian National University

and various colleagues to be mentioned or not...

Exhaustive generation

Lots of people want to generate exhaustive lists of combinatorial objects.

Tens of thousands of examples are published, involving many fields of science.

Usually, but not always, there is a concept of equivalent objects, and it is desired to obtain only one member of each equivalence class.

We will focus on graphs.

In 1974 it took 6 hours to generate all of the 274,668 graphs on 9 vertices (Baker, Dewdeny, Szilard). Now it takes 0.1 seconds.

Exhaustive generation

Lots of people want to generate exhaustive lists of combinatorial objects.

Tens of thousands of examples are published, involving many fields of science.

Usually, but not always, there is a concept of equivalent objects, and it is desired to obtain only one member of each equivalence class.

We will focus on graphs.

In 1974 it took 6 hours to generate all of the 274,668 graphs on 9 vertices (Baker, Dewdeny, Szilard). Now it takes 0.1 seconds.

It is practical to generate all 50,502,031,367,952 graphs on 13 vertices and plausible to generate all 29,054,155,657,235,488 graphs on 14 vertices.

Recursive generation

A recursive construction of a class of graphs consists of

1. a set of irreducible graphs in the class, and
2. a set of expansions that can be performed on graphs in the class,

such that each graph in the class can be constructed from an irreducible graph via a sequence of expansions while staying within the class.

The reverse of an expansion is a reduction.

Recursive generation

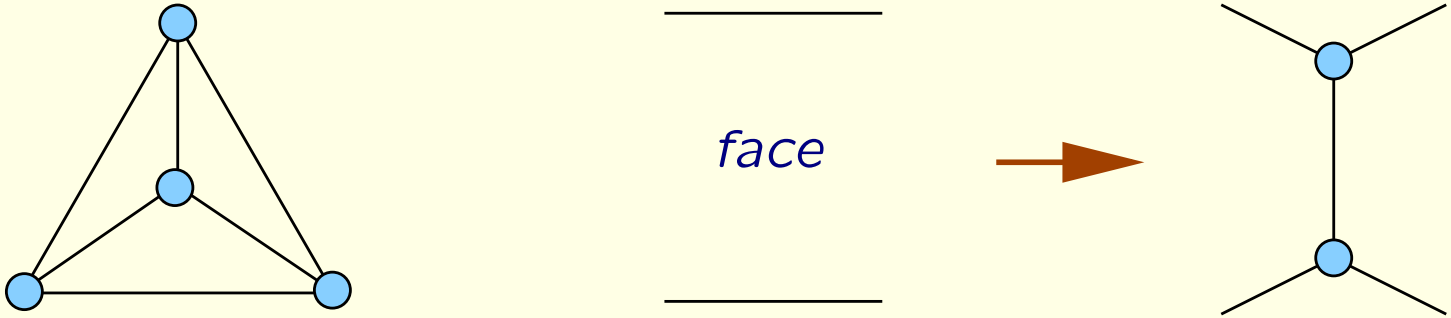
A recursive construction of a class of graphs consists of

1. a set of irreducible graphs in the class, and
2. a set of expansions that can be performed on graphs in the class,

such that each graph in the class can be constructed from an irreducible graph via a sequence of expansions while staying within the class.

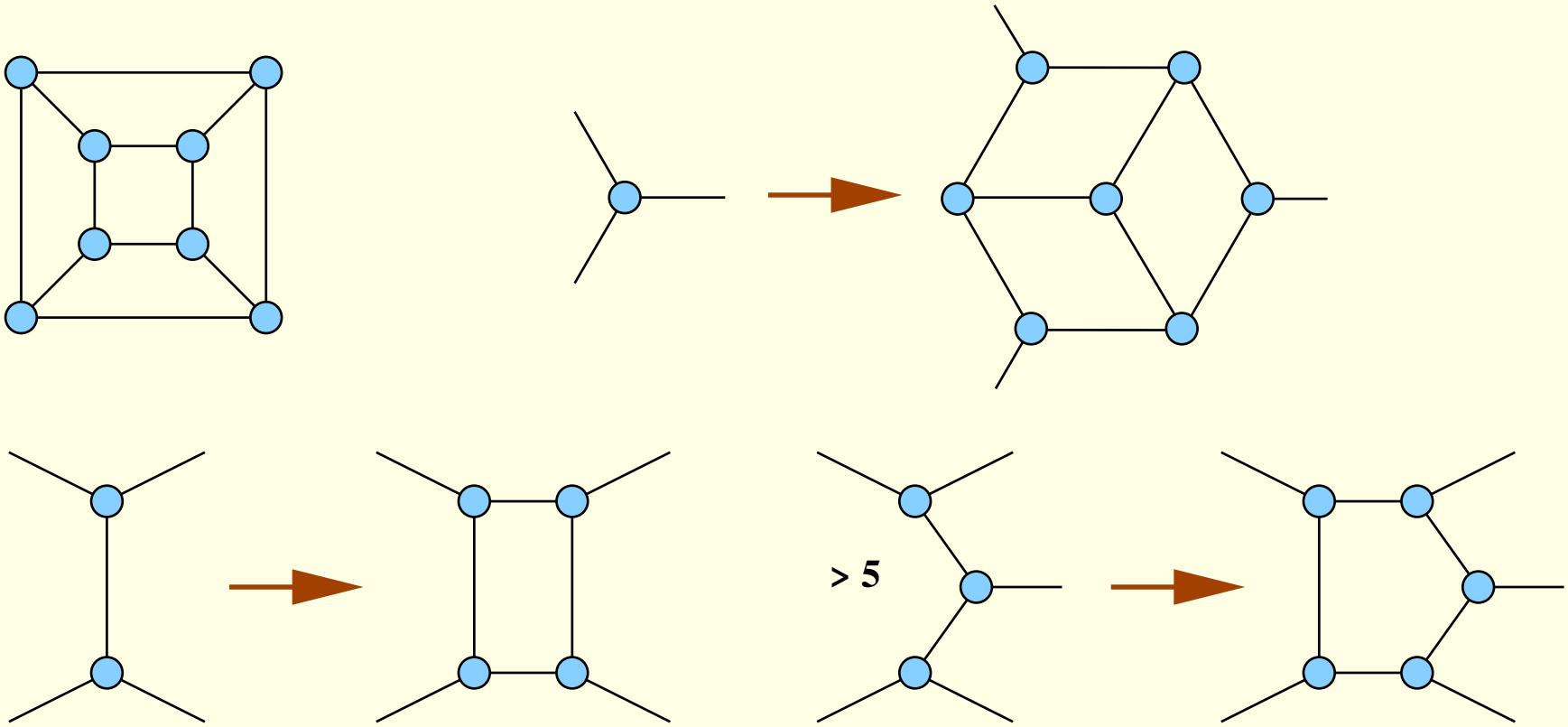
The reverse of an expansion is a reduction.

Example: 3-connected planar cubic graphs



Example: 3-connected planar cubic graphs without triangles

The following generation method is a slight improvement on one discovered by Batagelj (1989).



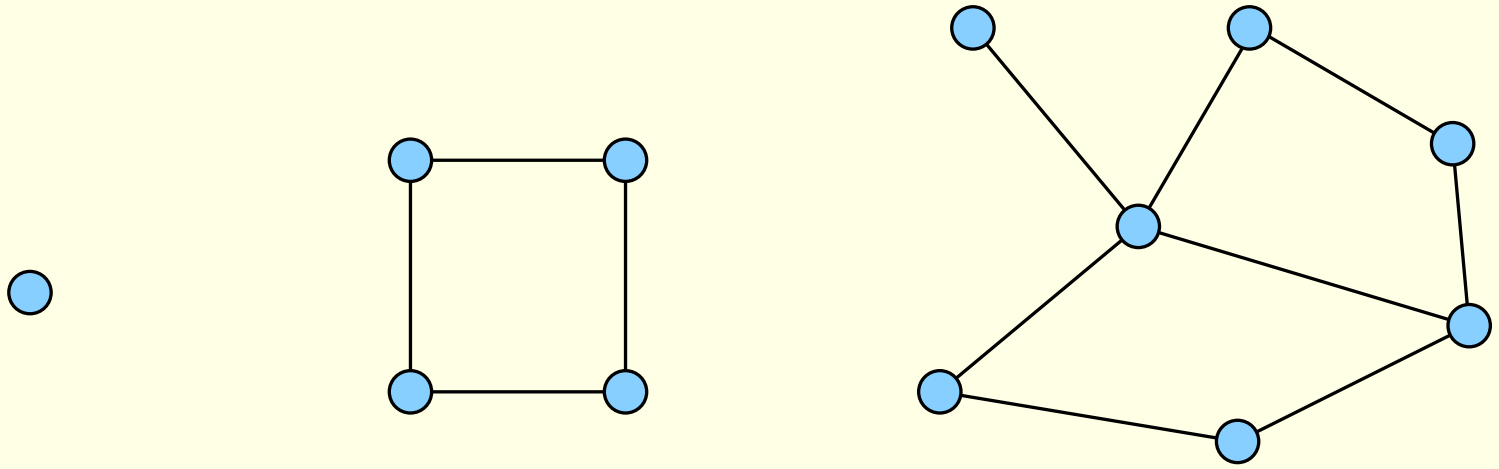
Isomorph-free generation

Once we have a recursive characterization, we can generate the graphs in the class, but how to we eliminate isomorphic copies?

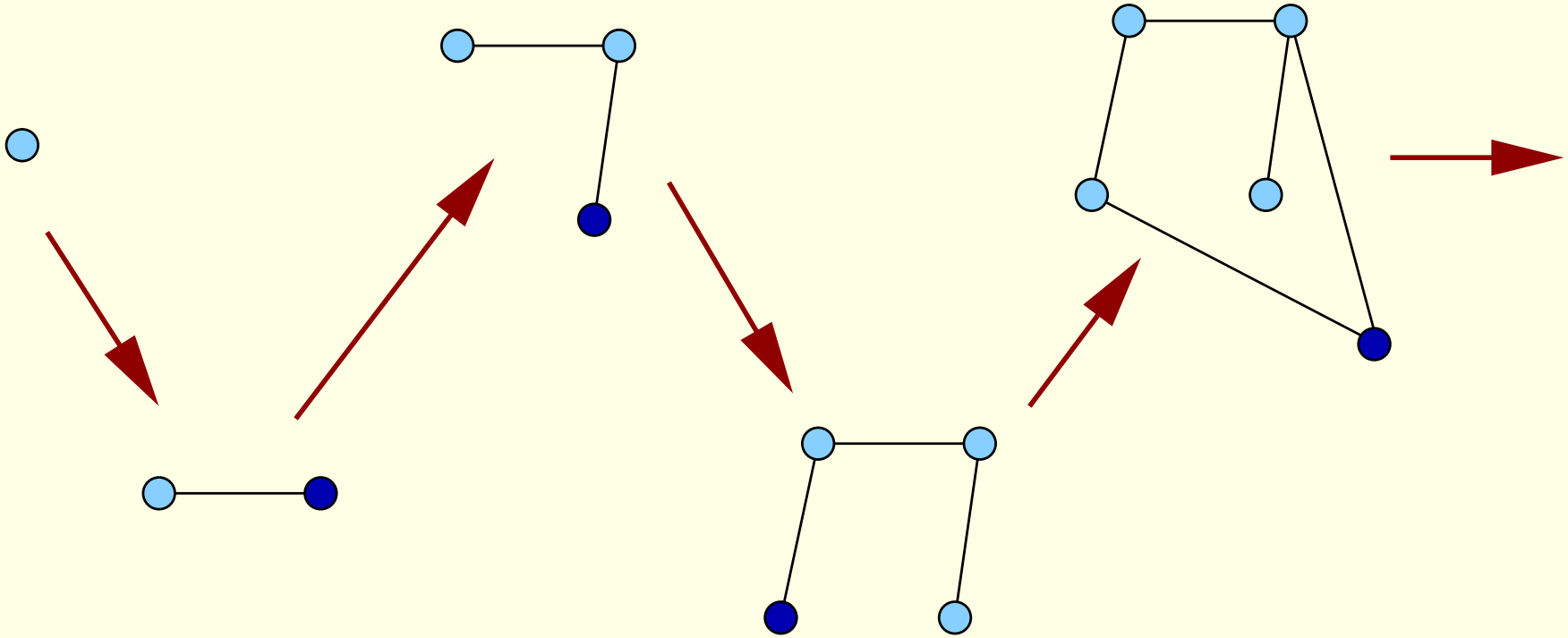
Isomorph-free generation

Once we have a recursive characterization, we can generate the graphs in the class, but how to we eliminate isomorphic copies?

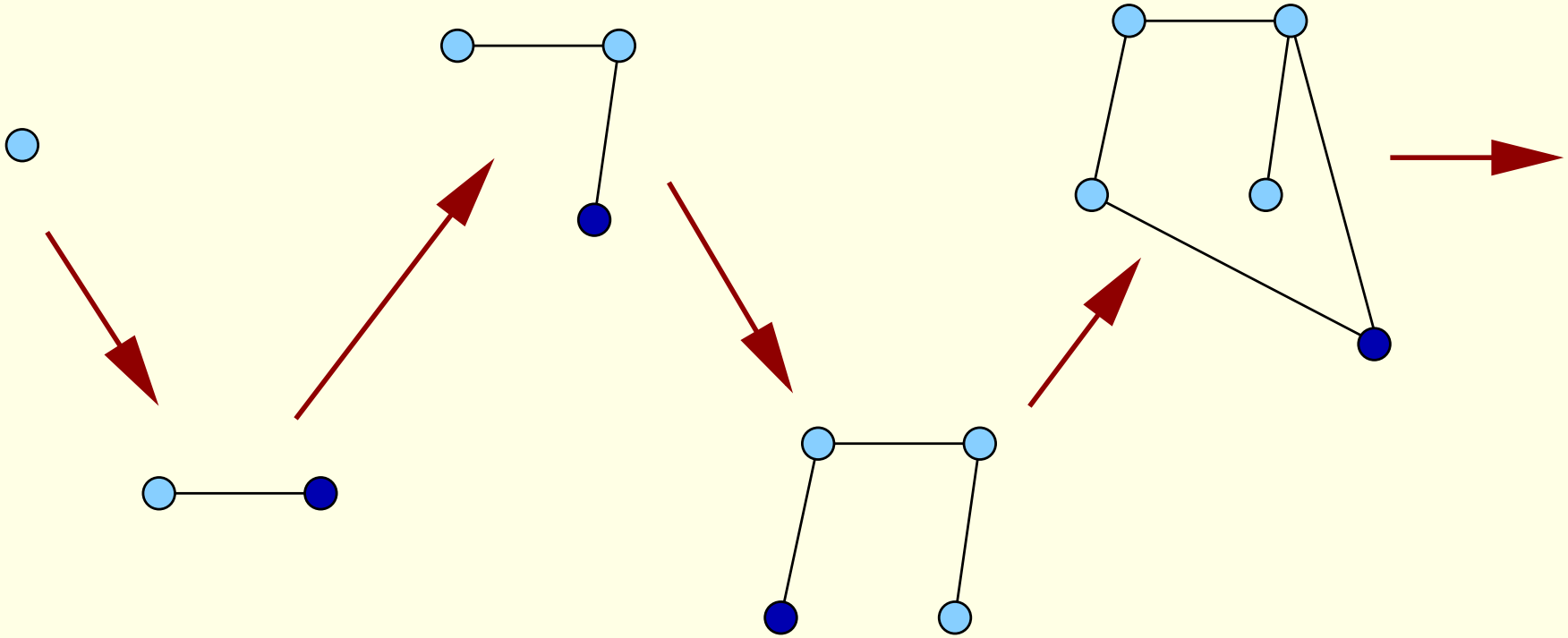
Toy Example: **triangle-free planar graphs**



Obvious recursive construction: add one vertex at a time starting with one vertex:



Obvious recursive construction: add one vertex at a time starting with one vertex:



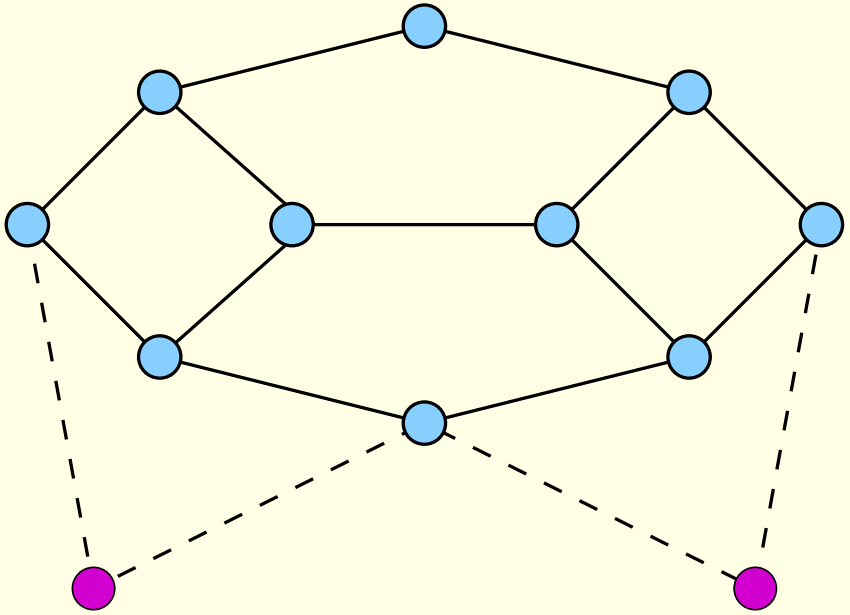
The difficulty is that isomorphic graphs appear.

1st source of isomorphs: symmetry

Equivalent expansions result in isomorphic children.

1st source of isomorphs: symmetry

Equivalent expansions result in isomorphic children.

