Faster minimum k-cut of a simple graph

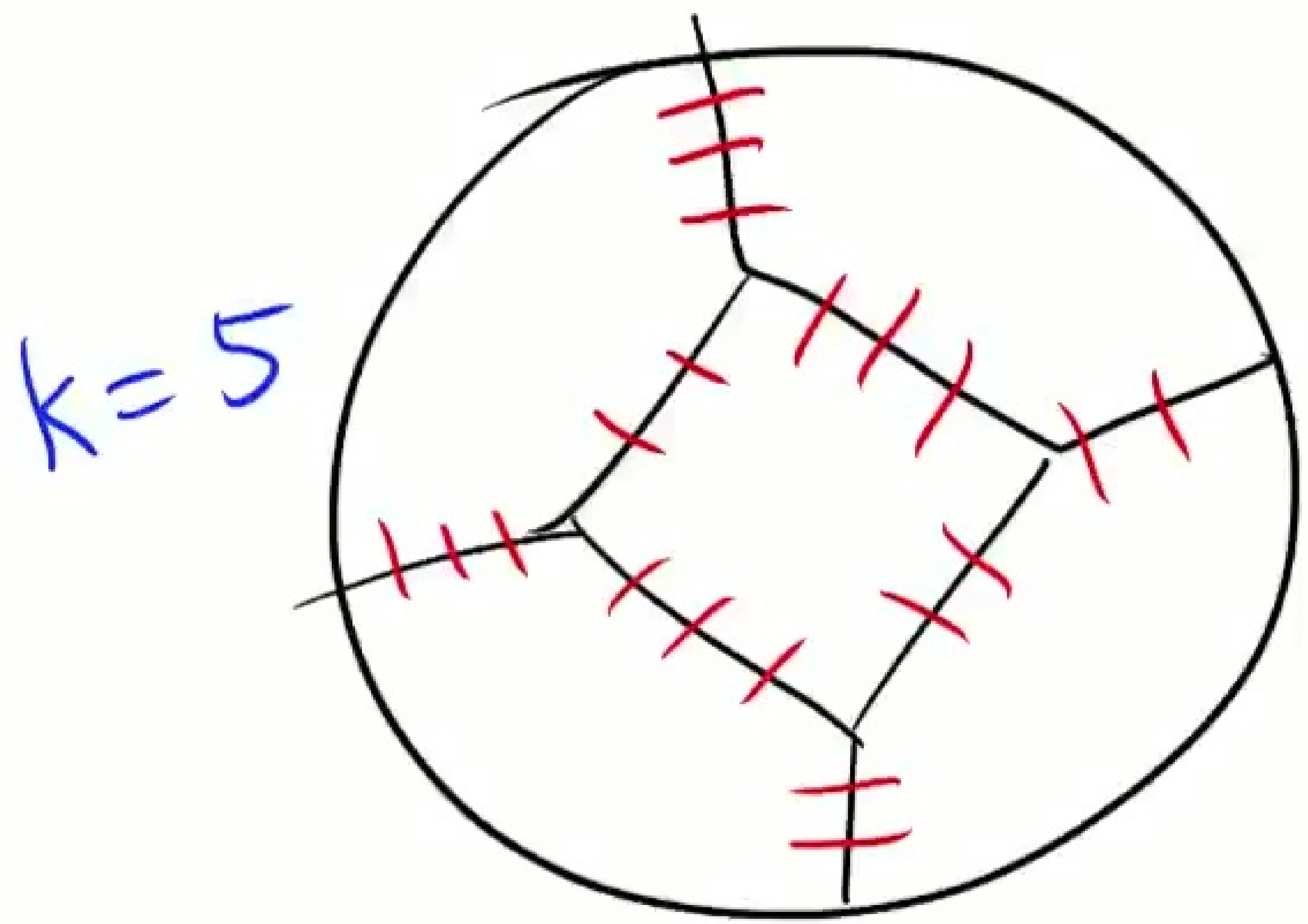
Jason Li

STOC 2019
November 11, 2019

Introduction

minimum k-cut: delete min weight edges to cut graph into $\geq k$ connected components

