Finding a Maximum Matching in a Sparse Random Graph in O(n) Expected Time

Prasad Chebolu

Joint work with Alan Frieze and Páll Melsted

Overview

- Preliminaries and previous work

Overview

- Preliminaries and previous work
- The Karp-Sipser algorithm

Overview

- Preliminaries and previous work
- The Karp-Sipser algorithm
- Our algorithm

Overview

- Preliminaries and previous work
- The Karp-Sipser algorithm
- Our algorithm
- Analysis

Introduction

Given a graph $G = (V, E)$ a *matching* is a set of vertex-disjoint edges $M \subseteq E$.

Introduction

Given a graph $G = (V, E)$ a *matching* is a set of vertex-disjoint edges $M \subseteq E$.

The endpoints of edges in M are said to be matched, others are unmatched.

Introduction

Given a graph $G = (V, E)$ a *matching* is a set of vertex-disjoint edges $M \subseteq E$.

The endpoints of edges in M are said to be matched, others are unmatched.

Maximum cardinality matching is a fundamental problem in combinatorial optimization.

Previous Work

Matchings in bipartite graphs can be found using flows. We are interested in general graphs.

Previous Work

Matchings in bipartite graphs can be found using flows. We are interested in general graphs.

Algorithms timeline

- Berge [1957] - exponential

Previous Work

Matchings in bipartite graphs can be found using flows. We are interested in general graphs.

Algorithms timeline

- Berge [1957] - exponential
- Edmonds [1965]- first poly-time algorithm $O(n^4)$

Previous Work

Matchings in bipartite graphs can be found using flows. We are interested in general graphs.

Algorithms timeline

- Berge [1957] - exponential
- Edmonds [1965]- first poly-time algorithm $O(n^4)$
- Hopcroft and Karp [1971] - $O(m\sqrt{n})$

Previous Work

Matchings in bipartite graphs can be found using flows. We are interested in general graphs.
Algorithms timeline

- Berge [1957] - exponential
- Edmonds [1965]- first poly-time algorithm $O(n^4)$
- Hopcroft and Karp [1971] - $O(m\sqrt{n})$
- Micali and Vazirani [1980] - $O(m\sqrt{n})$

Random Graphs

The random graph $G_{n,m}$ is chosen uniformly at random from all graphs with n vertices and m edges.

Random Graphs

The random graph $G_{n,m}$ is chosen uniformly at random from all graphs with n vertices and m edges.

$G_{n,m}^{\delta \geq 2}$ is chosen uniformly at random from all graphs with n vertices, m edges and minimum degree at least 2.

Random Graphs

The random graph $G_{n,m}$ is chosen uniformly at random from all graphs with n vertices and m edges.

$G_{n,m}^{\delta \geq 2}$ is chosen uniformly at random from all graphs with n vertices, m edges and minimum degree at least 2.

$G_{n,p}$ is a graph on n vertices where each edge appears independently with probability p.

Previous Work

For random graphs, we have

- Karp and Sipser [1981] - $O(n)$ time, size of matching $o(n)$ away from optimum

Previous Work

For random graphs, we have

- Karp and Sipser [1981] - $O(n)$ time, size of matching $o(n)$ away from optimum
- Motwani [1994] - $O(n \log n)$ for $p > \frac{\log n}{n}$

Previous Work

For random graphs, we have

- Karp and Sipser [1981] - $O(n)$ time, size of matching $o(n)$ away from optimum
- Motwani [1994] - $O(n \log n)$ for $p > \frac{\log n}{n}$
- Aronson, Frieze and Pittel [1998] - Karp-Sipser is $\tilde{O}(n^{1/5})$ away from optimum

Previous Work

For random graphs, we have

- Karp and Sipser [1981] - $O(n)$ time, size of matching $o(n)$ away from optimum
- Motwani [1994] - $O(n \log n)$ for $p > \frac{\log n}{n}$
- Aronson, Frieze and Pittel [1998] - Karp-Sipser is $\tilde{O}(n^{1/5})$ away from optimum
- Bast et al. [2005] - $O(n \log n)$ for sparse graphs, i.e. $p = \frac{c}{n}$

Our Work

Theorem

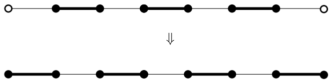
The maximum cardinality matching can be found in $O(n)$ expected time in $G_{n,m}$ where $m = c_1 n$.

Augmenting Paths

An augmenting path is a path between two unmatched vertices in a graph s.t. every other edge on the path is a matching edge.