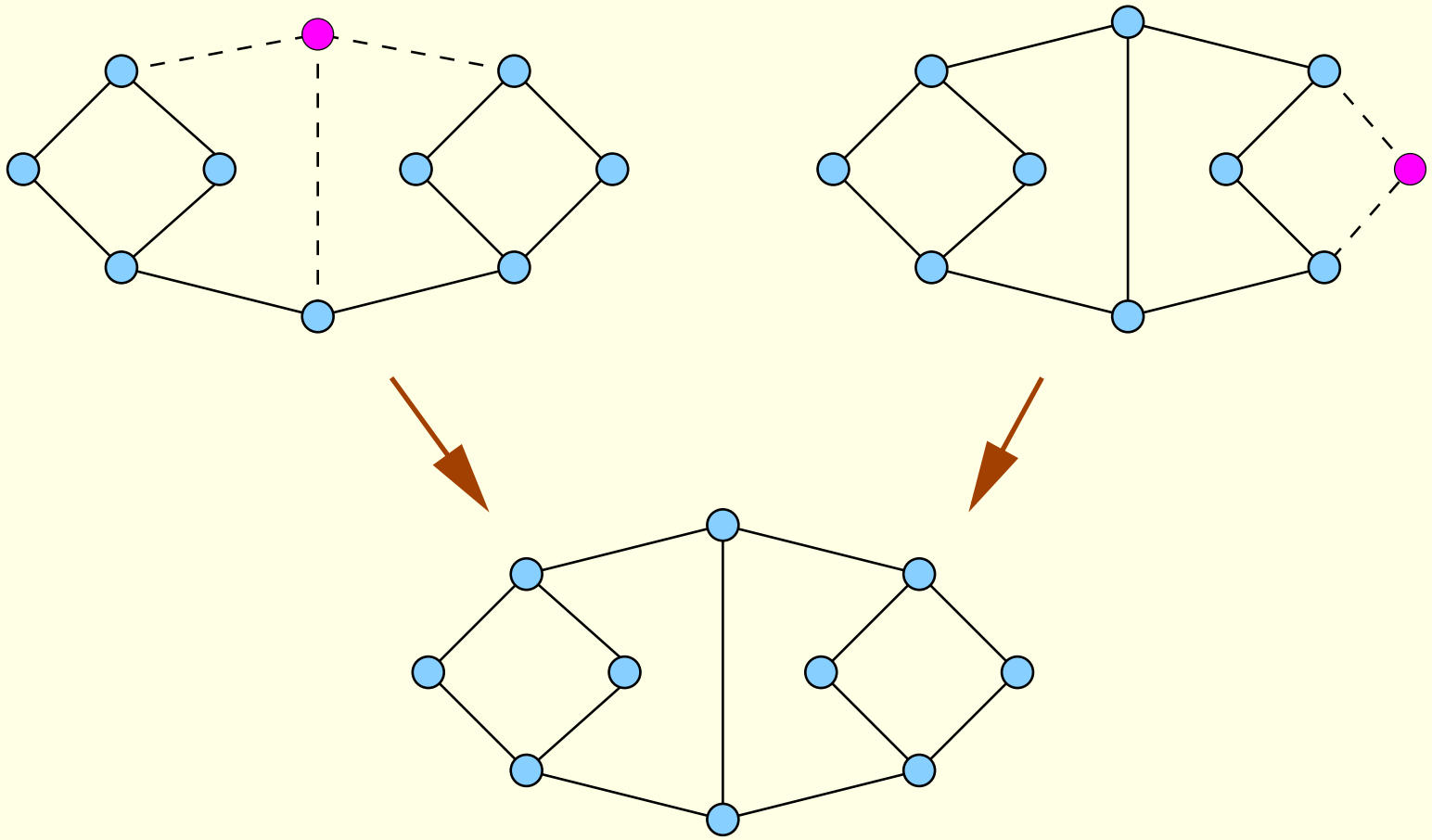


2nd source of isomorphs: different parents

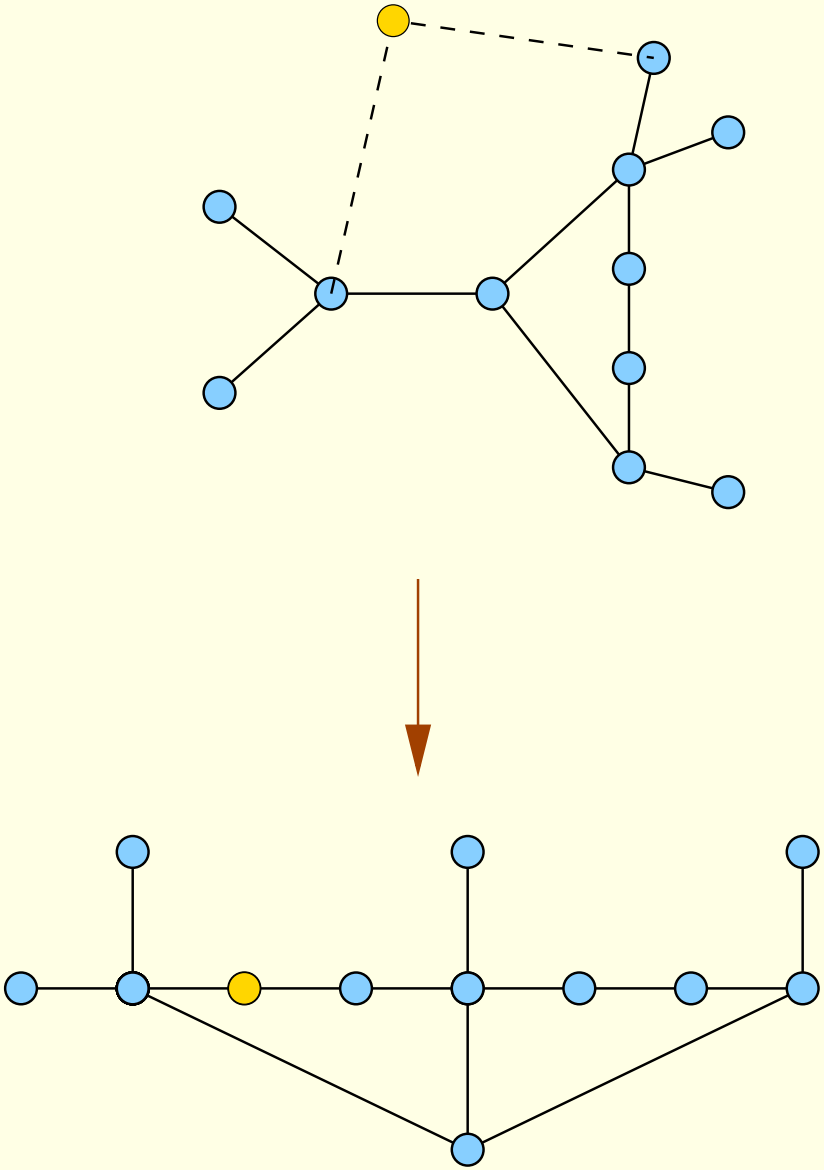
Slightly different parents can sometimes be extended to isomorphic children.

2nd source of isomorphs: different parents

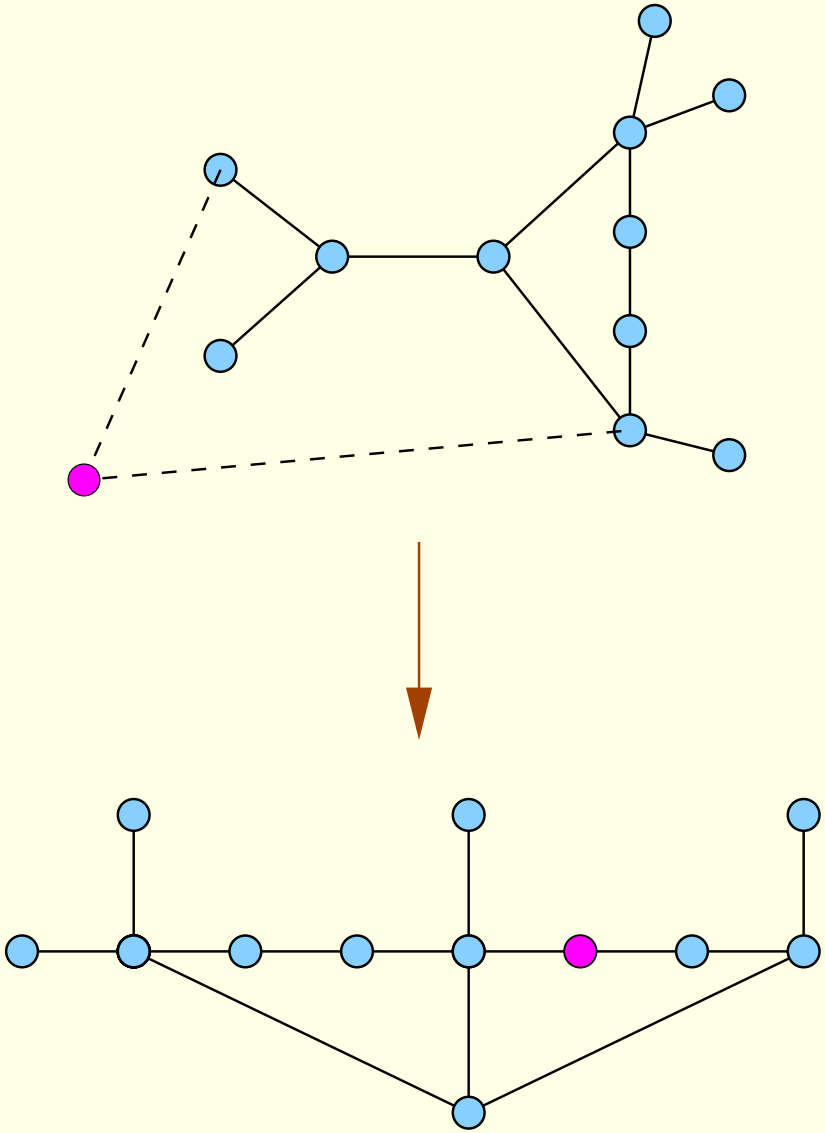
Slightly different parents can sometimes be extended to isomorphic children.



3rd source of isomorphs: pseudosimilarity



3rd source of isomorphs: pseudosimilarity



Generation by Canonical Construction Path

Also called canonical augmentation. McKay (1998)

Here we attempt to counter the three sources of isomorphs directly.

Generation by Canonical Construction Path

Also called canonical augmentation. McKay (1998)

Here we attempt to counter the three sources of isomorphs directly.

1st source: symmetry

Rule #1: Only make extensions inequivalent under the automorphism group of the smaller graph.

Generation by Canonical Construction Path

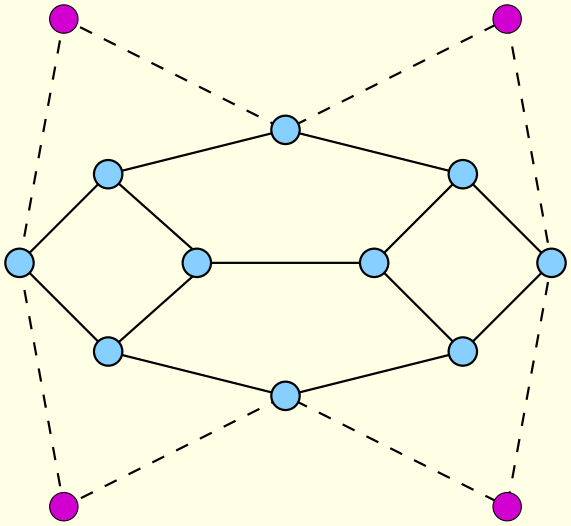
Also called canonical augmentation. McKay (1998)

Here we attempt to counter the three sources of isomorphs directly.

1st source: symmetry

Rule #1: Only make extensions inequivalent under the automorphism group of the smaller graph.

Perform at most one of these:



2nd and 3rd sources: different expansion

This includes construction from two different parents and construction from the same parent in two inequivalent ways.

2nd and 3rd sources: different expansion

This includes construction from two different parents and construction from the same parent in two inequivalent ways.

For each reducible graph, define a canonical equivalence class of reductions. Here "canonical" means "independent of the labelling" and "equivalence class" means "equivalent under the automorphism group".

In the triangle-free graphs example, an equivalence class of reductions is an orbit of vertices.

A *canonical* orbit of vertices could be the orbit that contains the vertex labelled first by a canonical labelling program like `nauty`. (In practice, we use a layered sequence of invariants to choose an equivalence class without invoking `nauty` most of the time.)

2nd source: different expansion (continued)

Canonical orbit of reductions:

$$\mathcal{C} : \text{graph } G \to \text{orbit of reductions}$$

$$\mathcal{C}(G^\gamma) = \mathcal{C}(G)^\gamma \qquad (\gamma \in S_n)$$

2nd source: different expansion (continued)

Rule #2: **If object G is made using expansion ϕ, reject G unless $\phi^{-1} \in \mathcal{C}(G)$.**

2nd source: different expansion (continued)

Canonical orbit of reductions:

$$\mathcal{C} : \text{graph } G \to \text{orbit of reductions}$$

$$\mathcal{C}(G^\gamma) = \mathcal{C}(G)^\gamma \qquad (\gamma \in S_n)$$

2nd source: different expansion (continued)

Rule #2: If object G is made using expansion ϕ, reject G unless $\phi^{-1} \in \mathcal{C}(G)$.

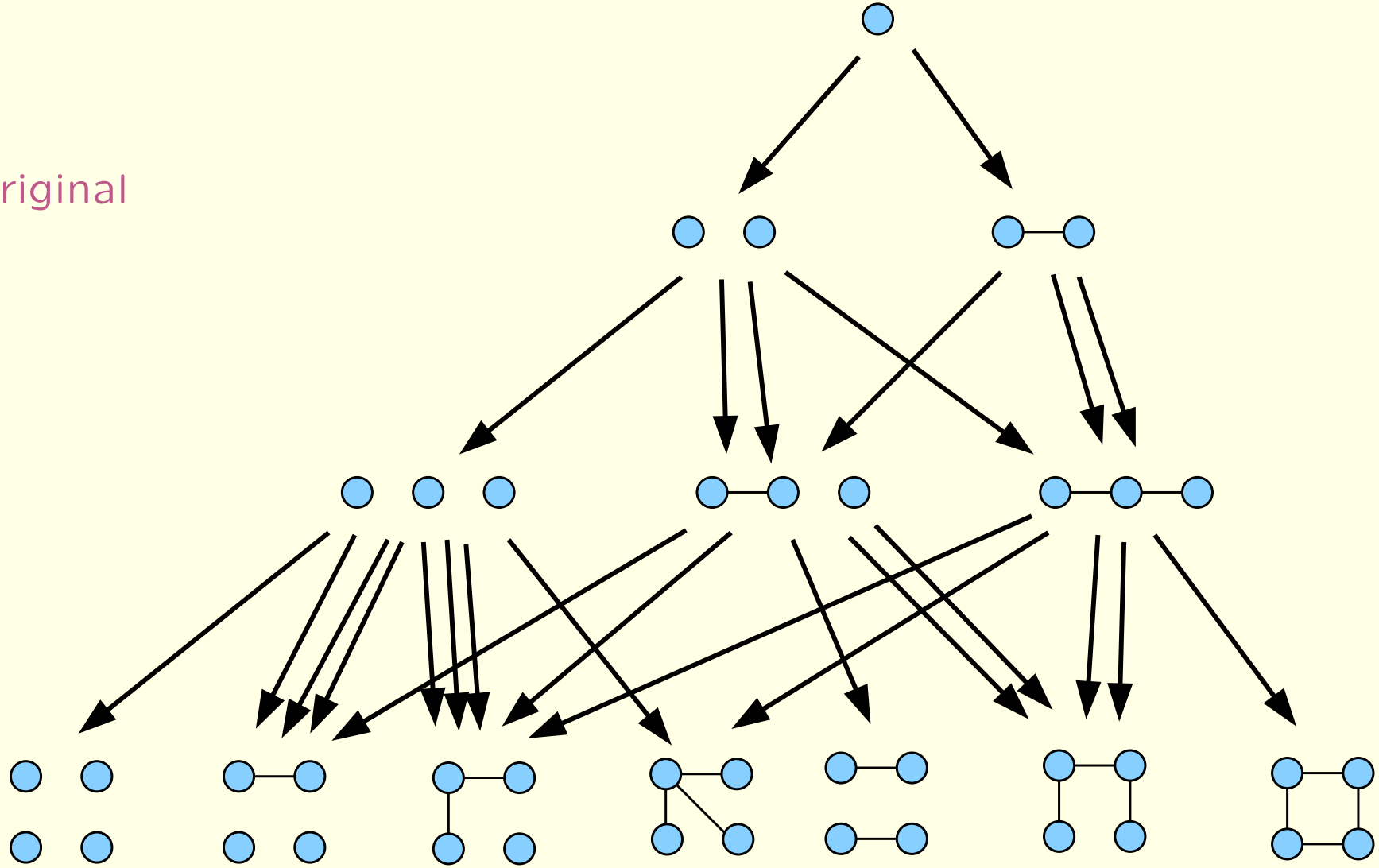
Theorem (McKay, 1989): If rules #1 and #2 are obeyed, and certain conditions hold, then all isomorphs are eliminated.

The "certain conditions" mostly involve the definition of symmetry.

Application to triangle-free planar graphs

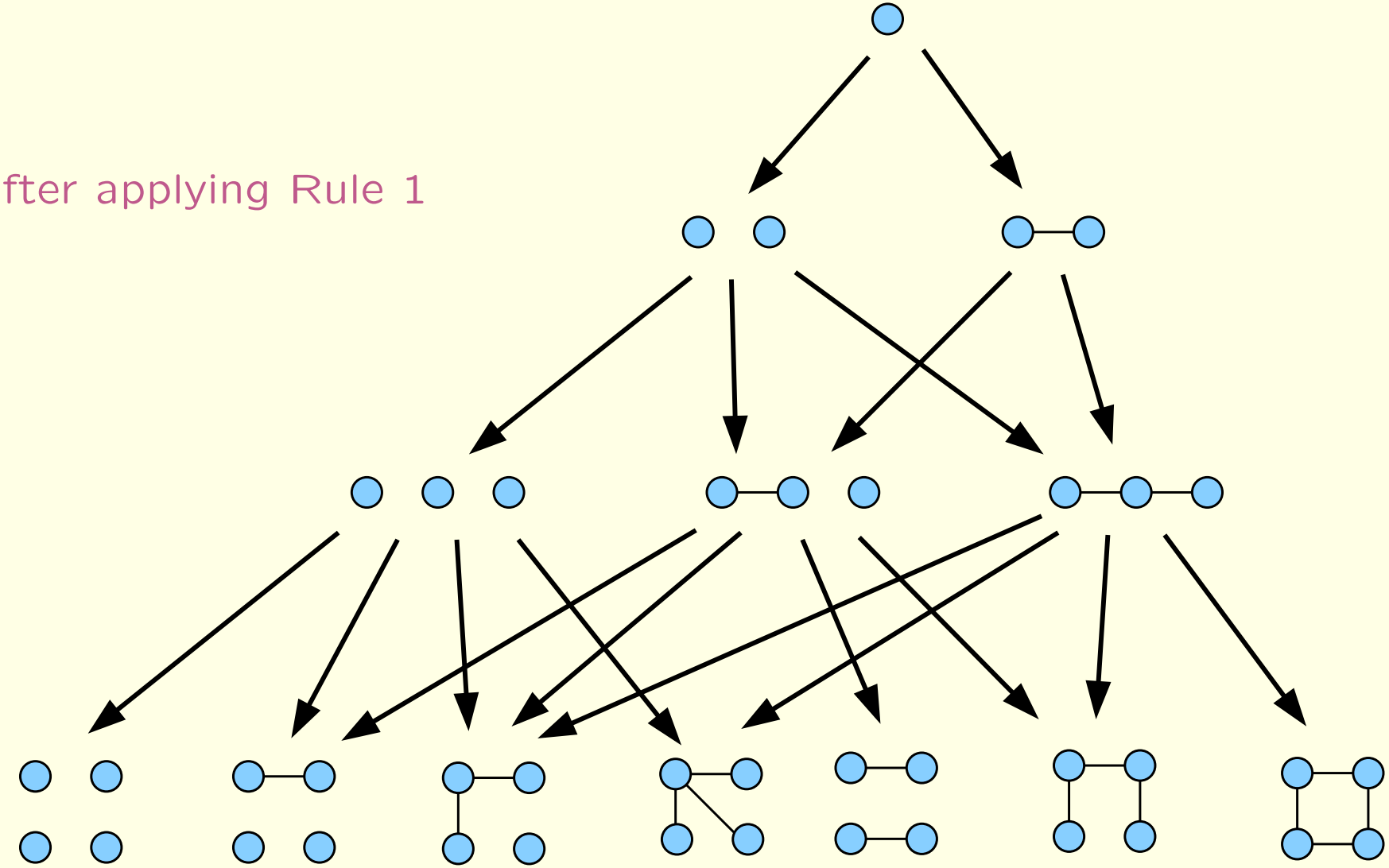
Application to triangle-free planar graphs