Setting: exact algorithm, k constant



Prior Work

Goldschmidt–Hochbaum 1994: $O(n^{(1/2-o(1))k^2})$ time deterministic

Karger–Stein 1994: $\tilde{O}(n^{2(k-1)})$ time randomized

Thorup 2008: $\tilde{O}(mn^{2k-2})$ time deterministic

Chekuri et al. 2018: $\tilde{O}(mn^{2k-3})$ time deterministic

Prior Work

Goldschmidt–Hochbaum 1994:	$O(n^{(1/2-o(1))k^2})$ time deterministic
Karger–Stein 1994:	$\tilde{O}(n^{2(k-1)})$ time randomized
Thorup 2008:	$\tilde{O}(mn^{2k-2})$ time deterministic
Chekuri et al. 2018:	$\tilde{O}(mn^{2k-3})$ time deterministic
Gupta, Lee, L. 2019:	$O_k(n^{(1.981+o(1))k})$ time randomized
Same authors, 2018:	$O_k(n^{(2\omega/3+o(1))k})$ time deterministic, integer weights $\leq n^{O(1)}$
This work:	$O_k(n^{(1+o(1))k})$ time randomized for simple graphs (unweighted, no multi-edges)

Prior Work

Goldschmidt–Hochbaum 1994: $O(n^{(1/2-o(1))k^2})$ time deterministic

Karger–Stein 1994: $\tilde{O}(n^{2(k-1)})$ time randomized

Thorup 2008: $\tilde{O}(mn^{2k-2})$ time deterministic

Chekuri et al. 2018: $\tilde{O}(mn^{2k-3})$ time deterministic

Gupta, Lee, L. 2019: $O_k(n^{(1.981+o(1))k})$ time randomized

Same authors, 2018: $O_k(n^{(2\omega/3+o(1))k})$ time deterministic, integer weights $\leq n^{O(1)}$

This work: $O_k(n^{(1+o(1))k})$ time randomized for simple graphs (unweighted, no multi-edges)

Lower bound: as hard as k-clique: $\Omega(n^{(\omega/3-o(1))k})$ time algebraic

$\Omega(n^{(1-o(1))k})$ time combinatorial

Prior Work

Goldschmidt–Hochbaum 1994: $O(n^{(1/2-o(1))k^2})$ time deterministic

Karger–Stein 1994: $\tilde{O}(n^{2(k-1)})$ time randomized

Thorup 2008: $\tilde{O}(mn^{2k-2})$ time deterministic

Chekuri et al. 2018: $\tilde{O}(mn^{2k-3})$ time deterministic

Gupta, Lee, L. 2019: $O_k(n^{(1.981+o(1))k})$ time randomized

Same authors, 2018: $O_k(n^{(2\omega/3+o(1))k})$ time deterministic, integer weights $\leq n^{O(1)}$