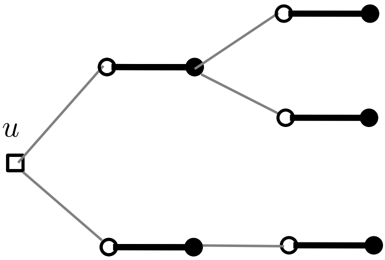
Augmenting Paths

An augmenting path is a path between two unmatched vertices in a graph s.t. every other edge on the path is a matching edge.



Augmenting Trees

Augmenting trees are rooted at an unmatched vertex, all leaves are connected with a matching edge and all paths from leaves to the root alternate between matching and nonmatching edges.

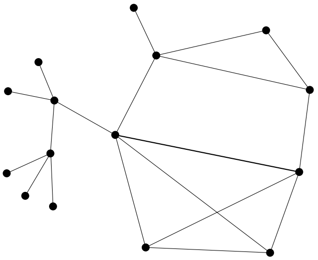


T_u

Karp-Sipser

Simple heuristic. Given a graph G

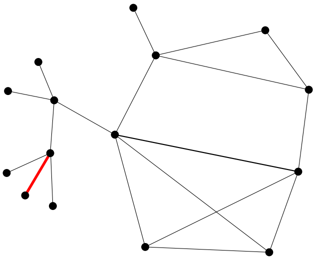
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

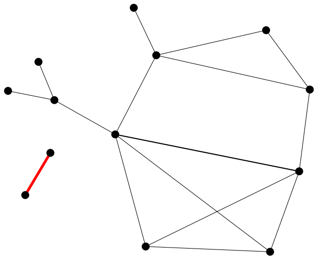
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the the matching



Karp-Sipser

Simple heuristic. Given a graph G

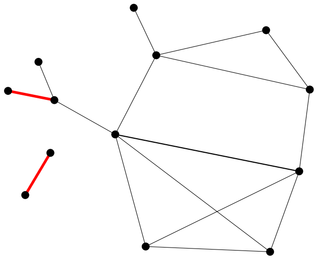
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

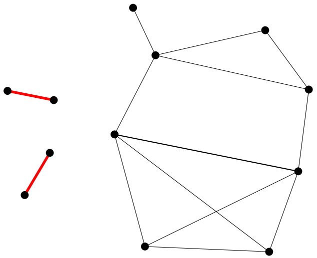
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

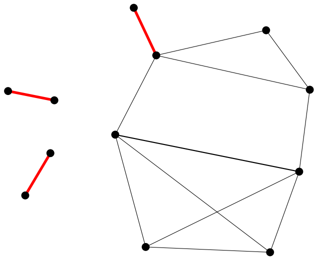
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

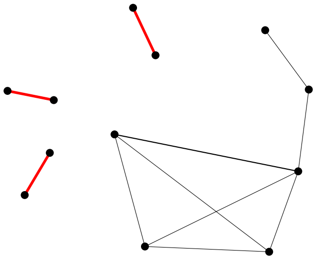
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

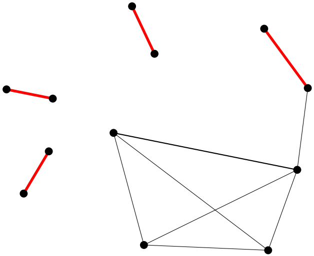
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

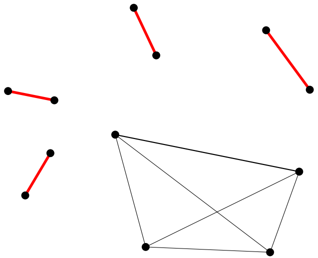
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

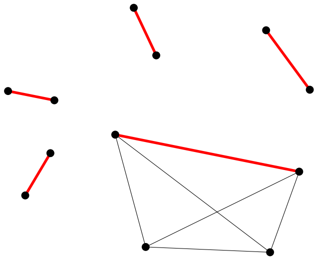
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

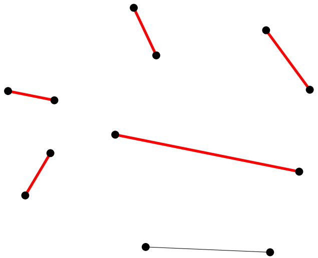
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

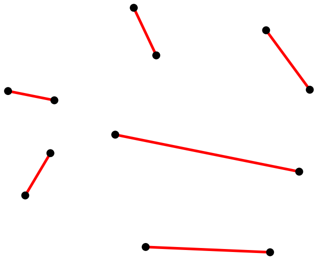
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

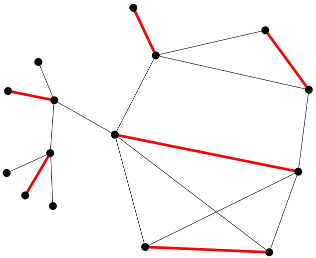
- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the matching



Karp-Sipser

Simple heuristic. Given a graph G

- If there are vertices of degree 1, pick one at random and include it in the matching
- Otherwise pick an edge at random, include it in the the matching



Karp-Sipser

We split the execution of Karp-Sipser into two phases.