Original



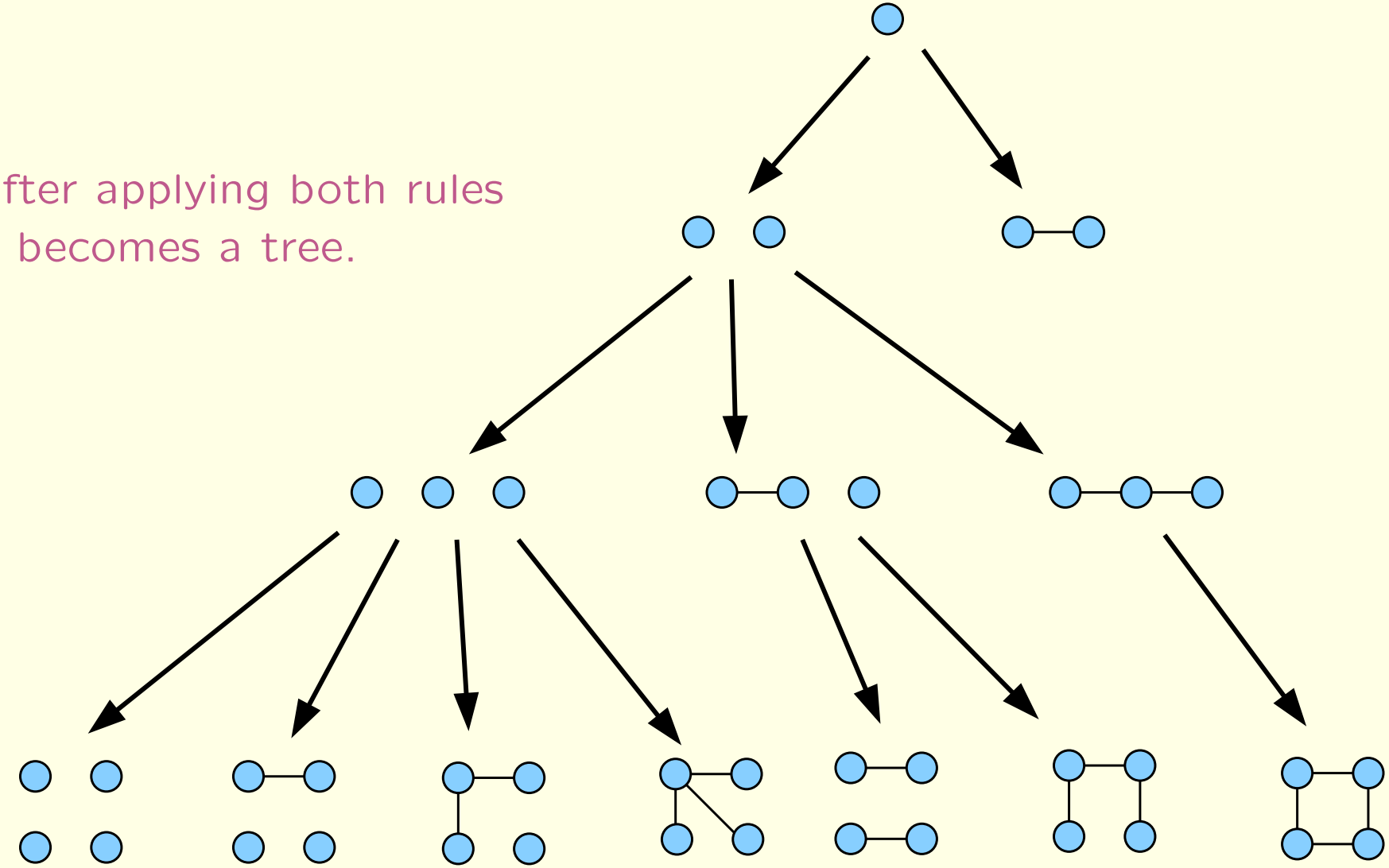
Application to triangle-free planar graphs

After applying Rule 1



Application to triangle-free planar graphs

After applying both rules
it becomes a tree.



3-connected planar cubic graphs without triangles

All 16,747,182,732,792 such graphs on 50 vertices were generated in about 80 days (program written with Gunnar Brinkmann).

- Suppose we want to estimate the number on 100 vertices.
- Suppose we want to estimate an average property at 100 vertices, such as the number of 6-cycles.

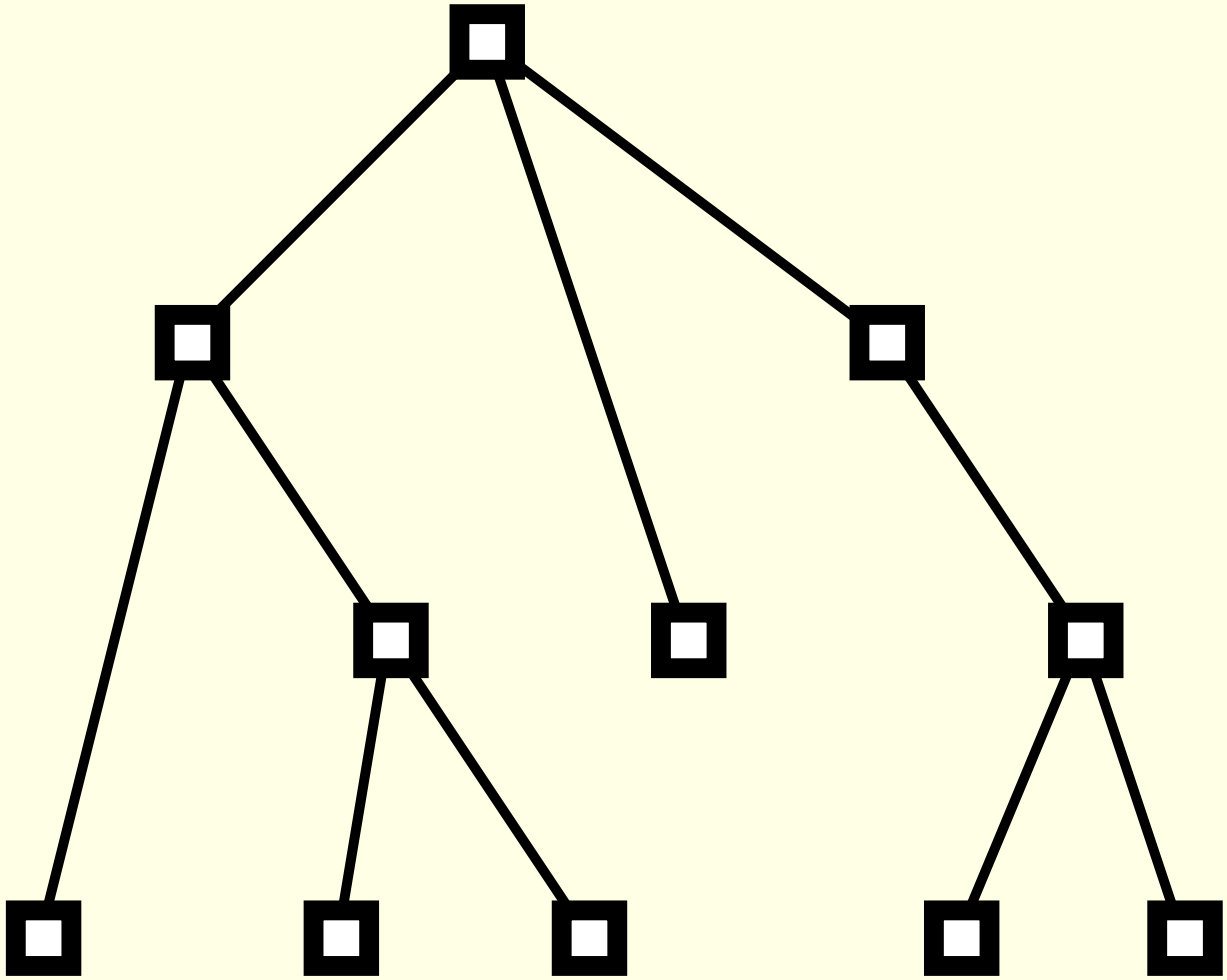
3-connected planar cubic graphs without triangles

All 16,747,182,732,792 such graphs on 50 vertices were generated in about 80 days (program written with Gunnar Brinkmann).

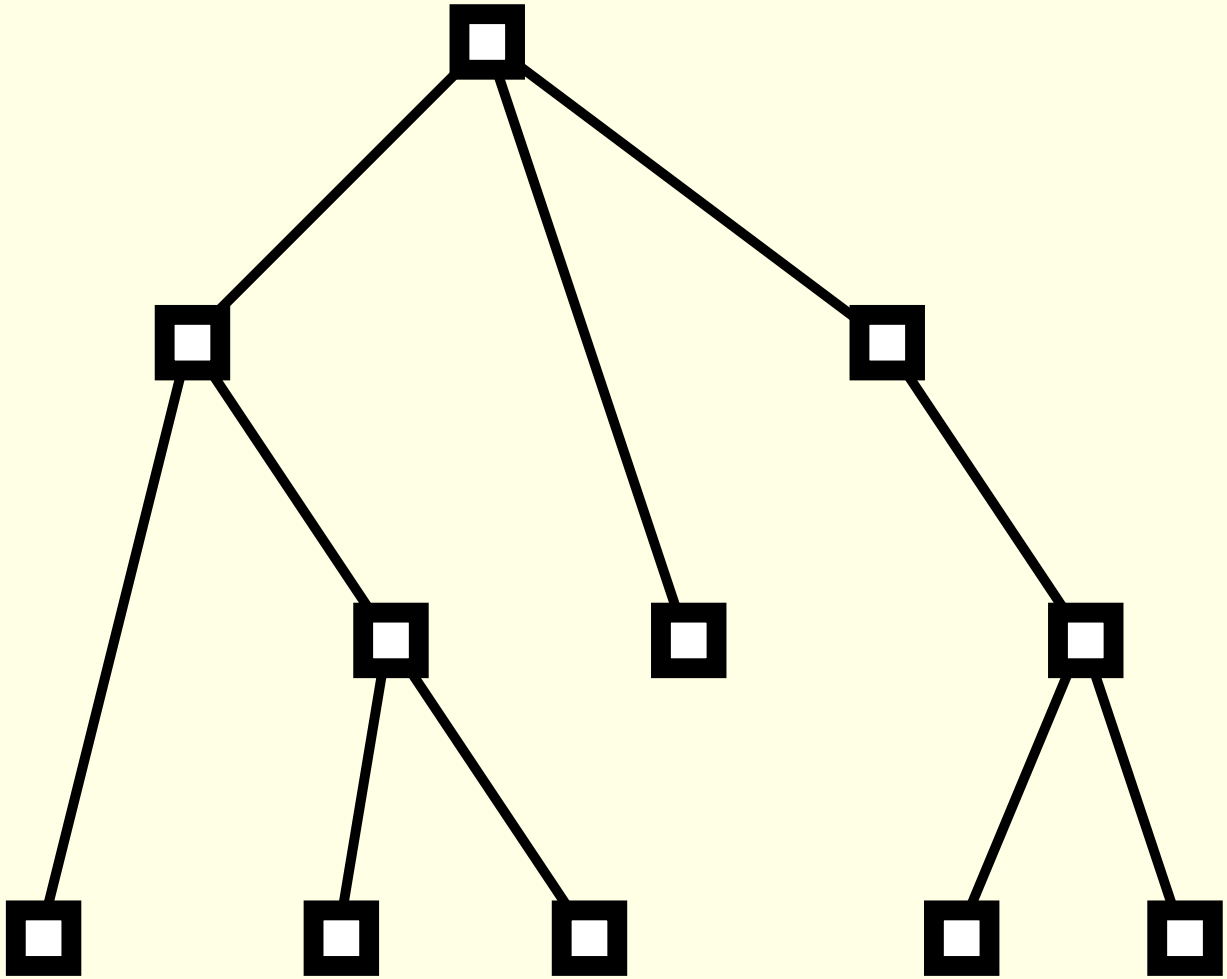
- Suppose we want to estimate the number on 100 vertices.
- Suppose we want to estimate an average property at 100 vertices, such as the number of 6-cycles.

The key to solving these problems is that the generation process has a tree structure.

Estimation

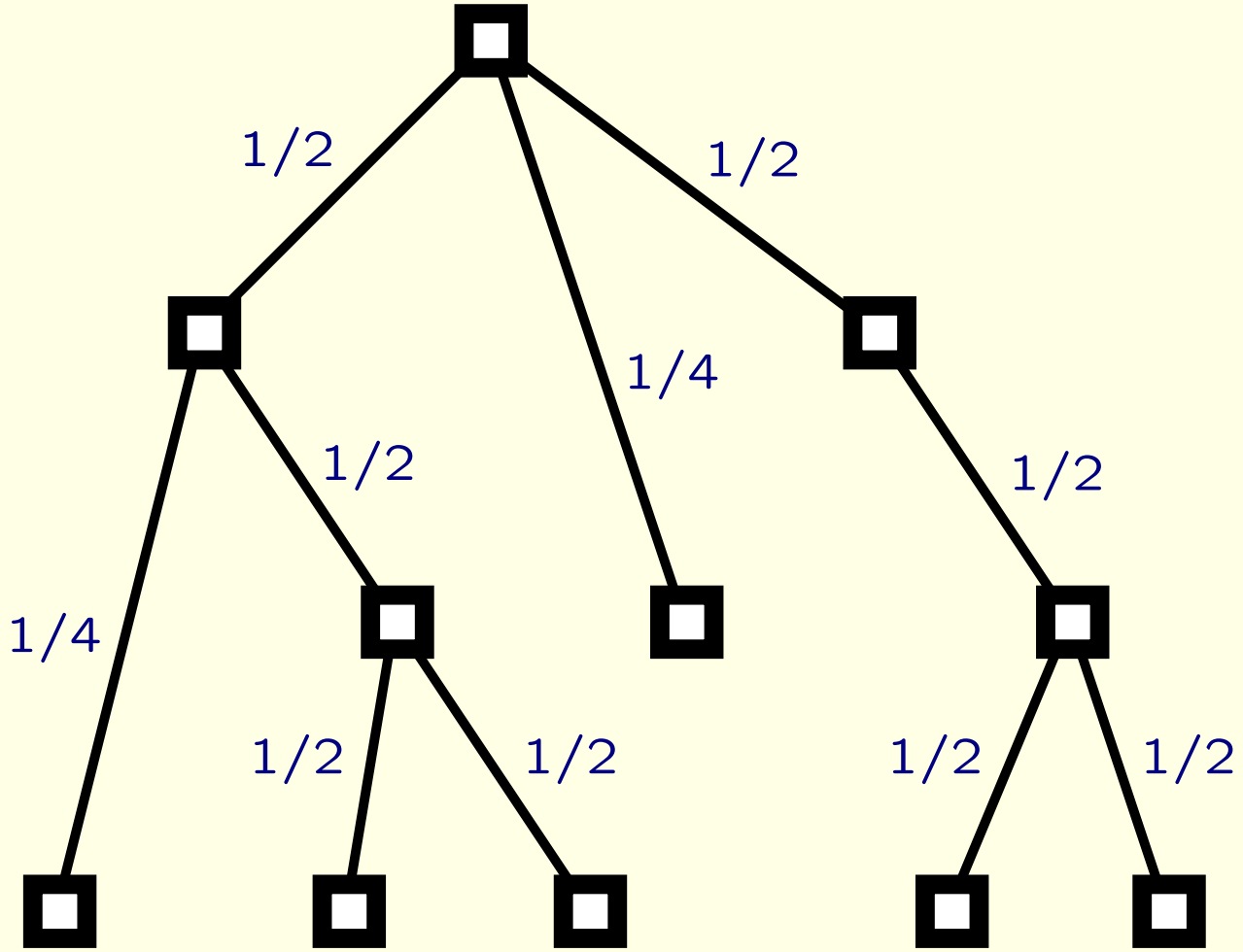


Estimation



The nodes at the lowest level are the output graphs.

Estimation



The nodes at the lowest level are the output graphs.

Assign probabilities to each edge so that the probability of each output is the same (here $p = 1/8$).

Estimation (continued)

Run the generation to completion, taking each branch with the probability assigned to that branch.

Let p be the probability that each graph in the class appears in the output. By design, p is constant.

Estimation (continued)

Run the generation to completion, taking each branch with the probability assigned to that branch.

Let p be the probability that each graph in the class appears in the output. By design, p is constant.

Define

$$N = \frac{1}{p}\,(\text{number of output graphs})$$

$$X_6 = \frac{1}{p} \sum_{\text{output } G} C_6(G),$$

where $C_6(G)$ is the number of 6-cycles in G.

Estimation (continued)

Run the generation to completion, taking each branch with the probability assigned to that branch.

Let p be the probability that each graph in the class appears in the output. By design, p is constant.

Define

$$N = \frac{1}{p}\,(\text{number of output graphs})$$

$$X_6 = \frac{1}{p} \sum_{\text{output } G} C_6(G),$$

where $C_6(G)$ is the number of 6-cycles in G.

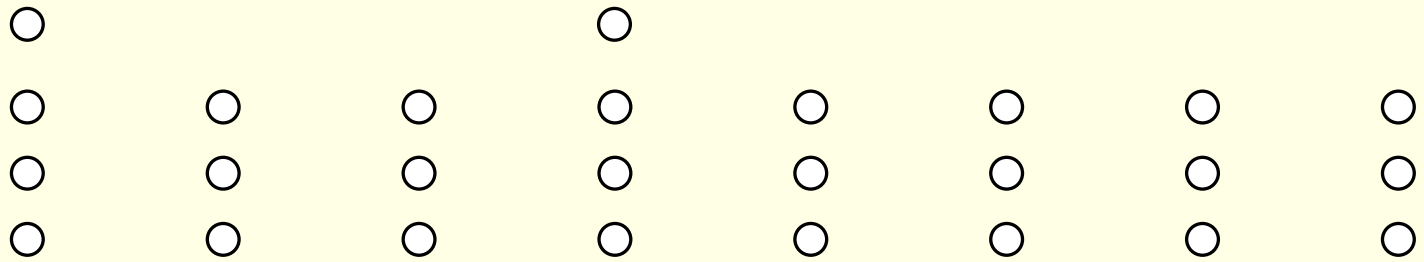
Then N and X_6 are unbiased estimators of the total number of graphs and the total number of 6-cycles in all the graphs. X_6/N is a (biased) estimator of the average number of 6-cycles.

But, how to make one graph uniformly at random?

Example: Simple graphs with degrees 4,3,3,4,3,3,3,3

But, how to make one graph uniformly at random?

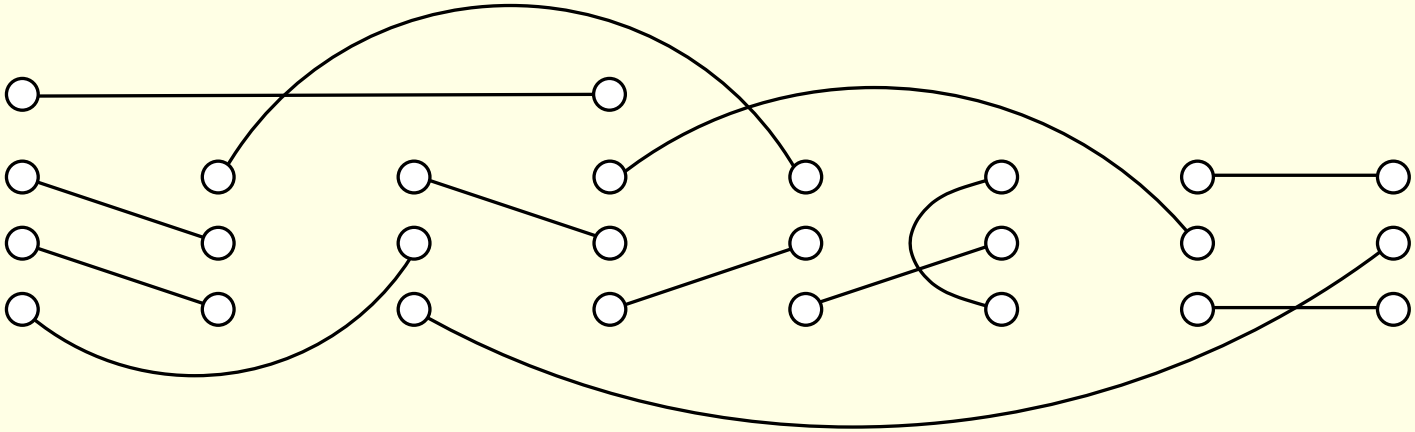
Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



Take groups of dots according to the required degrees.