This work: $O_k(n^{(1+o(1))k})$ time randomized for simple graphs

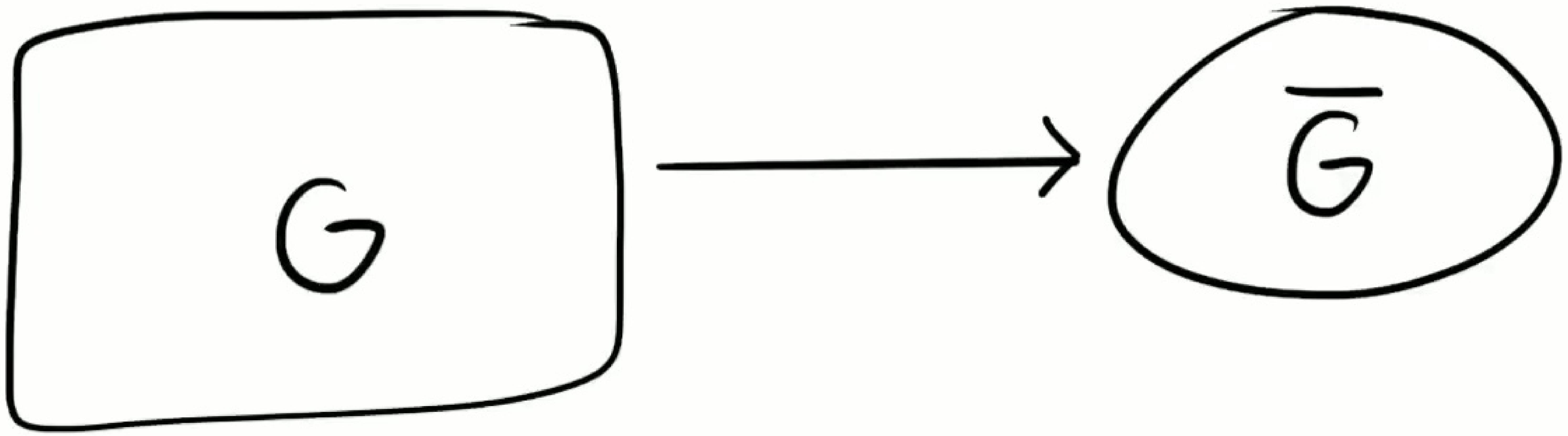
combinatorial

(unweighted, no multi-edges)