We start in phase 1 and go into phase 2 when there are no vertices of degree 1 for the first time.

Karp-Sipser

We split the execution of Karp-Sipser into two phases.

We start in phase 1 and go into phase 2 when there are no vertices of degree 1 for the first time.

The Karp-Sipser algorithm makes no mistakes in phase 1.

Previous work on KS

Previous work on KS

Karp and Sipser (1981): KS on $G_{n,cn/2}$:

- If $c \leq e$ then **whp** at the end of Phase 1, G has only $o(n)$ vertices
- If $c > e$ then **whp** Phase 2 leaves $o(n)$ vertices unmatched

Previous work on KS

Karp and Sipser (1981): KS on $G_{n,cn/2}$:

- If $c \leq e$ then **whp** at the end of Phase 1, G has only $o(n)$ vertices
- If $c > e$ then **whp** Phase 2 leaves $o(n)$ vertices unmatched

Aronson, Frieze and Pittel (1998):

- If $c < e$ then **whp** at the end of Phase 1, G consists of disjoint cycles
- If $c > e$ then **whp** Phase 2 leaves $\tilde{O}(n^{1/5})$ vertices unmatched

Previous work on KS

Karp and Sipser (1981): KS on $G_{n,cn/2}$:

- If $c \leq e$ then **whp** at the end of Phase 1, G has only $o(n)$ vertices
- If $c > e$ then **whp** Phase 2 leaves $o(n)$ vertices unmatched

Aronson, Frieze and Pittel (1998):

- If $c < e$ then **whp** at the end of Phase 1, G consists of disjoint cycles
- If $c > e$ then **whp** Phase 2 leaves $\tilde{O}(n^{1/5})$ vertices unmatched

Frieze and Pittel (2004):
If $c > e$ then **whp** the graph at the end of Phase 2 has a perfect matching modulo isolated odd cycles.

Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G,
  find an augmenting path by growing augmenting trees from both vertices.

Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G, find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G,
  find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

- KS finds a matching in G with $\tilde{O}(n^{.2})$ unmatched vertices

Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G,
  find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

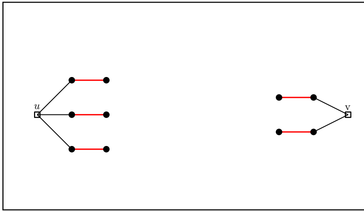
- KS finds a matching in G with $\tilde{O}(n^{.2})$ unmatched vertices
- G has a perfect matching modulo isolated odd cycles

Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G,
  find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

- KS finds a matching in G with $\tilde{O}(n^{.2})$ unmatched vertices
- G has a perfect matching modulo isolated odd cycles

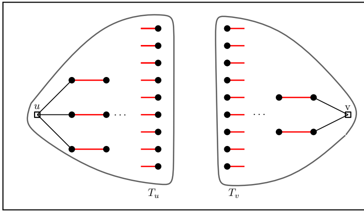


Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G, find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

- KS finds a matching in G with $\tilde{O}(n^{.2})$ unmatched vertices
- G has a perfect matching modulo isolated odd cycles
- Augmenting trees grow by a constant factor per level

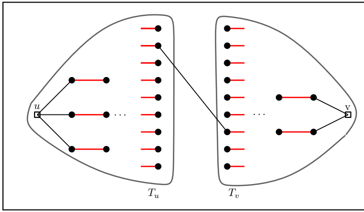


Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G,
  find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

- KS finds a matching in G with $\tilde{O}(n^{.2})$ unmatched vertices
- G has a perfect matching modulo isolated odd cycles
- Augmenting trees grow by a constant factor per level
- Once trees are of size $\sqrt{n \log n}$, they should connect across

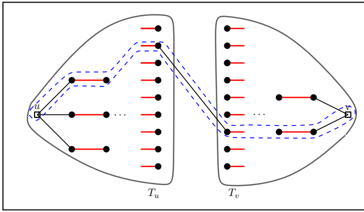


Linear expected time algorithm:

- Run KS to find initial matching. Let G be the graph at the end of Phase 2
- For every two unmatched vertices in G, find an augmenting path by growing augmenting trees from both vertices.