But, how to make one graph uniformly at random?

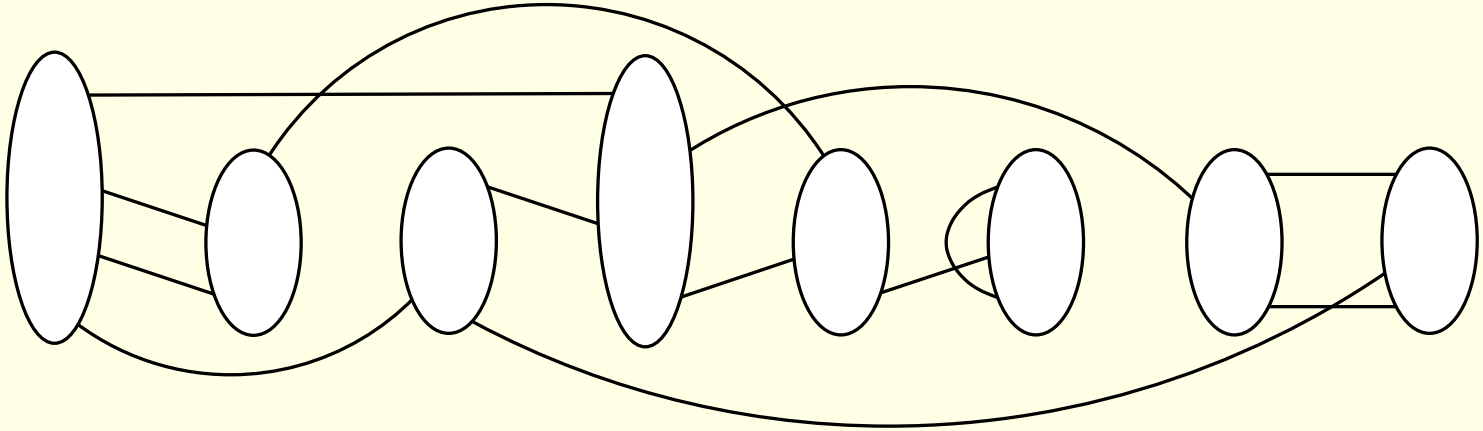
Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



Pair them at random.

But, how to make one graph uniformly at random?

Example: Simple graphs with degrees 4,3,3,4,3,3,3,3

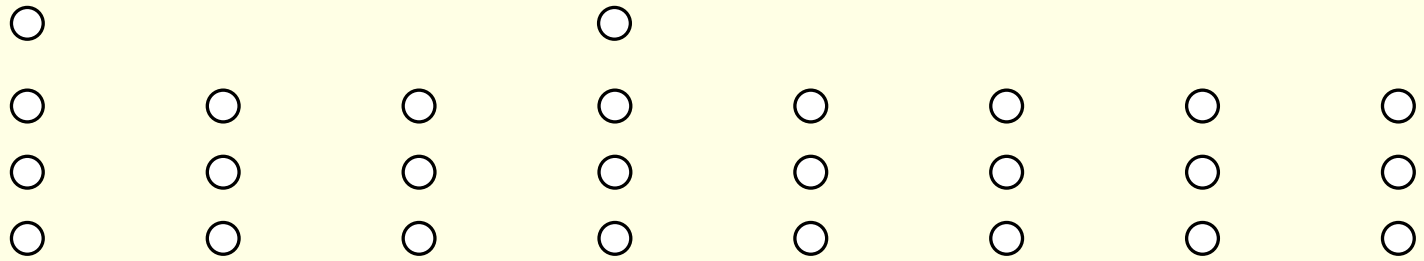


Convert the groups of dots into vertices.

Note the loops and multiple edges.

But, how to make one graph uniformly at random?

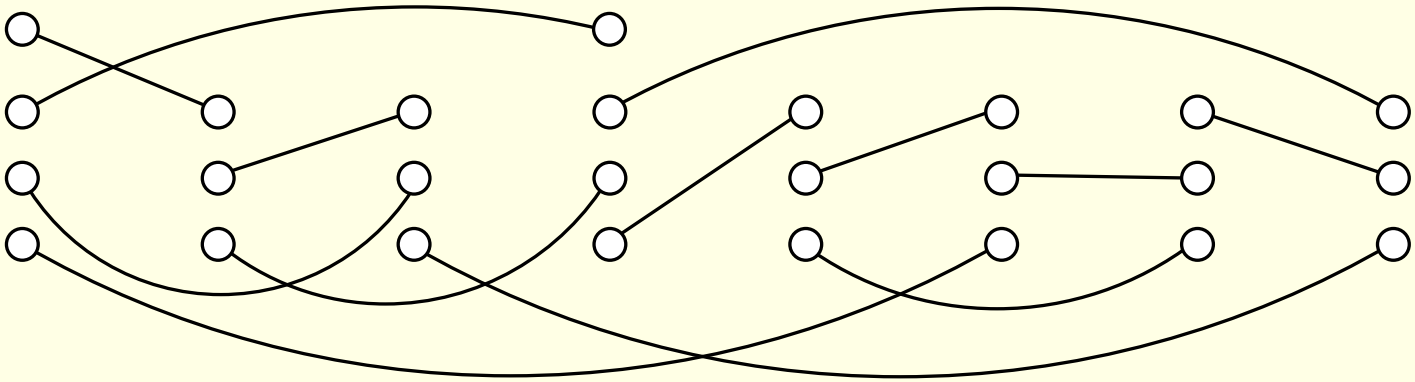
Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



Try again: Take groups of dots.

But, how to make one graph uniformly at random?

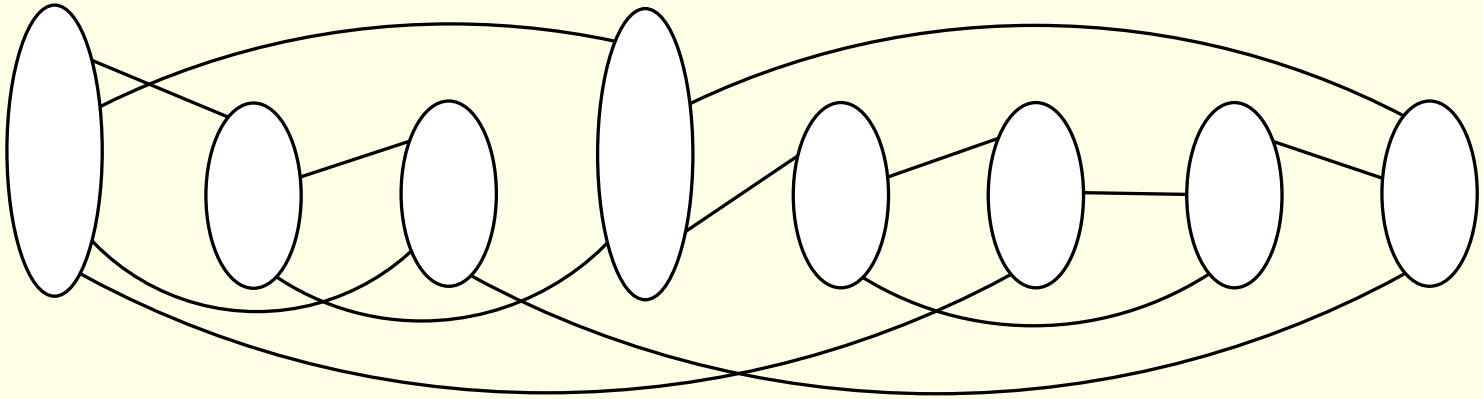
Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



Pair them at random.

But, how to make one graph uniformly at random?

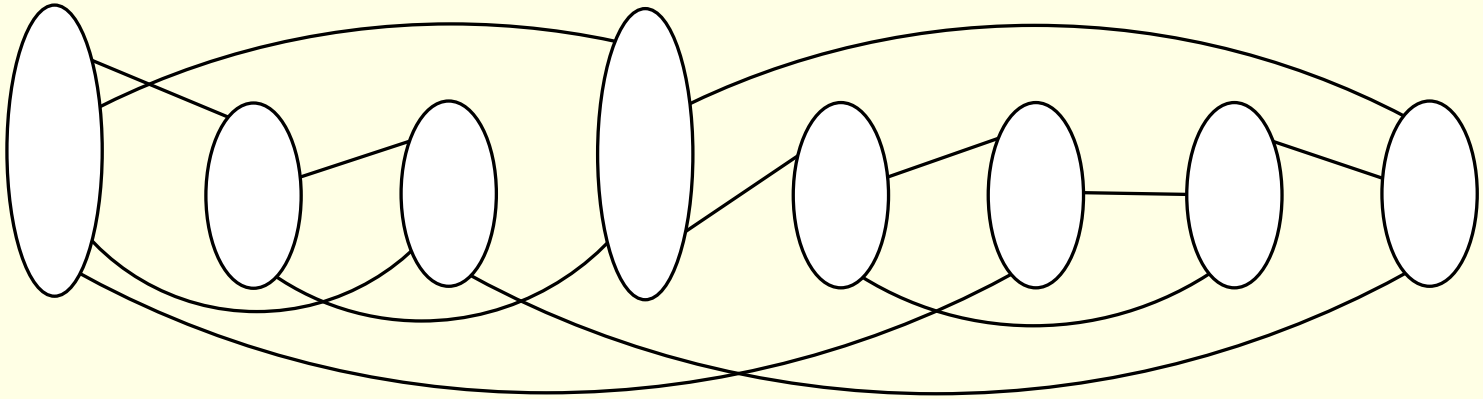
Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



This time the result is simple.

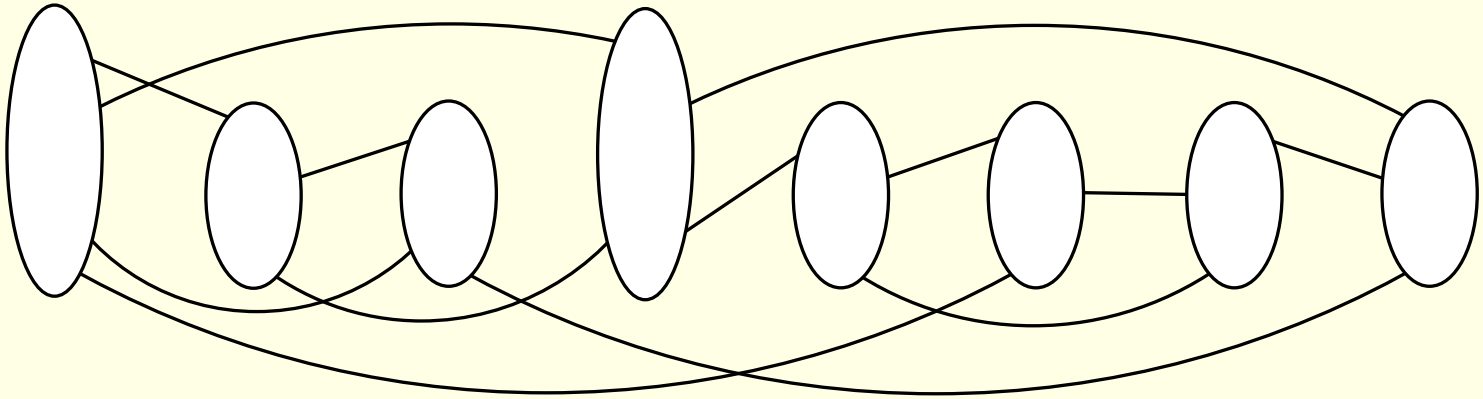
But, how to make one graph uniformly at random?

Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



But, how to make one graph uniformly at random?

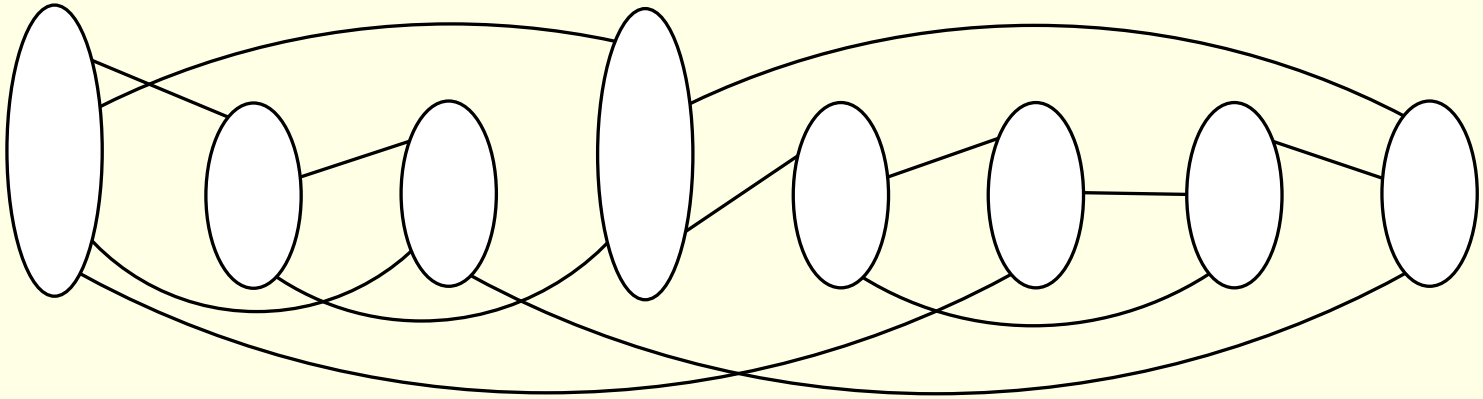
Example: Simple graphs with degrees 4,3,3,4,3,3,3,3



The key observation is that every simple graph with the given degree sequence is equally likely to be generated.

But, how to make one graph uniformly at random?

Example: Simple graphs with degrees 4,3,3,4,3,3,3,3

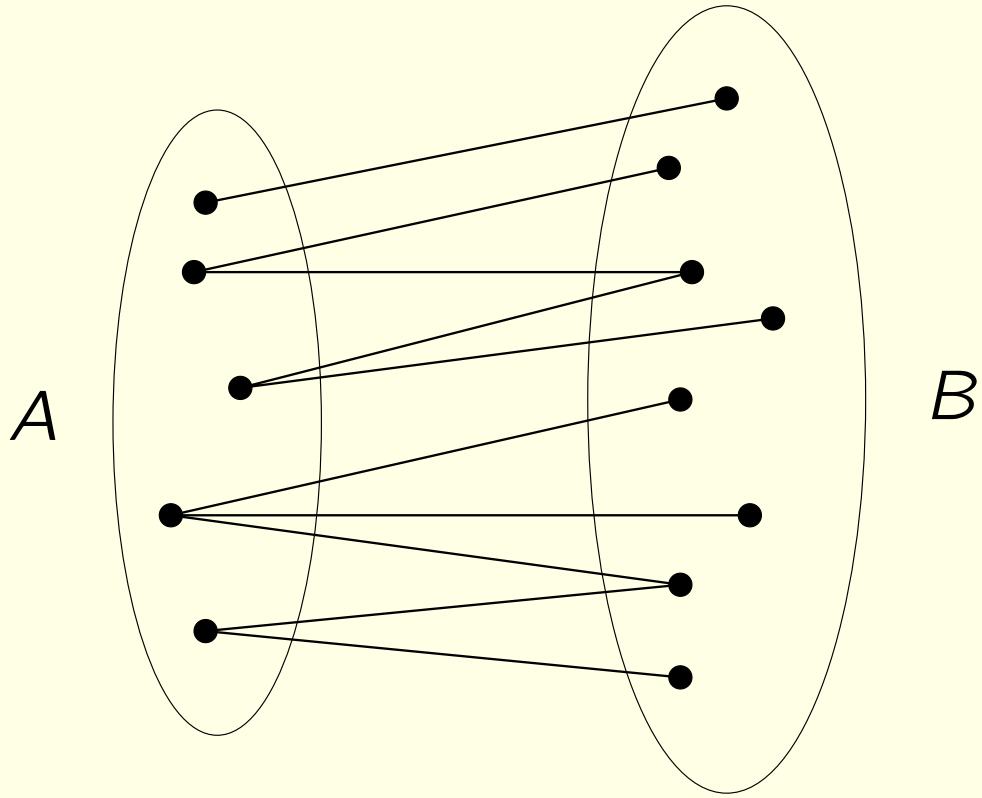


The key observation is that every simple graph with the given degree sequence is equally likely to be generated.

Alas, this is only efficient for low degree. For higher degree, too many attempts are required before a simple graph is obtained.

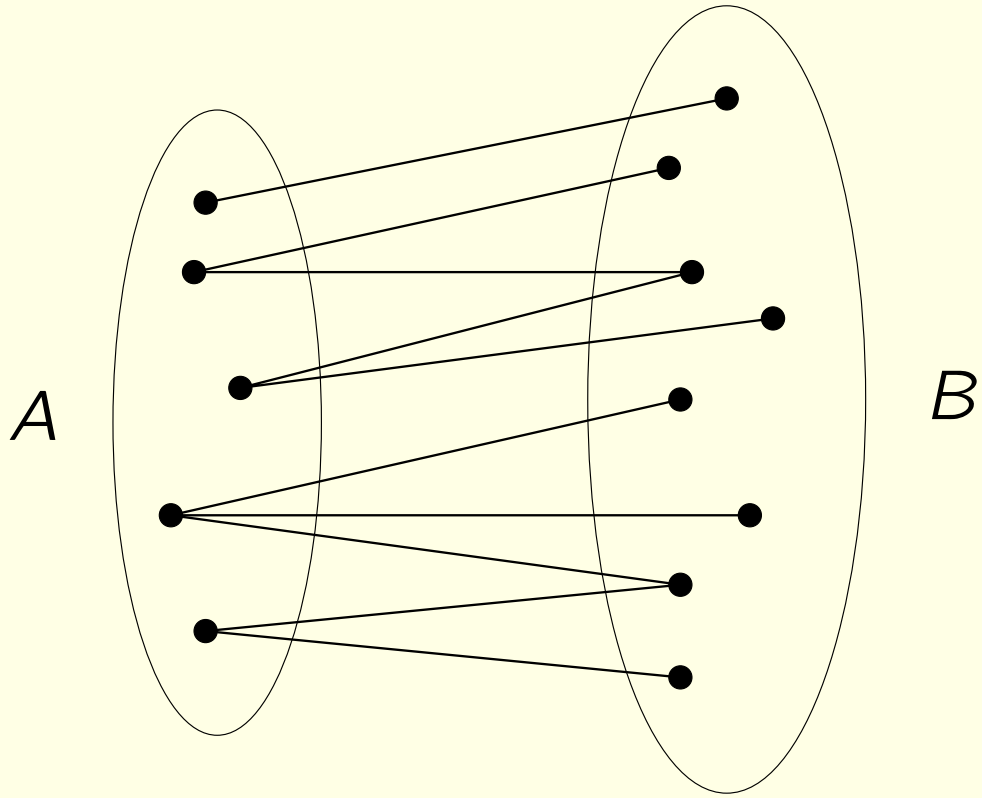
Accept-reject strategy

Consider two sets and a relation between them.