Lower bound: as hard as k-clique: $\Omega(n^{(\omega/3-o(1))k})$ time algebraic

$\Omega(n^{(1-o(1))k})$ time combinatorial

Algorithm Outline



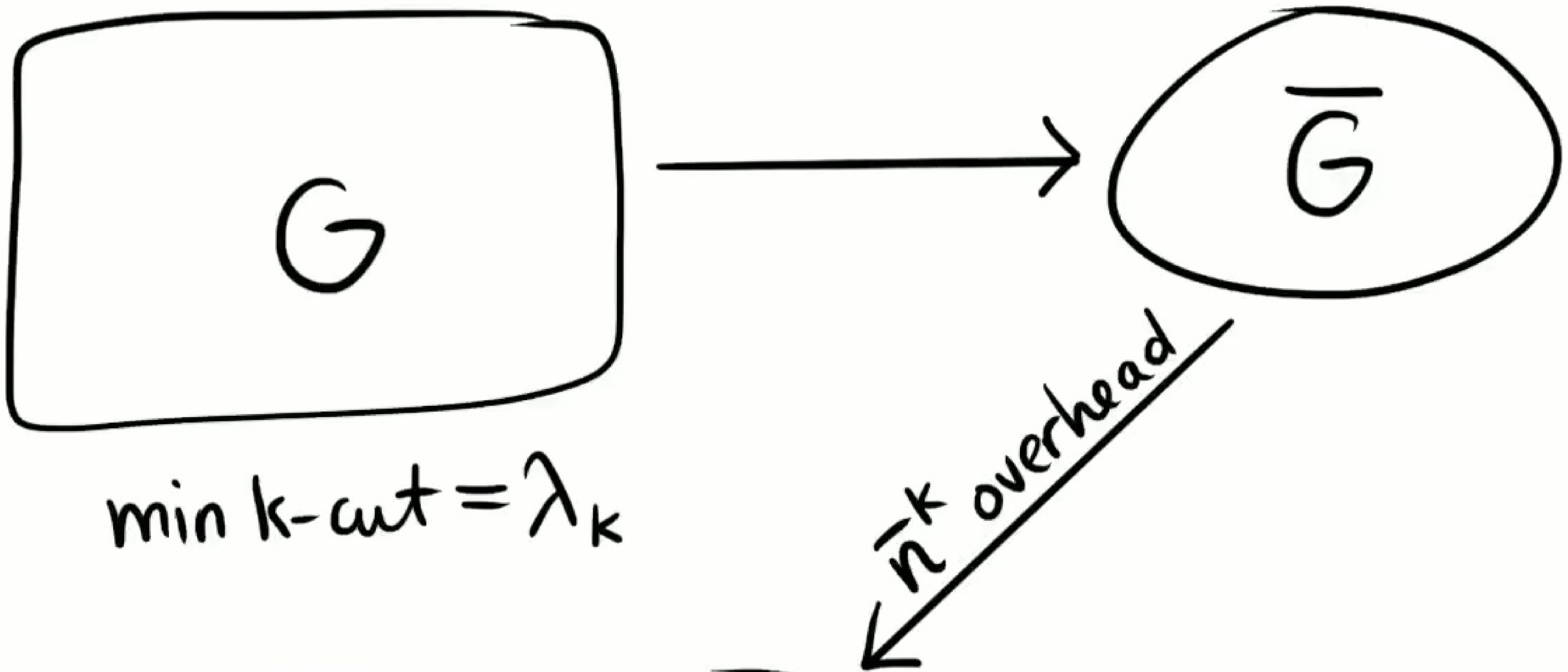
min k-cut $= \lambda_k$

Kawarabayashi-Thorup Sparsification

- $\bar{n} := n/\lambda_k$ vertices
- Preserves min k-cut

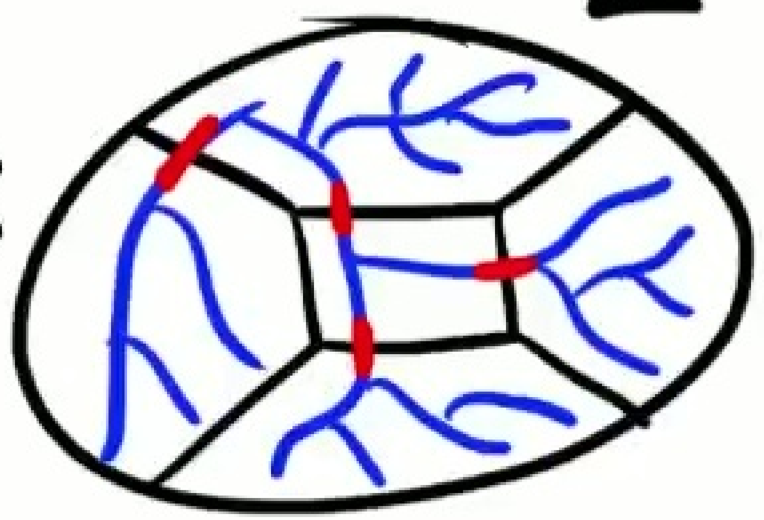
  (if it's nontrivial)

Algorithm Outline



Kawarabayashi-Thorup Sparsification

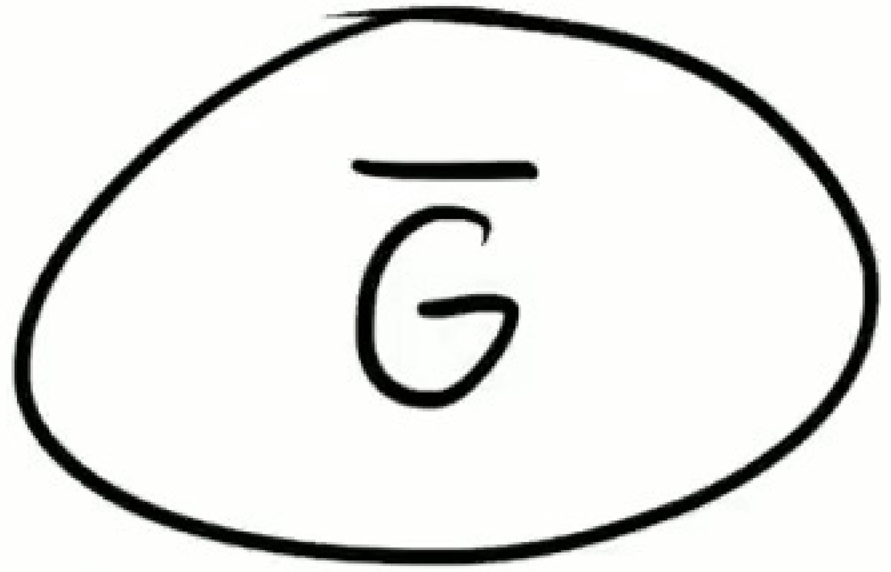
- $\bar{n} := n/\lambda_k$ vertices
- Preserves min k-cut
  (if it's nontrivial)