Why should this work?

- KS finds a matching in G with $\tilde{O}(n^{.2})$ unmatched vertices
- G has a perfect matching modulo isolated odd cycles
- Augmenting trees grow by a constant factor per level
- Once trees are of size $\sqrt{n \log n}$, they should connect across yielding an augmenting path



What could go wrong?

What could go wrong?

- Trees might not grow to a large size

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$ will expand by a constant factor

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles
- Large trees might not have edges connecting them

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$ will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles
- Large trees might not have edges connecting them
  - This is very unlikely if we have randomness

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles
- Large trees might not have edges connecting them
  - This is very unlikely if we have randomness
  - How much randomness do we have?

How much randomness is there?

How much randomness is there?

Initially G is random and then we run KS it sees the entire graph.

How much randomness is there?

Initially G is random and then we run KS it sees the entire graph.
Is there any randomness left?

How much randomness is there?

Initially G is random and then we run KS it sees the entire graph.
Is there any randomness left?

We condition on the output of KS

How much randomness is there?

Initially G is random and then we run KS it sees the entire graph.
Is there any randomness left?

We condition on the output of KS and designate certain edges as witnesses.

How much randomness is there?

Initially G is random and then we run KS it sees the entire graph.
Is there any randomness left?

We condition on the output of KS and designate certain edges as witnesses.

All other edges will not affect the course of the algorithm. They are our source of randomness.

How much randomness is there?

Initially G is random and then we run KS it sees the entire graph.
Is there any randomness left?

We condition on the output of KS and designate certain edges as witnesses.

All other edges will not affect the course of the algorithm. They are our source of randomness.

Analysis - KS conditioning

We have two sources of randomness, in the input and in the algorithm.

Analysis - KS conditioning

We have two sources of randomness, in the input and in the algorithm.

We can get around this by assuming the edges come with an ordering and the algorithm always picks the first available edge. From now we assume the input is an ordered set of edges.

Analysis - KS conditioning

We have two sources of randomness, in the input and in the algorithm.

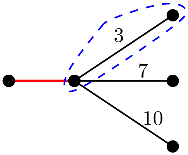
We can get around this by assuming the edges come with an ordering and the algorithm always picks the first available edge. From now we assume the input is an ordered set of edges.

When vertices are removed from the graph they either have degree ≥ 2, 1 or 0. We refer to them as regular, pendant and isolated.

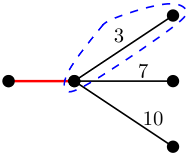
Witness edges

Witness edges

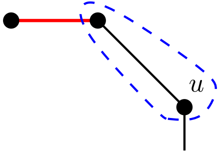
If KS removes a regular vertex, the first edge incident to it becomes its regular witness edge.
It ensures this vertex had degree at least 2



If KS removes a regular vertex, the first edge incident to it becomes its regular witness edge.
It ensures this vertex had degree at least 2

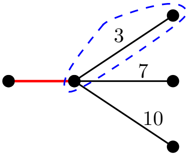


If KS removes an edge such that one of its endpoints becomes pendant, this edge becomes a pendant witness edge.
It ensures this vertex had degree 2 before that time

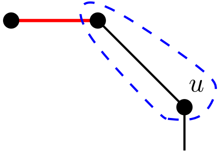


Witness edges

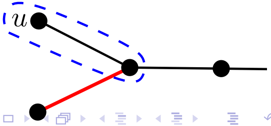
If KS removes a regular vertex, the first edge incident to it becomes its regular witness edge.
It ensures this vertex had degree at least 2



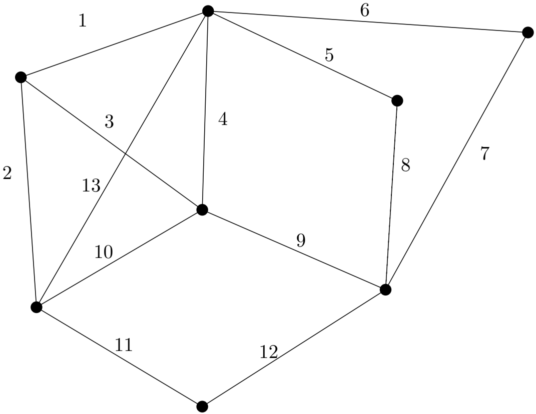
If KS removes an edge such that one of its endpoints becomes pendant, this edge becomes a pendant witness edge.
It ensures this vertex had degree 2 before that time