Accept-reject strategy

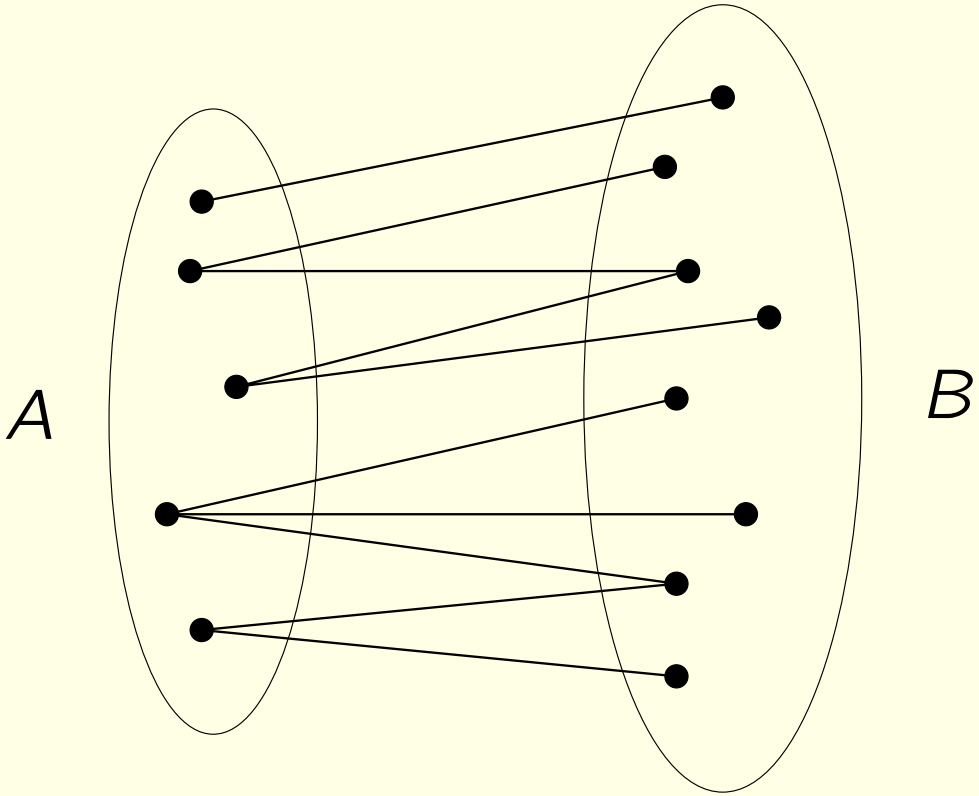
Consider two sets and a relation between them.



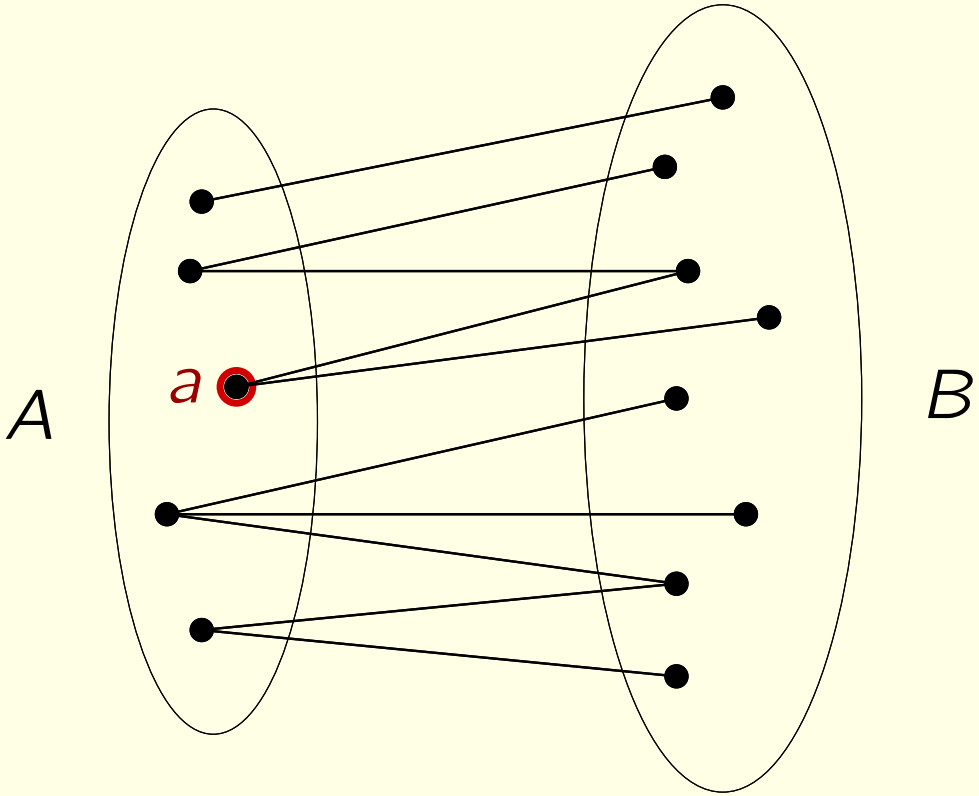
Suppose we know how to generate a random element of A.

How do we generate a random element of B?

Accept-reject strategy

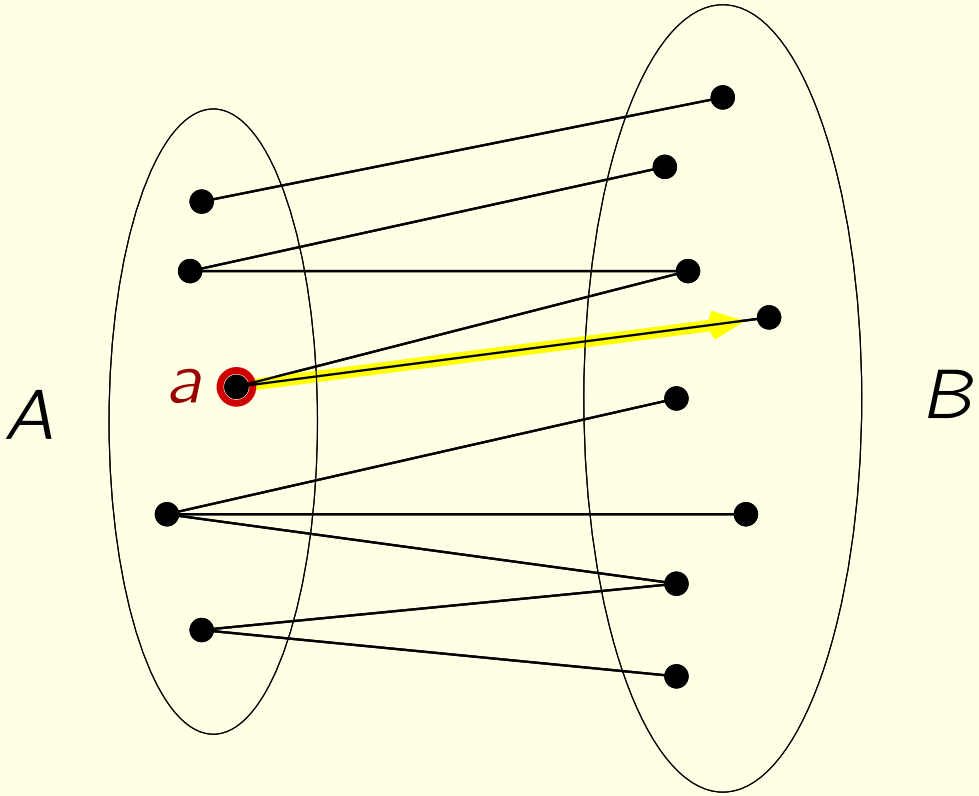


Accept-reject strategy



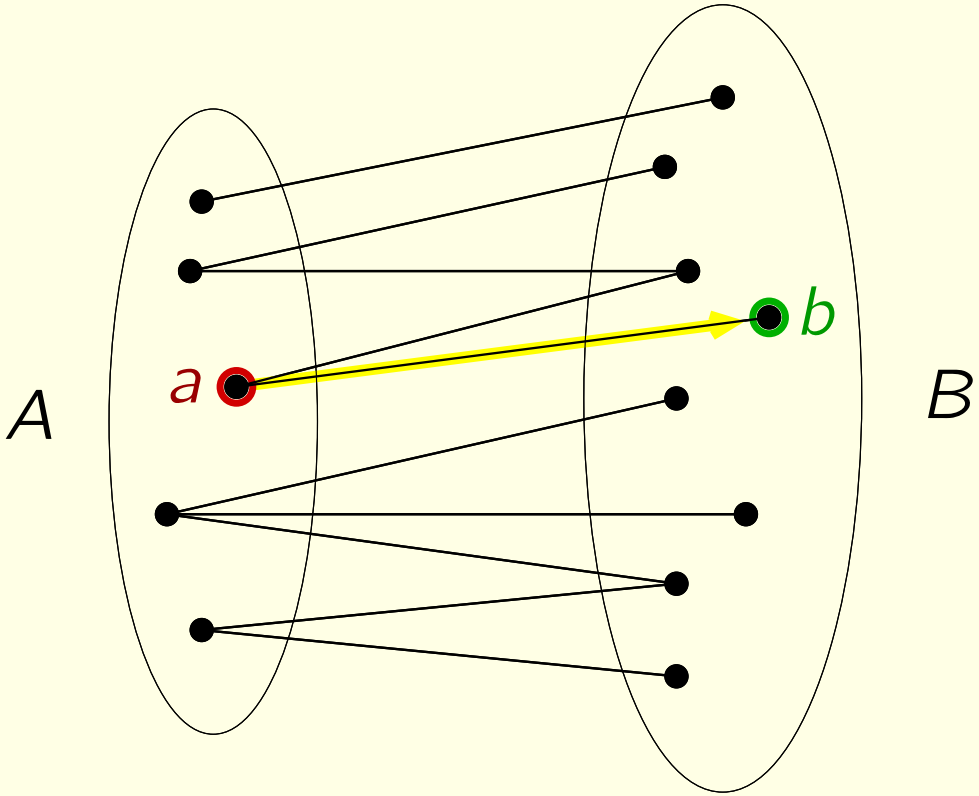
1. Choose random $a \in A$.

Accept-reject strategy



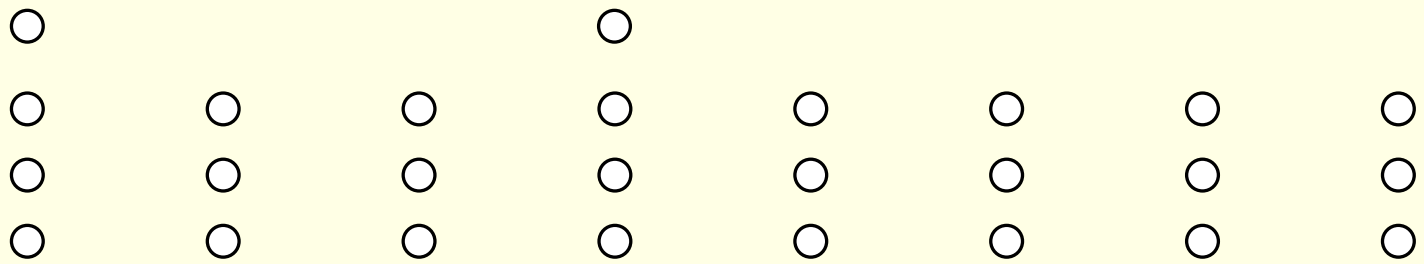
1. Choose random $a \in A$.
2. Take a random edge to B.

Accept-reject strategy



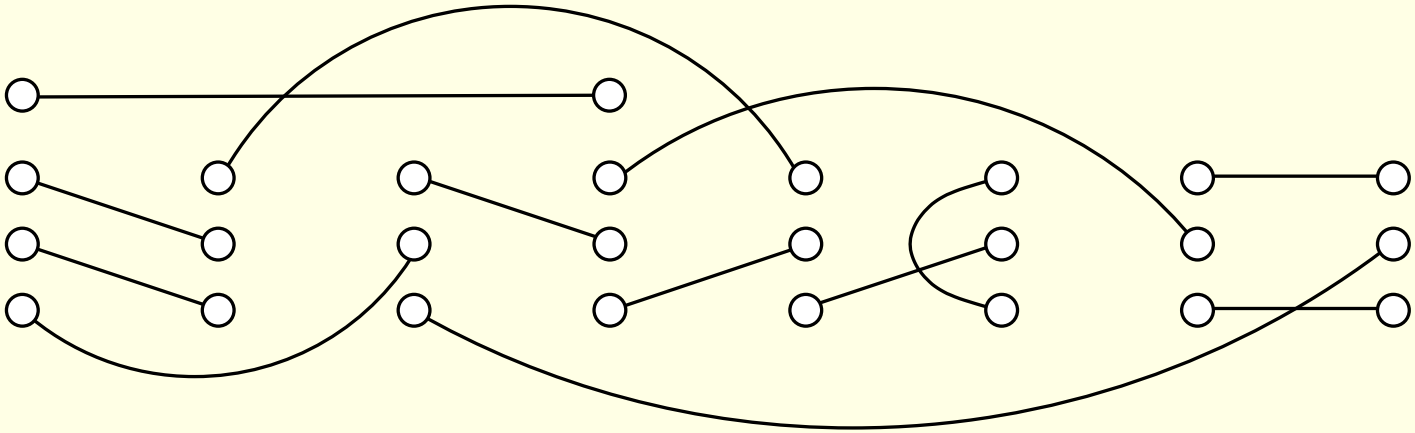
1. Choose random $a \in A$.
2. Take a random edge to B.
3. Accept $b \in B$ with probability proportional to $\deg(a)/\deg(b)$.
   If unsuccessful, try again.

Back to simple graphs with degrees 4,3,3,4,3,3,3,3



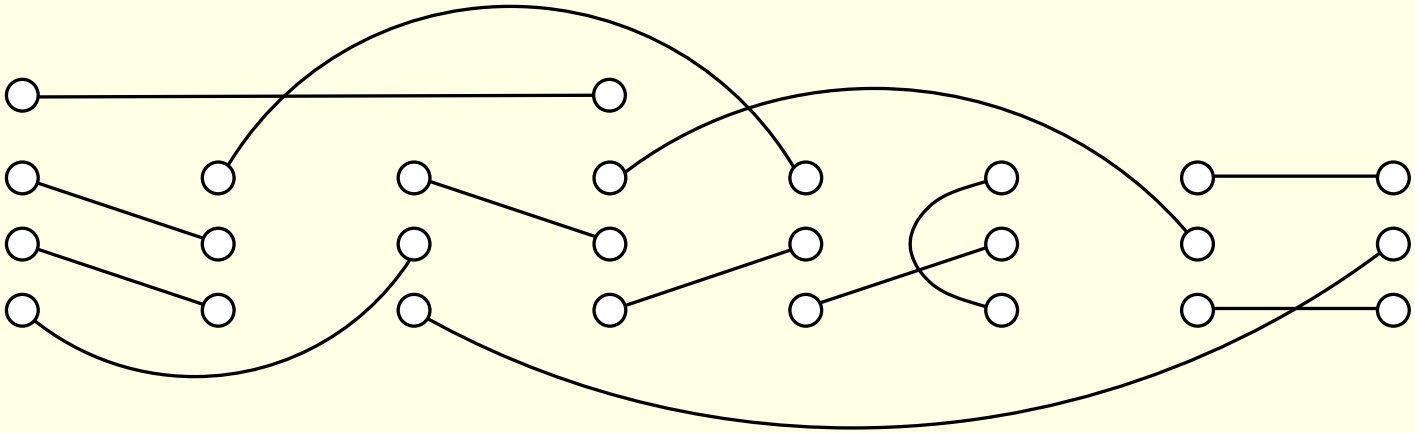
Take groups of dots according to the required degrees.

Back to simple graphs with degrees 4,3,3,4,3,3,3,3



Pair them at random.

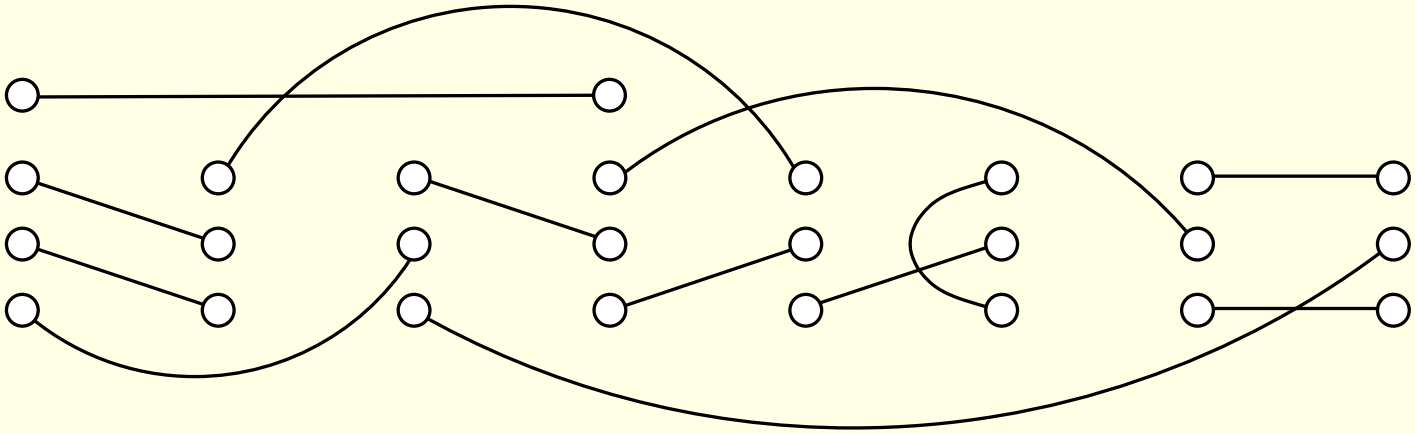
Back to simple graphs with degrees 4,3,3,4,3,3,3,3



Pair them at random.

Let's call this a random member of $G(1,2)$ because it has 1 loop and 2 double edges.

Back to simple graphs with degrees 4,3,3,4,3,3,3,3



Pair them at random.

Let's call this a random member of $G(1,2)$ because it has 1 loop and 2 double edges.