Tree packing:
$\bar{n}^k$ trees s.t.
exists a tree
intersecting
OPT k-1 times

Algorithm Outline

min k-cut $= \lambda_k$

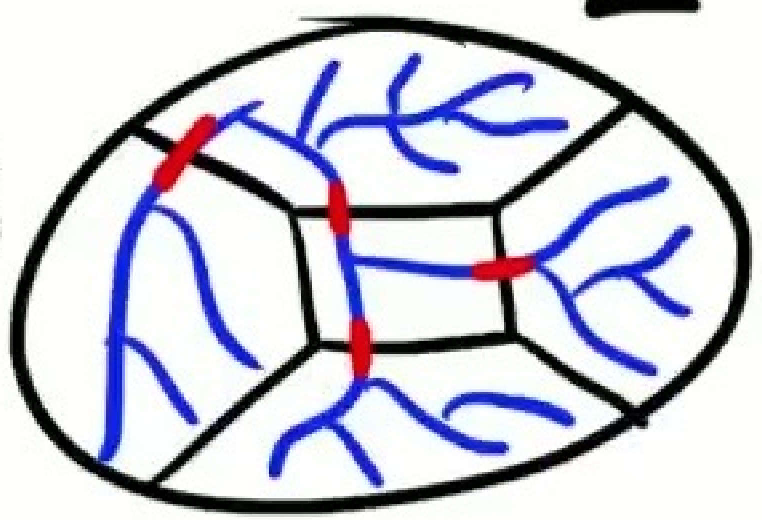


Kawarabayashi-Thorup Sparsification

- $\bar{n} := n/\lambda_k$ vertices
- Preserves min k-cut (if it's nontrivial)

$\bar{n}^k$ overhead

Tree packing: $\bar{n}^k$ trees s.t. exists a tree intersecting OPT k-1 times



Solve tree problem: cut best k-1 edges

- color-coding (this talk)
- heavy-light decomposition
- $\lambda_k^k \bar{n}^{o(k)}$ time

Algorithm Outline

$G \longrightarrow \bar{G}$

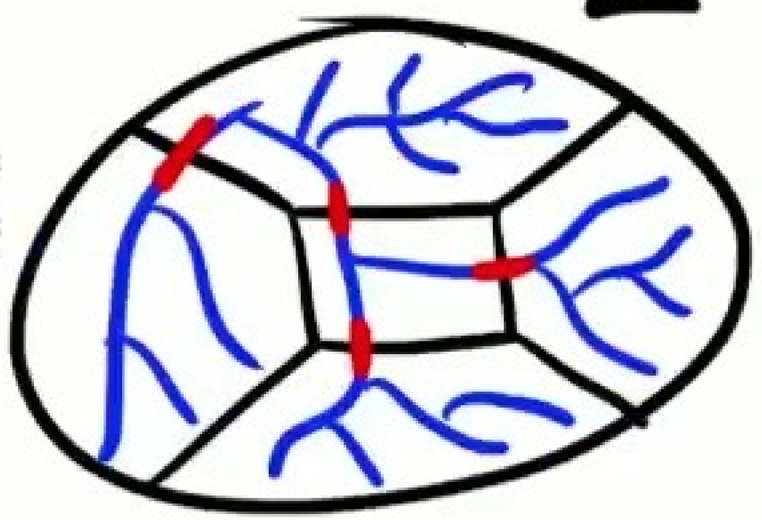
min k-cut $= \lambda_k$

Kawarabayashi-Thorup Sparsification

- $\bar{n} := n/\lambda_k$ vertices
- Preserves min k-cut (if it's nontrivial)

$\bar{n}^k$ overhead

Tree packing: $\bar{n}^k$ trees s.t. exists a tree intersecting OPT k-1 times



$$\boxed{\left(\frac{n}{\lambda_k}\right)^k \lambda_k^k n^{o(k)} = n^{(1+o(1))k} \text{ time}}$$

Solve tree problem: cut best k-1 edges

- color-coding (this talk)
- heavy-light decomposition
- $\lambda_k^k \bar{n}^{o(k)}$ time