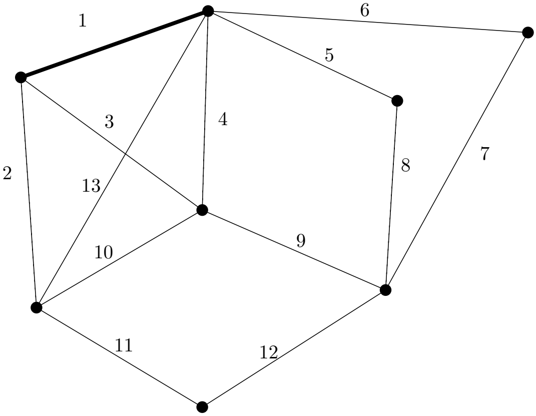
If KS removes an edge such that one of its endpoints becomes isolated, this edge is an isolated witness edge. It ensures this vertex was not isolated before that time



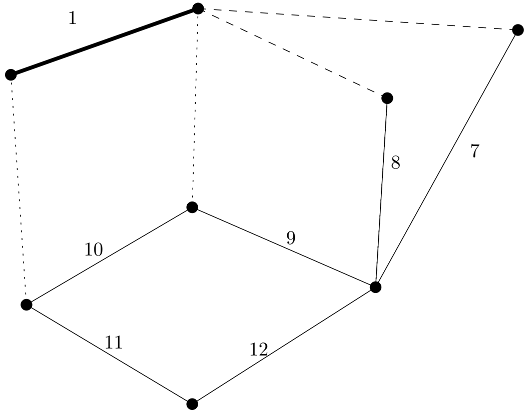
Analysis - KS conditioning



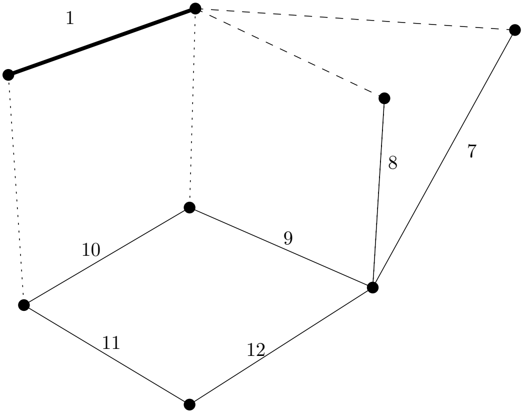
Analysis - KS conditioning



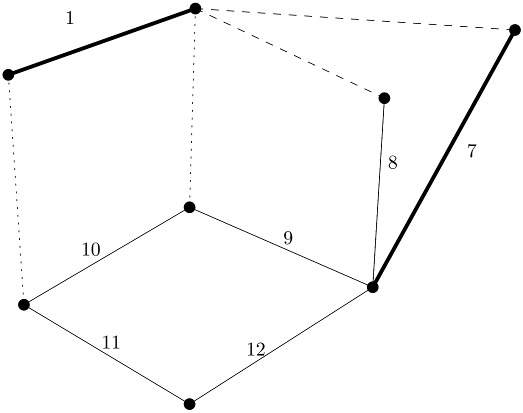
Analysis - KS conditioning



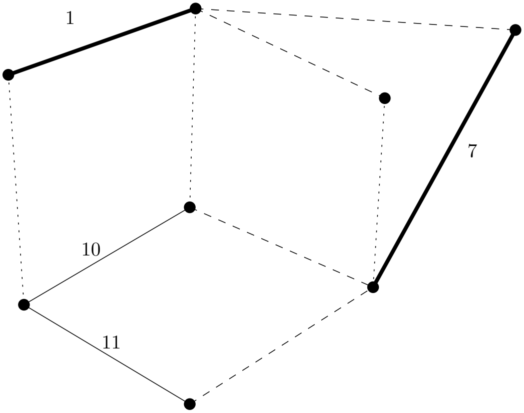
Analysis - KS conditioning



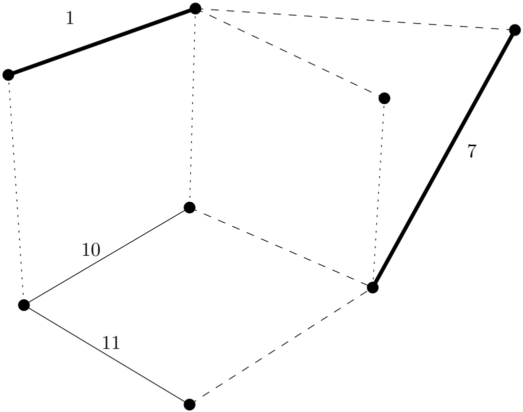
Analysis - KS conditioning



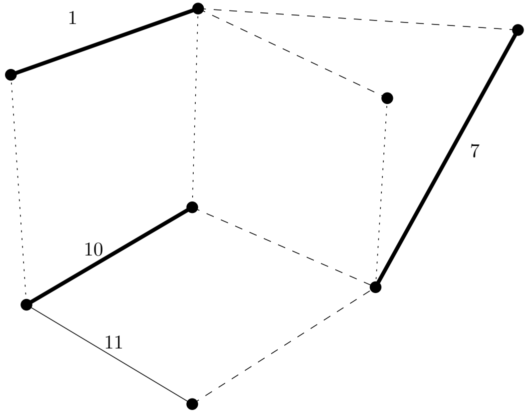
Analysis - KS conditioning



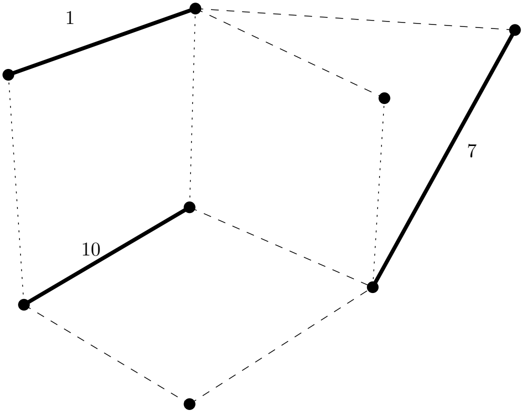
Analysis - KS conditioning



Analysis - KS conditioning



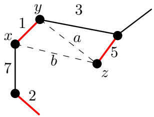
Analysis - KS conditioning



Conditioning on the matching and witness edges we view G as an ordered list of boxes containing edges, of which some have

been revealed.

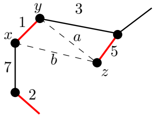
Conditioning on the matching and witness edges we view G as an ordered list of boxes containing edges, of which some have



M	M	W	?	M	?	W	?	⋯
1	2	3	4	5	6	7	8	

been revealed.

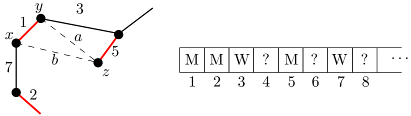
Conditioning on the matching and witness edges we view G as an ordered list of boxes containing edges, of which some have



M	M	W	?	M	?	W	?	...
1	2	3	4	5	6	7	8	

been revealed.
a can go into boxes 4,6 and 8

Conditioning on the matching and witness edges we view G as an ordered list of boxes containing edges, of which some have

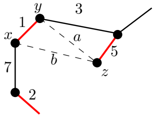


been revealed.

a can go into boxes 4,6 and 8