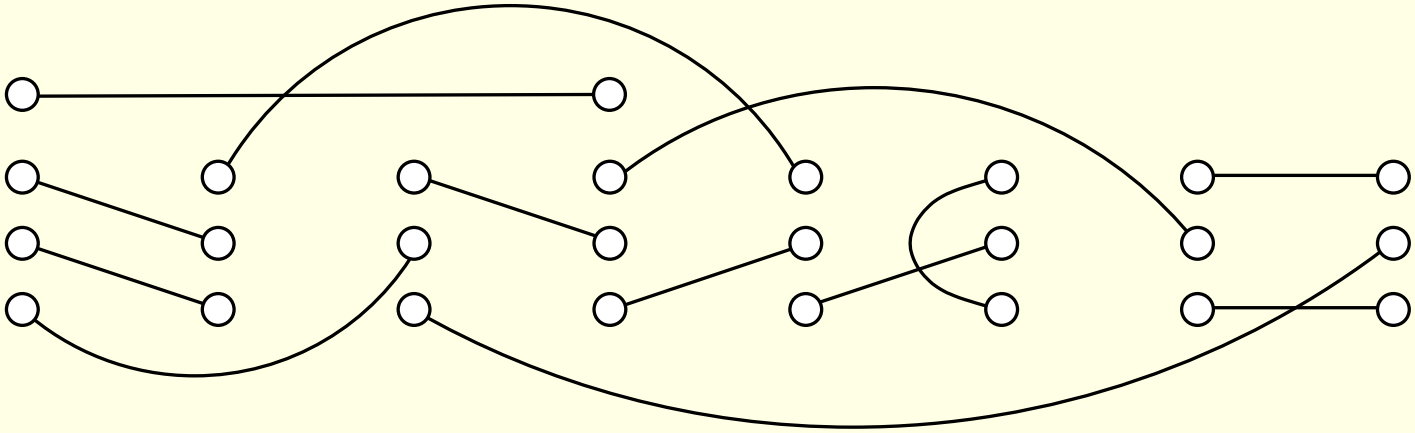
Using an accept-reject strategy, we can transfer uniform randomness:

$$G(1,2) \to G(1,1) \to G(1,0) \to G(0,0)$$

and then we will have a random simple graph.

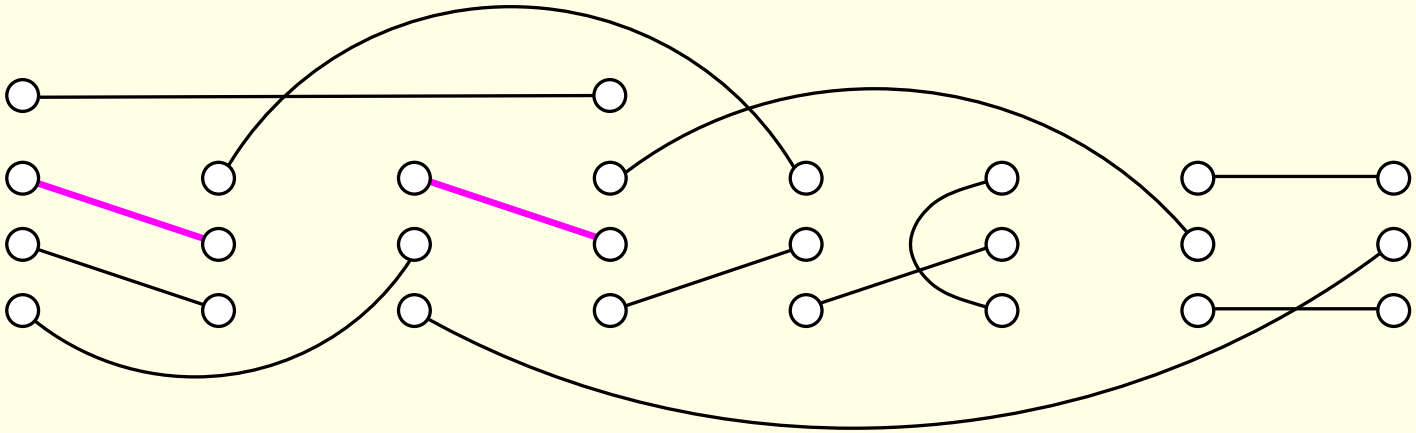
Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)

Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)



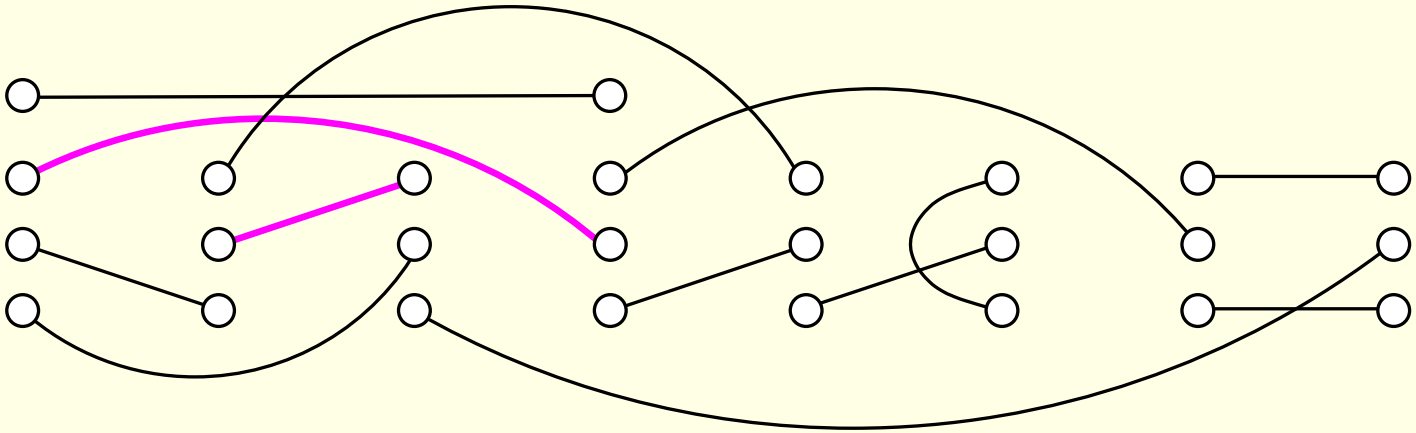
A random member of $G(2,1)$.

Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)



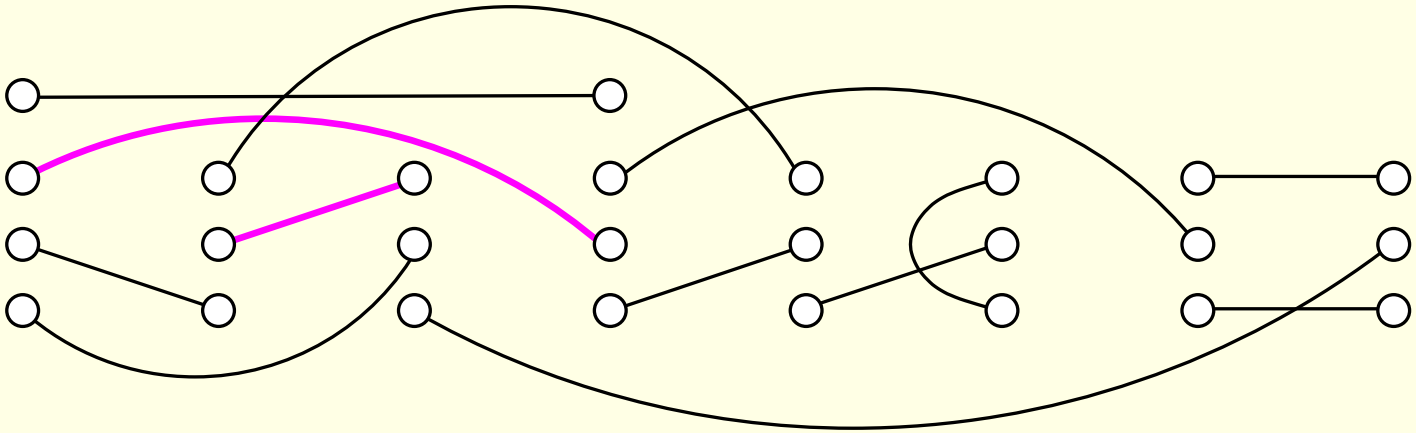
Choose an edge in a double edge and one other.

Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)



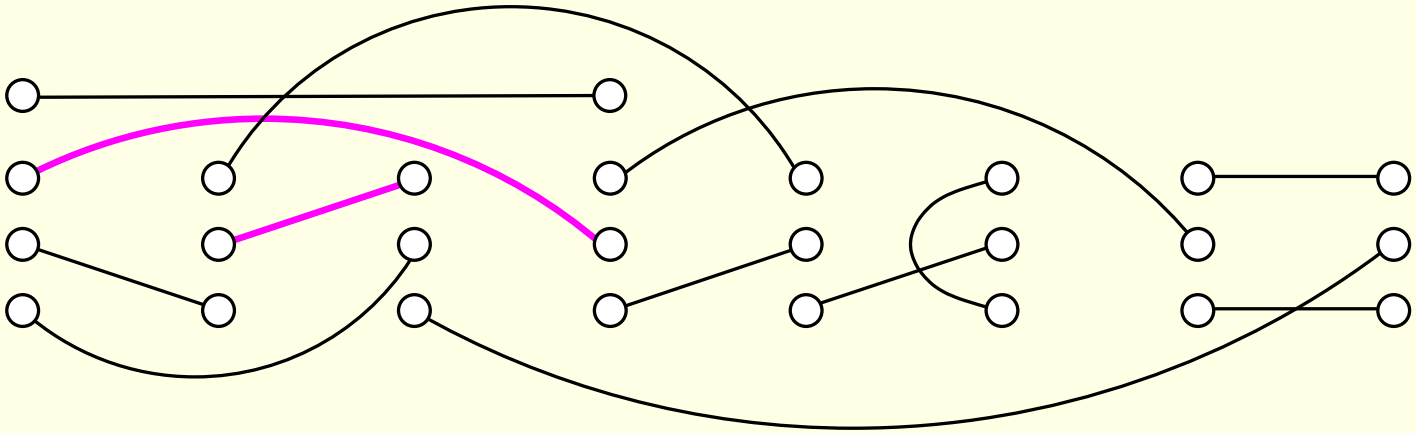
Swap for two other edges.

Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)



Possibly accept to get a member of $G(1,1)$.

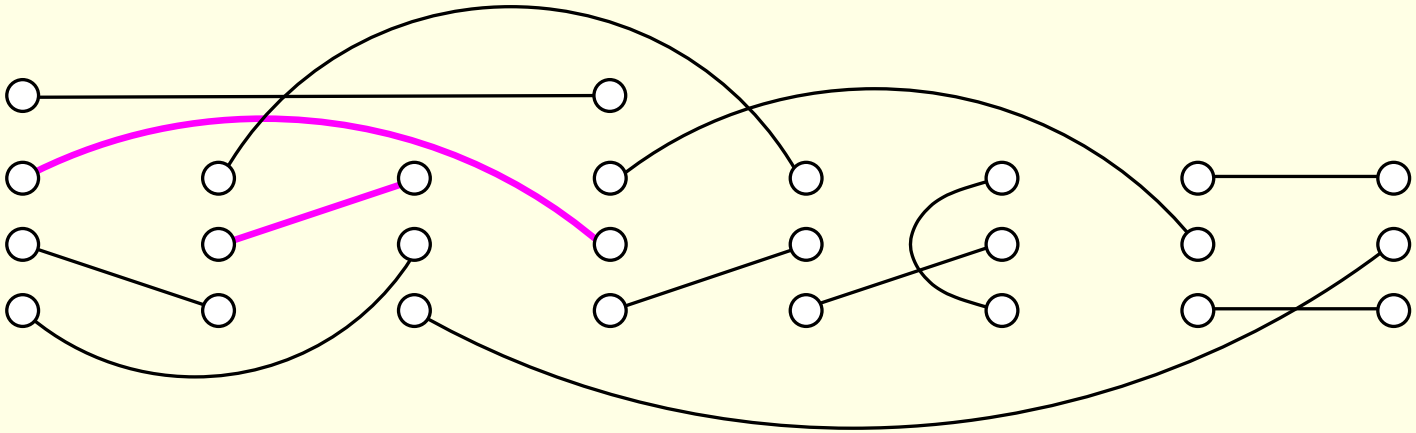
Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)



Possibly accept to get a member of $G(1,1)$.

This strategy is efficient for moderately high degree, but not for very high degree. Best results by Gao and Wormald.

Simple graphs with degrees 4,3,3,4,3,3,3,3 (continued)



Possibly accept to get a member of $G(1,1)$.

This strategy is efficient for moderately high degree, but not for very high degree. Best results by Gao and Wormald.

Nobody knows how to efficiently generate exactly uniform graphs of specified very high degrees.
For example, regular graphs of n vertices and degree $n/2$.

Switchings applied to approximate counting

A small modification such as replacing two edges by two others is called a switching.

Switchings applied to approximate counting

A small modification such as replacing two edges by two others is called a switching.

If N_1 is the average number of ways to switch a member of $G(2,1)$ to a member of $G(1,1)$, and
N_2 is the average number of ways to switch a member of $G(1,1)$ to a member of $G(2,1)$, then

$$\frac{|G(1,1)|}{|G(2,1)|} = \frac{N_1}{N_2}.$$

Switchings applied to approximate counting

A small modification such as replacing two edges by two others is called a switching.

If N_1 is the average number of ways to switch a member of $G(2,1)$ to a member of $G(1,1)$, and
N_2 is the average number of ways to switch a member of $G(1,1)$ to a member of $G(2,1)$, then

$$\frac{|G(1,1)|}{|G(2,1)|} = \frac{N_1}{N_2}.$$

Since the total number of pairings is easy to calculate, such ratios can be used to find an estimate for $|G(0,0)|$ (the number of simple graphs).

Switchings applied to approximate counting

A small modification such as replacing two edges by two others is called a switching.

If N_1 is the average number of ways to switch a member of $G(2,1)$ to a member of $G(1,1)$, and
N_2 is the average number of ways to switch a member of $G(1,1)$ to a member of $G(2,1)$, then

$$\frac{|G(1,1)|}{|G(2,1)|} = \frac{N_1}{N_2}.$$

Since the total number of pairings is easy to calculate, such ratios can be used to find an estimate for $|G(0,0)|$ (the number of simple graphs).

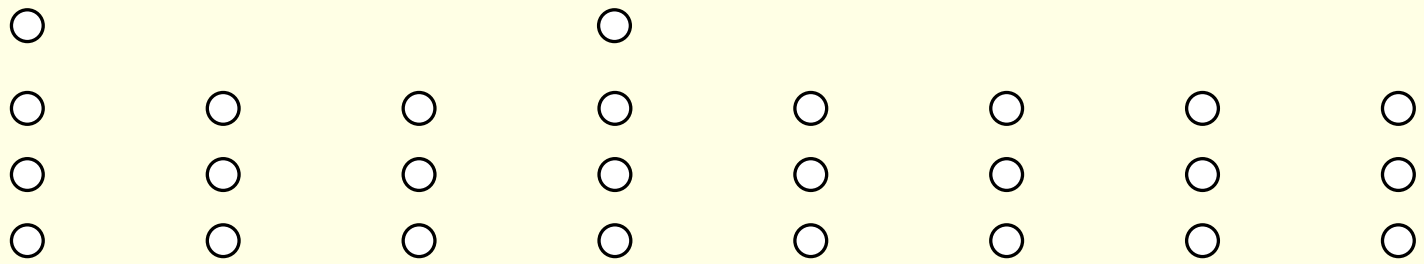
This is the most successful method for low degrees.

Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.

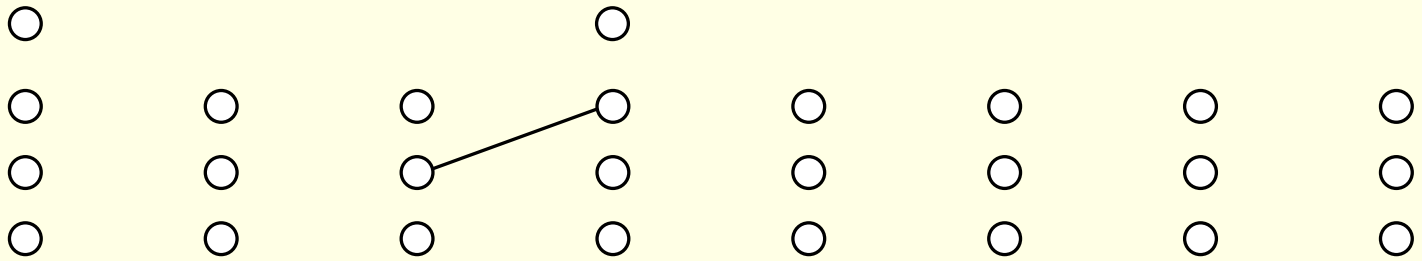
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



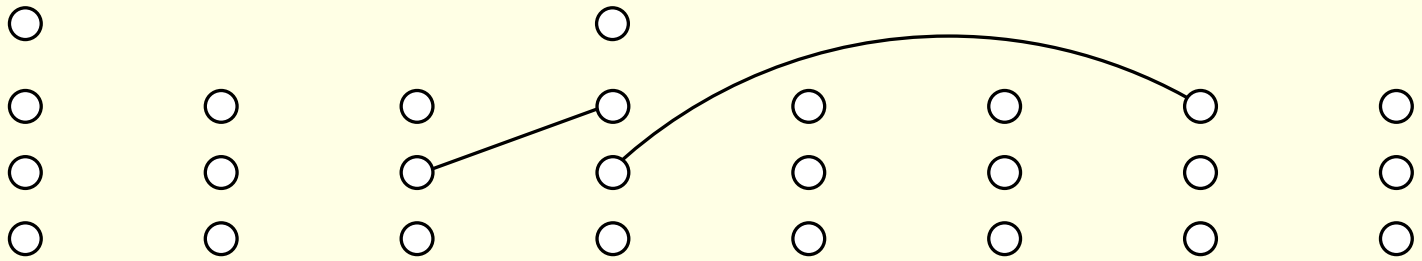
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



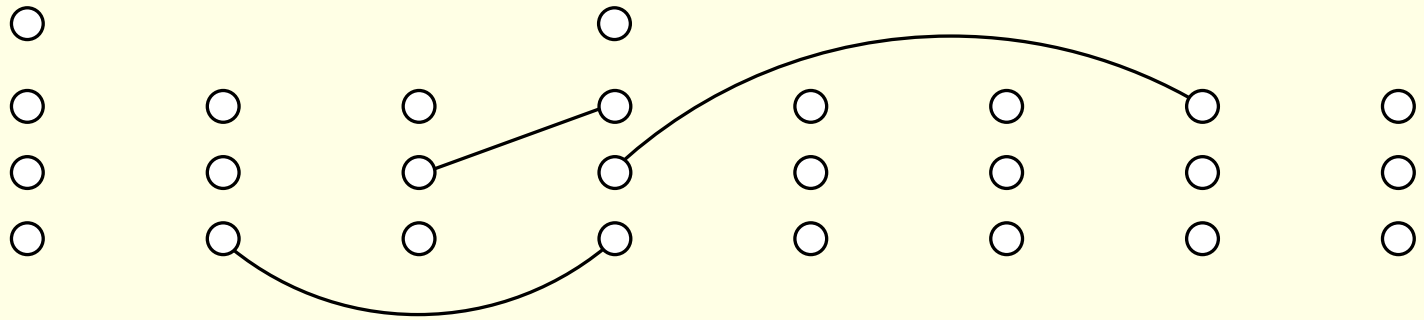
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



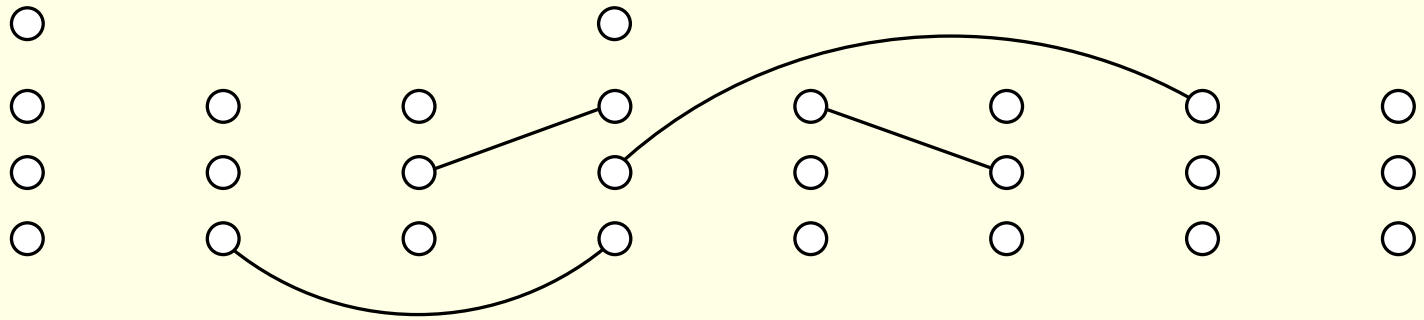
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