b cannot go into boxes 4 or 6, since edge 7 would not be a witness edge for x

Conditioning on the matching and witness edges we view G as an ordered list of boxes containing edges, of which some have



M	M	W	?	M	?	W	?	$\cdots$
1	2	3	4	5	6	7	8	

been revealed.

a can go into boxes 4,6 and 8

b cannot go into boxes 4 or 6, since edge 7 would not be a witness edge for x , but b can go into box 8

We can put an edge between a regular vertex x and any vertex y as long as

We can put an edge between a regular vertex x and any vertex y as long as

- the edge is placed in a box after the witness edge of x

We can put an edge between a regular vertex x and any vertex y as long as

- the edge is placed in a box after the witness edge of x
- y is removed from the graph after x is removed

We can put an edge between a regular vertex x and any vertex y as long as

- the edge is placed in a box after the witness edge of x
- y is removed from the graph after x is removed
- y has degree at least 2 when x is removed

We can put an edge between a regular vertex x and any vertex y as long as

- the edge is placed in a box after the witness edge of x
- y is removed from the graph after x is removed
- y has degree at least 2 when x is removed

Some edges can appear in more boxes than others.

We can put an edge between a regular vertex x and any vertex y as long as

- the edge is placed in a box after the witness edge of x
- y is removed from the graph after x is removed
- y has degree at least 2 when x is removed

Some edges can appear in more boxes than others.

If an edge can go into an open box, it can go into any box that comes after it.

We can put an edge between a regular vertex x and any vertex y as long as

- the edge is placed in a box after the witness edge of x
- y is removed from the graph after x is removed
- y has degree at least 2 when x is removed

Some edges can appear in more boxes than others.

If an edge can go into an open box, it can go into any box that comes after it.

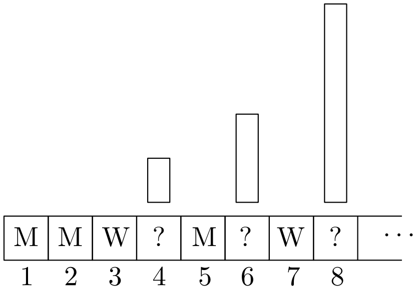
Whether an edge can go into a box or not depends only on the matching and witness edges.

This allows us to sample the random graph, conditioned on the output of KS

M	M	W	?	M	?	W	?	$\cdots$
1	2	3	4	5	6	7	8	

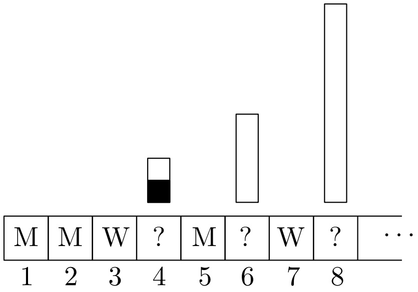
This allows us to sample the random graph, conditioned on the output of KS

- For each open box, create a list of potential edges



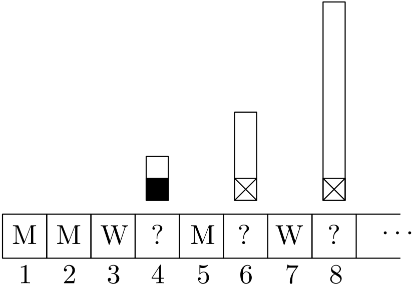
This allows us to sample the random graph, conditioned on the output of KS

- For each open box, create a list of potential edges
- Starting with the first box, pick an edge from the list at random



This allows us to sample the random graph, conditioned on the output of KS

- For each open box, create a list of potential edges
- Starting with the first box, pick an edge from the list at random
- Remove the edge chosen from all the remaining lists



What could go wrong?

What could go wrong?

- Trees might not grow to a large size

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles

What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$ will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles
- Large trees might not have edges connecting them

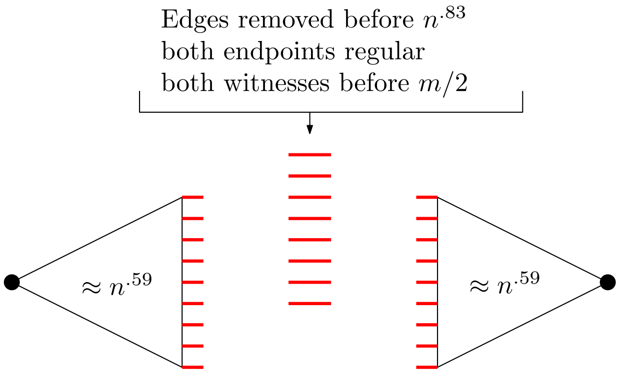
What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles
- Large trees might not have edges connecting them
  - This is very unlikely if we have randomness

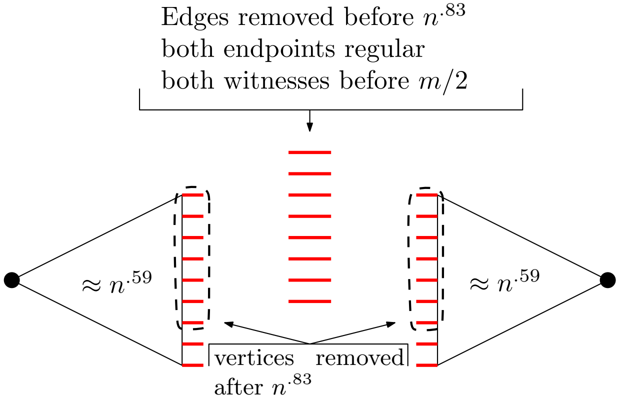
What could go wrong?

- Trees might not grow to a large size
  - **Whp** trees larger than $\Omega\left(\frac{\log n}{c}\right)$ and smaller than $n^{0.99}$ will expand by a constant factor
  - Trees smaller than $O\left(\frac{\log n}{\log c}\right)$ will grow, unless there are two short cycles connected by a short path
  - **Whp** $G_{n,m}$ has no such cycles
- Large trees might not have edges connecting them
  - This is very unlikely if we have randomness
  - How much randomness do we have?