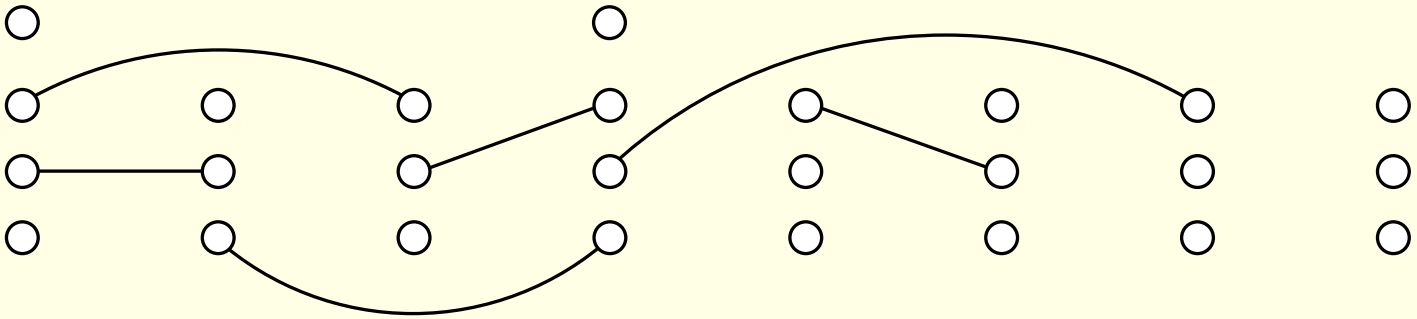
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



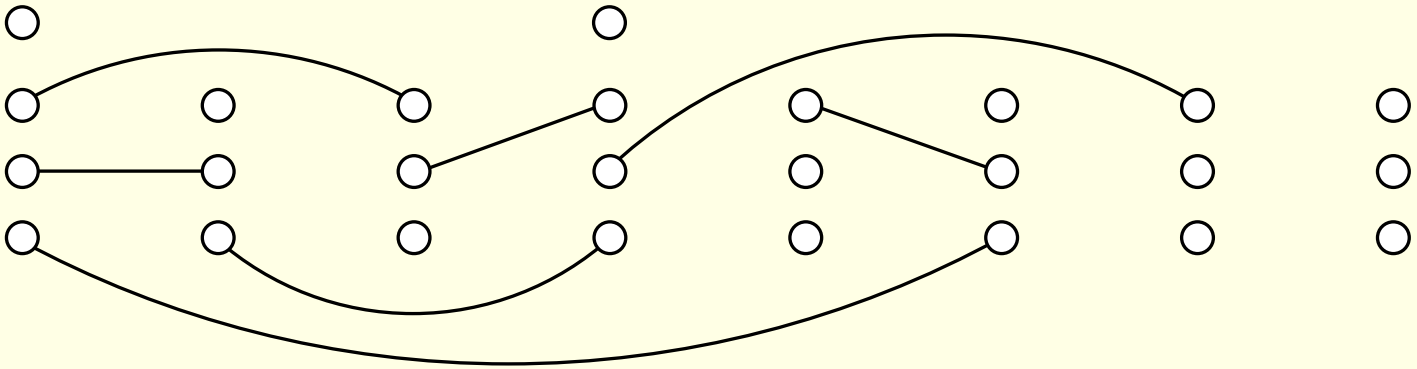
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



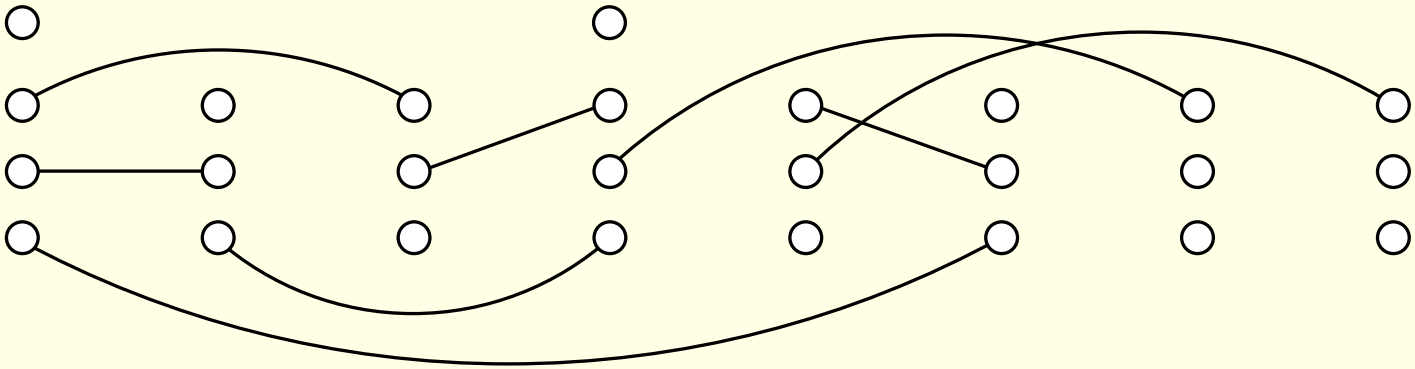
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



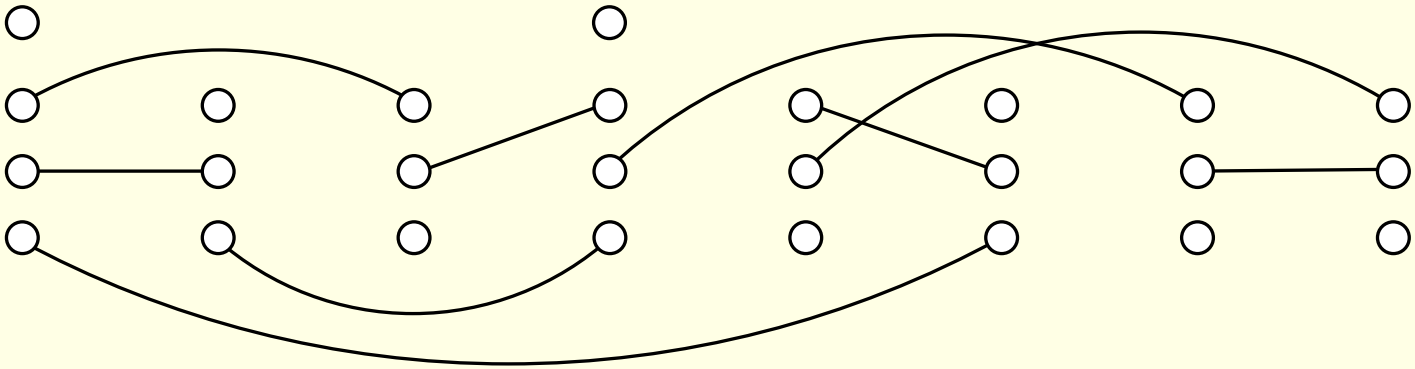
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



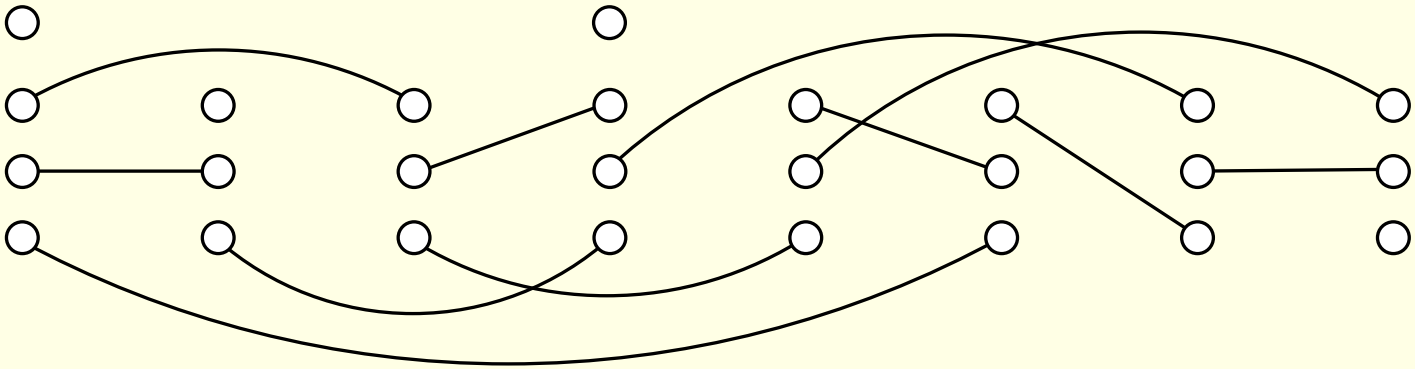
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



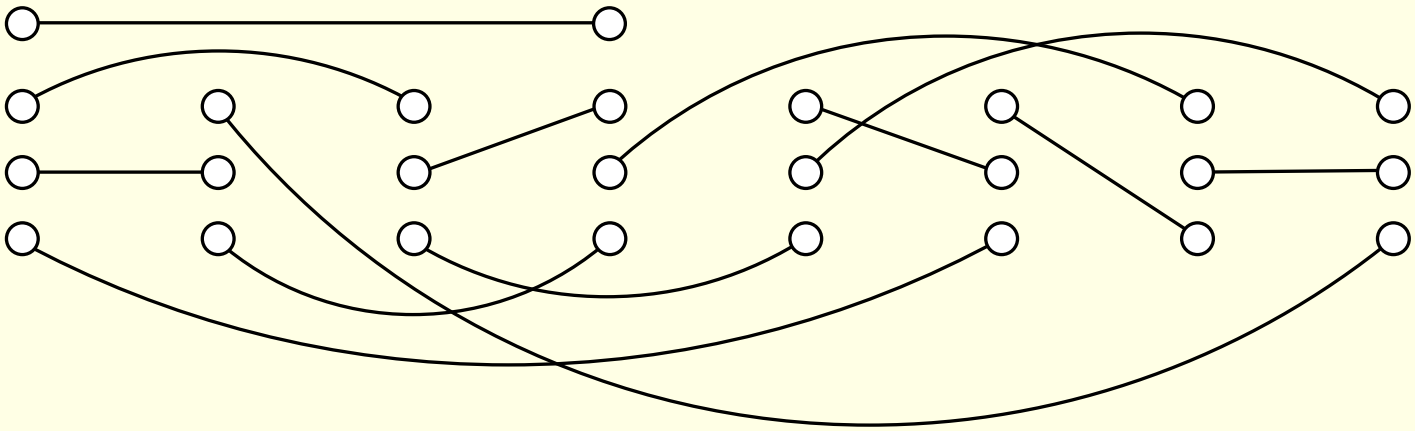
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



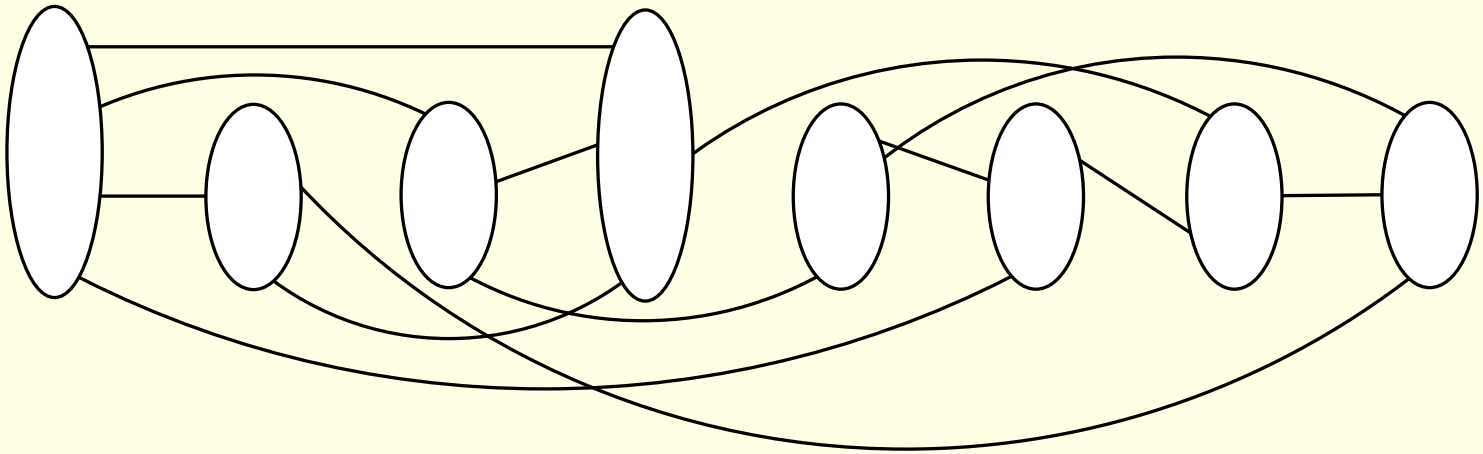
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



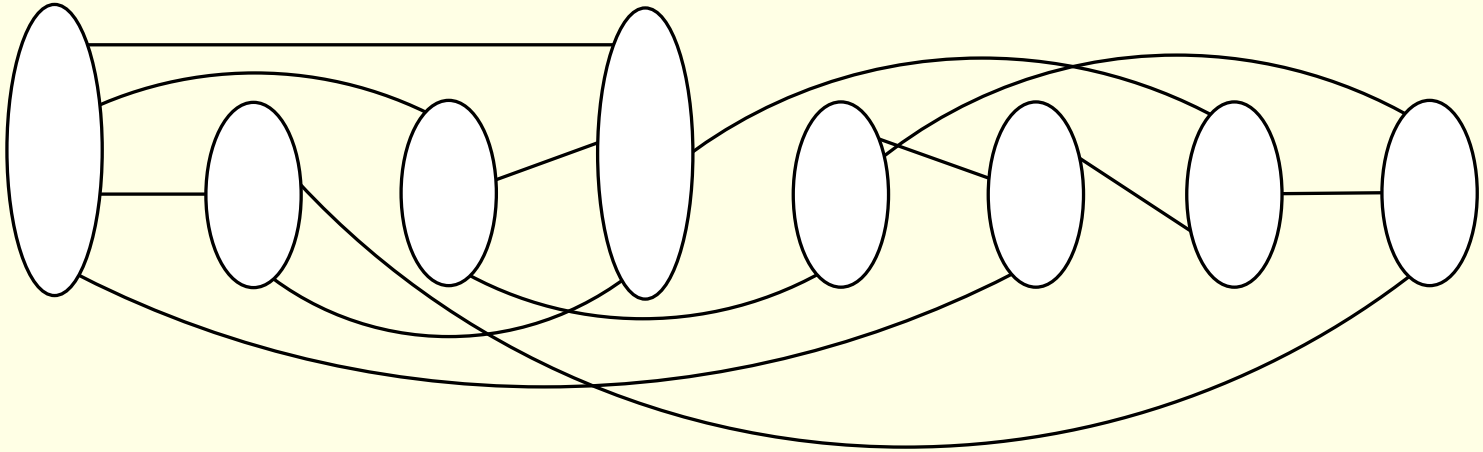
Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.

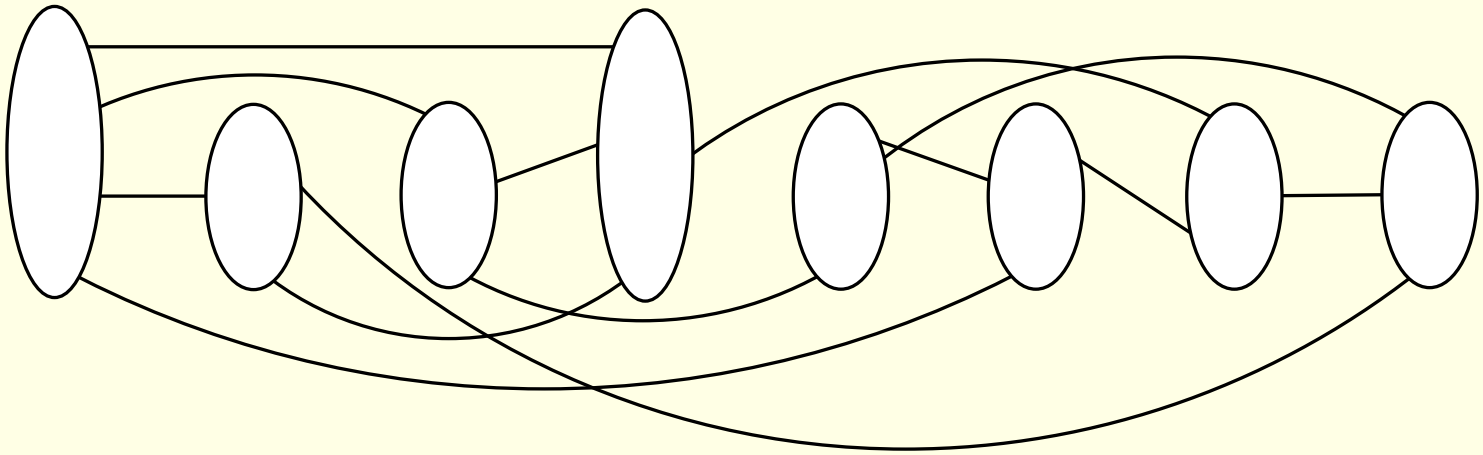


If you get stuck, start over.

This happens less often than anyone can prove.

Approximately uniform random generation — the d-model

Instead of choosing a random pairing, choose one pair at a time at random from the available legal pairs.



If you get stuck, start over.

This happens less often than anyone can prove.

Alas, the result is not uniformly random. Steger and Wormald proved it is approximately random for low degrees.

Random walks on graphs

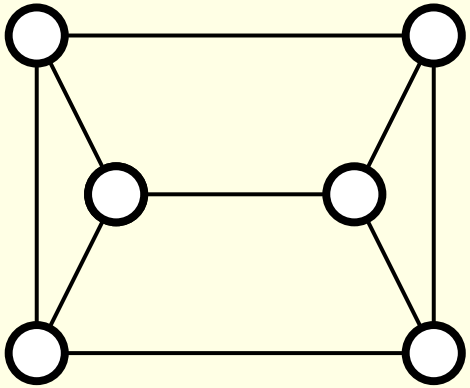
Suppose we have a connected regular graph with at least one cycle of odd length.

Start anywhere and walk at random for a long time.

Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

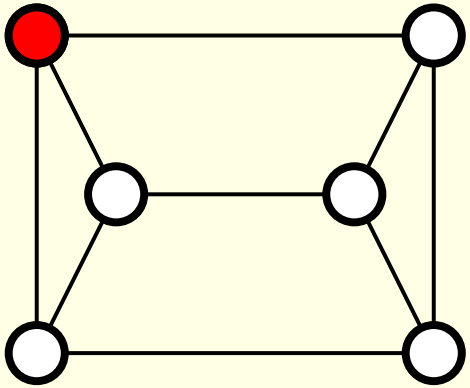
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

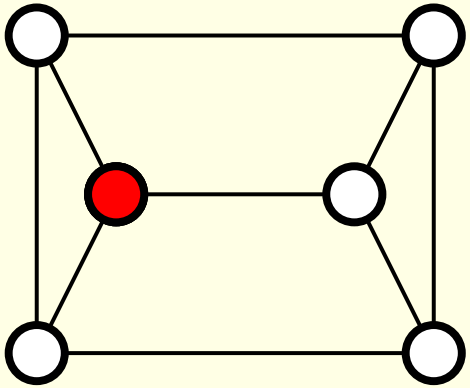
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

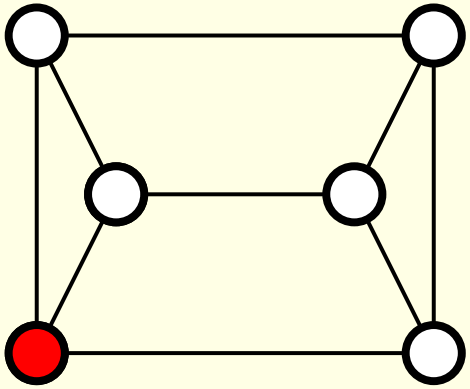
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

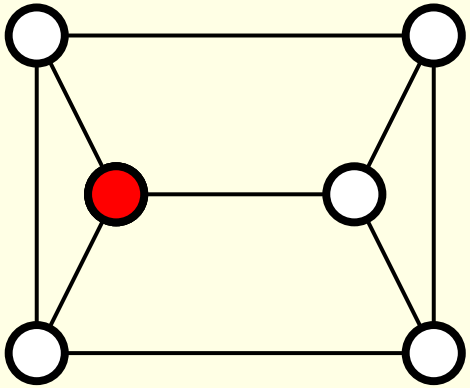
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

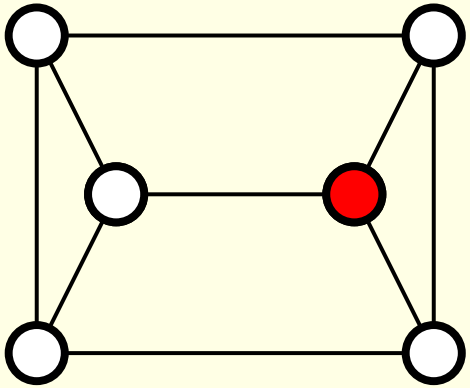
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

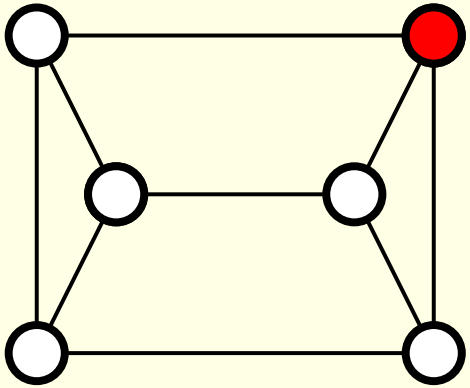
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

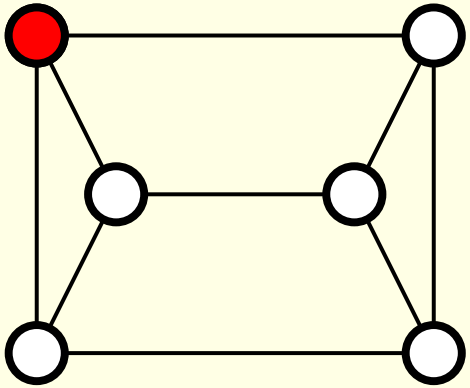
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

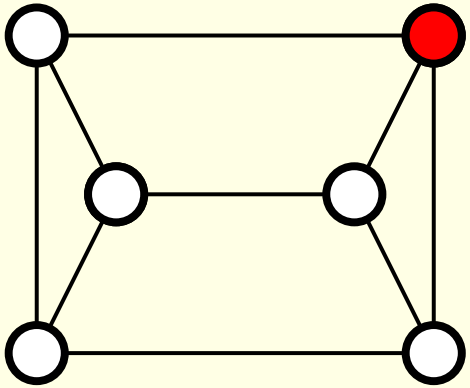
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

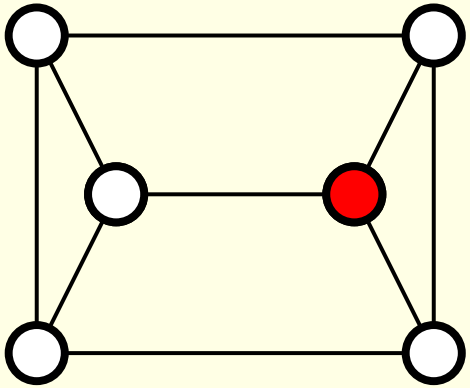
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

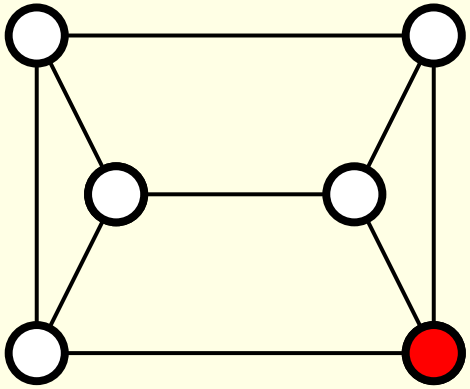
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

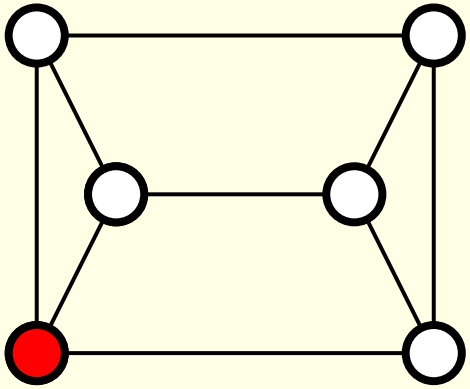
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

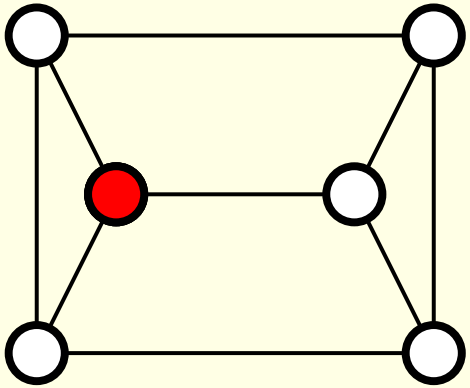
Start anywhere and walk at random for a long time.



Random walks on graphs

Suppose we have a connected regular graph with at least one cycle of odd length.

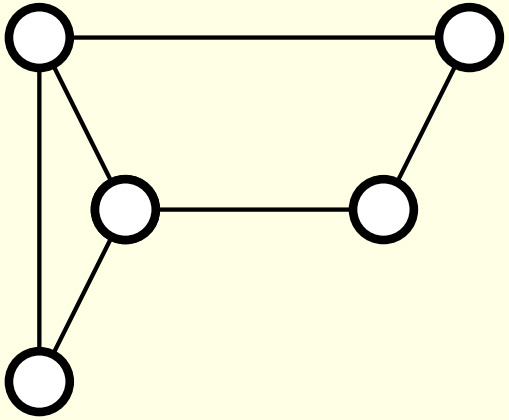
Start anywhere and walk at random for a long time.



By the theory of Markov Chains, we are equally likely to be anywhere on the graph.

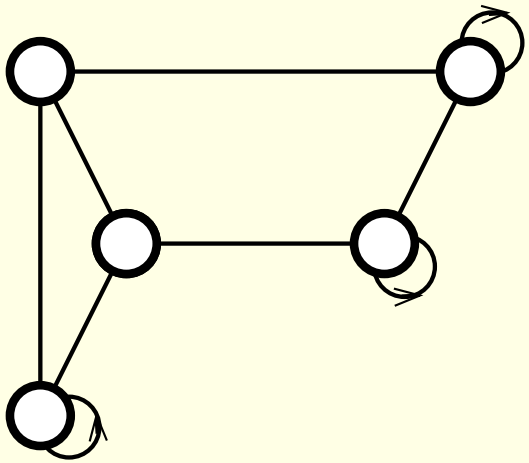
Random walks on graphs (continued)

Random walks on graphs (continued)



If the graph is not regular, the limiting distribution is not uniform.

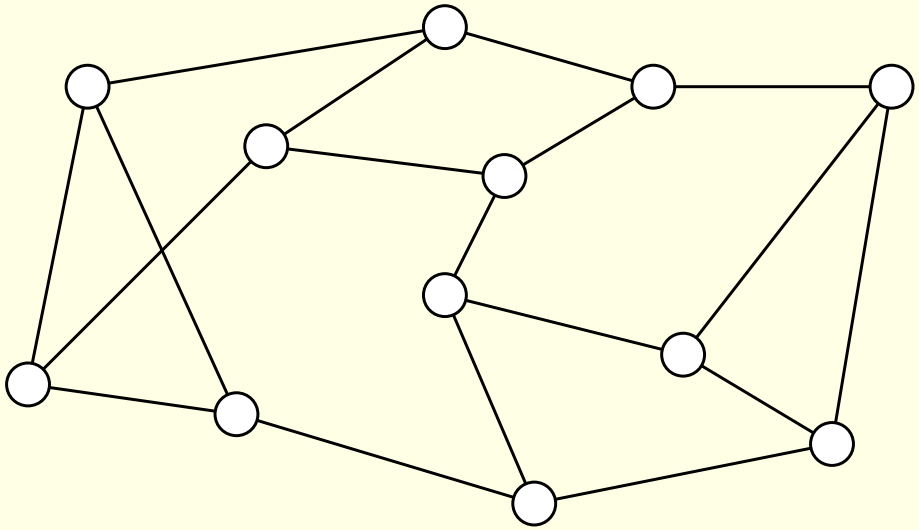
Random walks on graphs (continued)



But it can be made
regular by adding loops.

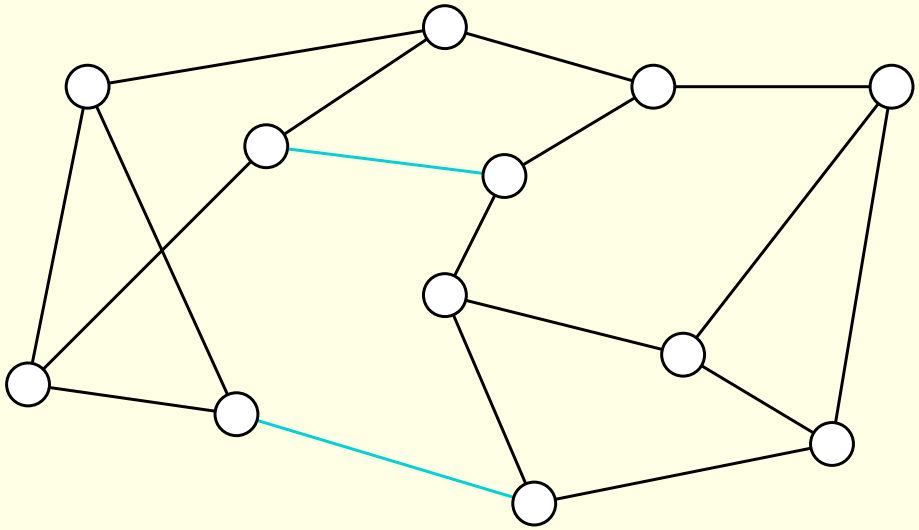
Application to random generation (eg. cubic graphs)

Application to random generation (eg. cubic graphs)



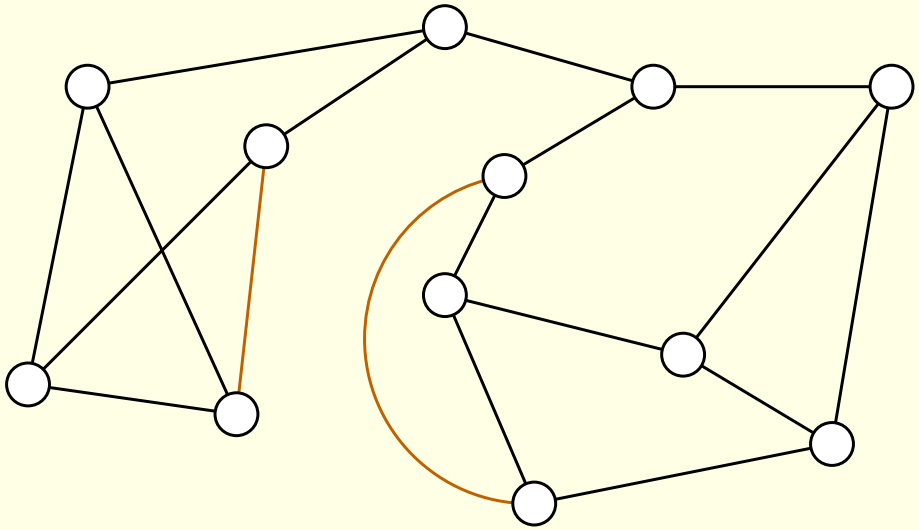
A cubic graph.

Application to random generation (eg. cubic graphs)



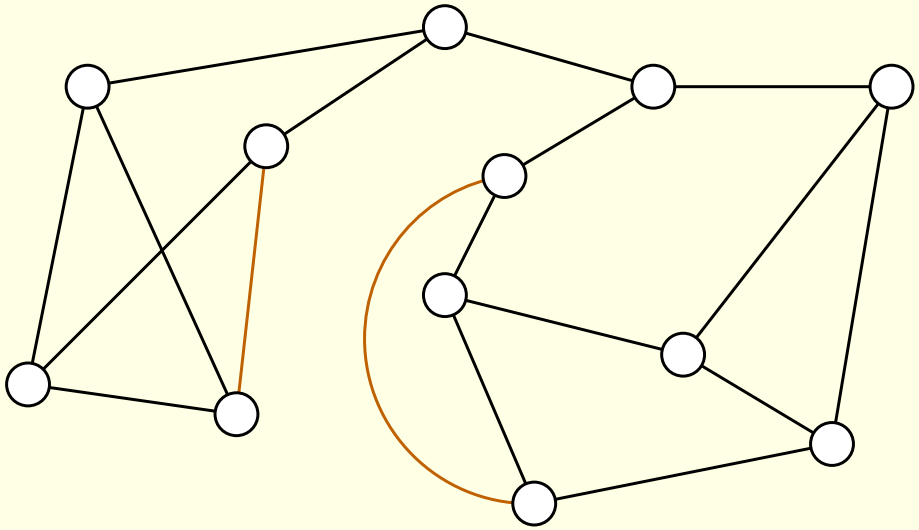
One switching.

Application to random generation (eg. cubic graphs)



One switching.

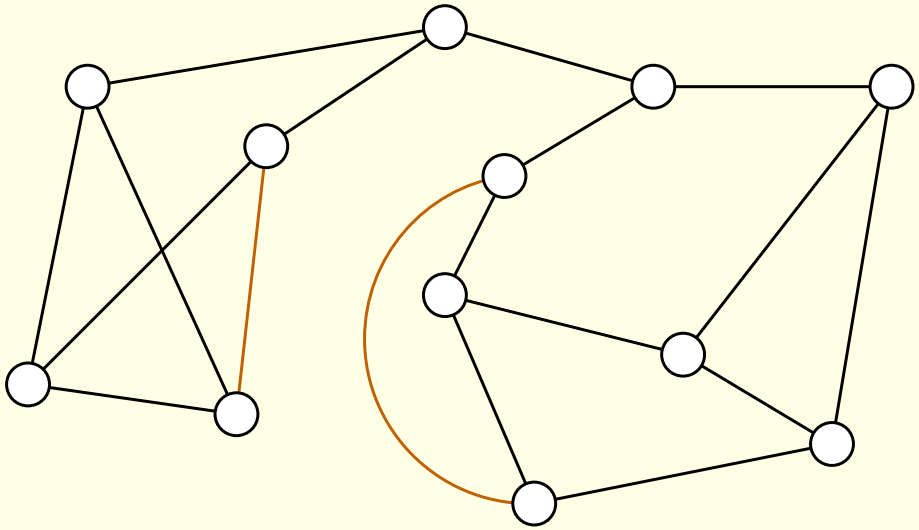
Application to random generation (eg. cubic graphs)



One switching.

Start with any cubic graph and do switchings in random places.

Application to random generation (eg. cubic graphs)

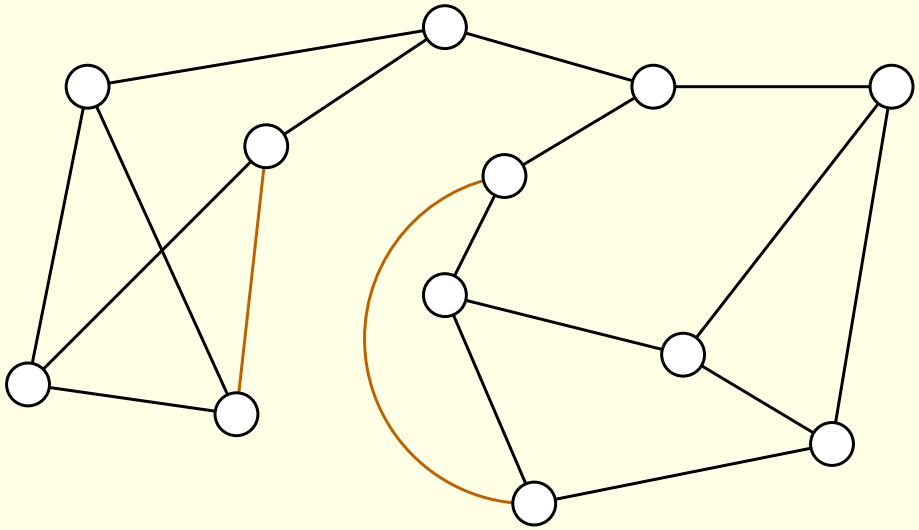


One switching.

Start with any cubic graph and do switchings in random places.

This is like walking at random on a graph whose vertices are cubic graphs.

Application to random generation (eg. cubic graphs)



One switching.

Start with any cubic graph and do switchings in random places.

This is like walking at random on a graph whose vertices are cubic graphs.

But different cubic graphs have different numbers of available switchings, so this is a walk on an irregular graph.

Markov chain for cubic graphs

- Choose any cubic graph.
- Choose a large number N.
- Do this N times:
  - Randomly select edges v_1v_2 and w_1w_2.
  - If $v_1v_2, w_1w_2 \longrightarrow v_1w_1, v_2w_2$ is a valid switching, then perform it. If not, do nothing.

Markov chain for cubic graphs

- Choose any cubic graph.
- Choose a large number N.
- Do this N times:
  - Randomly select edges v_1v_2 and w_1w_2.
  - If $v_1v_2, w_1w_2 \longrightarrow v_1w_1, v_2w_2$ is a valid switching, then perform it. If not, do nothing.

If N is very large, the result is close to a uniformly random cubic graph.

Markov chain for cubic graphs

- Choose any cubic graph.
- Choose a large number N.
- Do this N times:
  - Randomly select edges v_1v_2 and w_1w_2.
  - If $v_1v_2, w_1w_2 \longrightarrow v_1w_1, v_2w_2$ is a valid switching, then perform it. If not, do nothing.

If N is very large, the result is close to a uniformly random cubic graph.

The rate of convergence to a uniform distribution is called the mixing time and is very much studied.