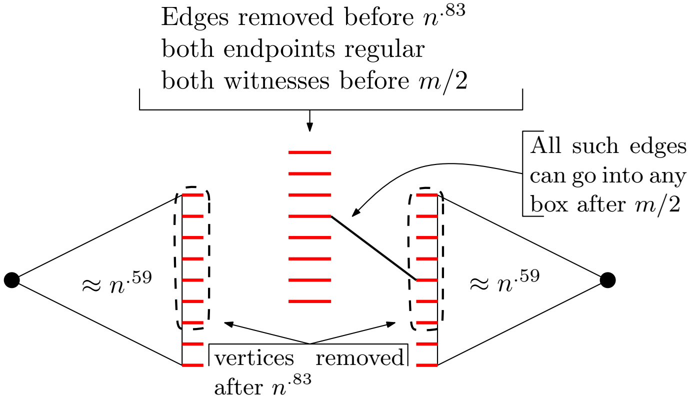
Start with large augmenting trees



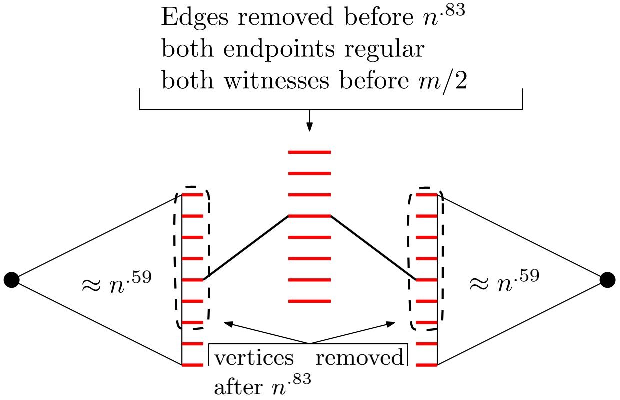
Start with large augmenting trees
Good edges can help us connect across



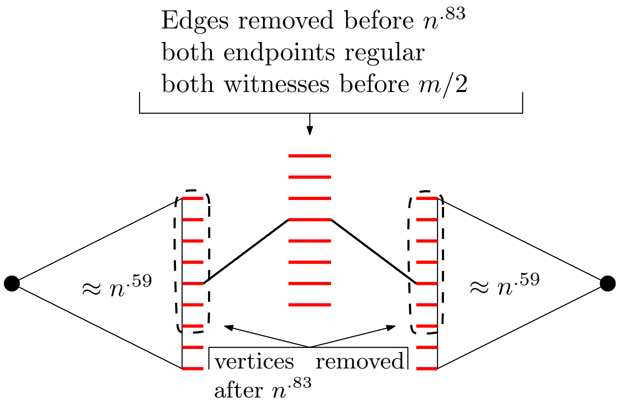
Start with large augmenting trees
Good edges can help us connect across



Start with large augmenting trees
Good edges can help us connect across
Such pairs of edges ensure the algorithm finds an augmenting path after two rounds



Start with large augmenting trees
Good edges can help us connect across
Such pairs of edges ensure the algorithm finds an augmenting path after two rounds



We have skipped a few technical details here.

Since the trees are at most $O(n^{.79})$ in size

Since the trees are at most $O(n^{.79})$ in size and we repeat this $\tilde{O}(n^{.2})$ times this implies a $o(n)$ running time for finding augmenting paths.

Since the trees are at most $O(n^{.79})$ in size and we repeat this $\tilde{O}(n^{.2})$ times this implies a $o(n)$ running time for finding augmenting paths.

The running time is dominated by KS which runs in linear time.

Since the trees are at most $O(n^{.79})$ in size and we repeat this $\tilde{O}(n^{.2})$ times this implies a $o(n)$ running time for finding augmenting paths.

The running time is dominated by KS which runs in linear time. Actually we have to be more careful in conserving randomness

when we repeat this $\tilde{O}(n^{.2})$ times. Consecutive iterations are not independent.

Since the trees are at most $O(n^{.79})$ in size and we repeat this $\tilde{O}(n^{.2})$ times this implies a $o(n)$ running time for finding augmenting paths.

The running time is dominated by KS which runs in linear time. Actually we have to be more careful in conserving randomness

when we repeat this $\tilde{O}(n^{.2})$ times. Consecutive iterations are not independent.

Still, this can be shown to take no more than $o(n)$ time.

Thank you