Algorithm Outline

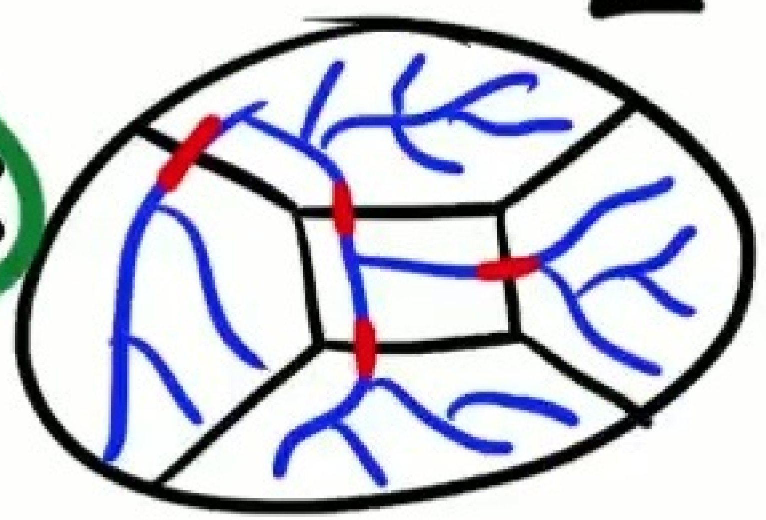
$G \longrightarrow \bar{G}$

min k-cut $= \lambda_k$

Kawarabayashi-Thorup
Sparsification
- $\bar{n} := n/\lambda_k$ vertices
- Preserves min k-cut
(if it's nontrivial)

$\bar{n}^k$ overhead

Tree packing:
$\bar{n}^k$ trees s.t.
exists a tree
intersecting
OPT k-1 times



$$\boxed{\left(\frac{n}{\lambda_k}\right)^k \lambda_k^k n^{o(k)} = n^{(1+o(1))k} \text{ time}}$$

Solve tree problem:
cut best k-1 edges
- color-coding (this talk)
- heavy-light decomposition
- $\lambda_k^k \bar{n}^{o(k)}$ time

Vertex Sparsification

- [Kawarabayashi-Thorup'15]
  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

Vertex Sparsification

- [Kawarabayashi-Thorup'15]

  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

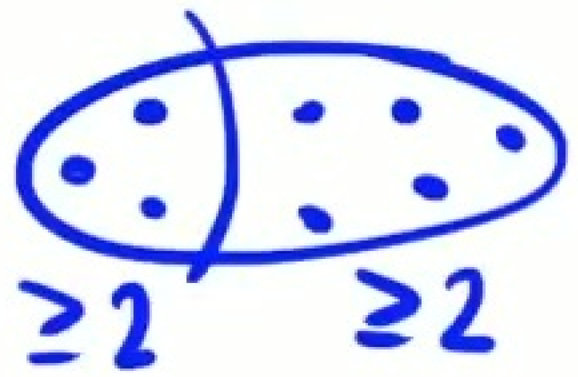
  - no nontrivial mincut has any edges contracted
  - number of vertices is $\tilde{O}(n/\lambda)$

Vertex Sparsification

- [Kawarabayashi-Thorup'15]

  For any simple graph G, can contract edges into multigraph $\bar{G}$ s.t.

  - no nontrivial mincut has any edges contracted

    at least two vertices on each side 

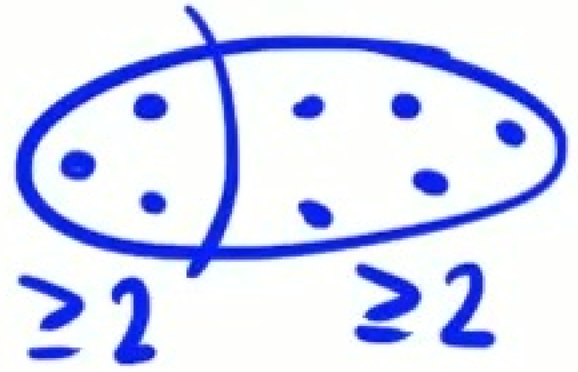
  - number of vertices is $\tilde{O}(n/\lambda)$

Vertex Sparsification

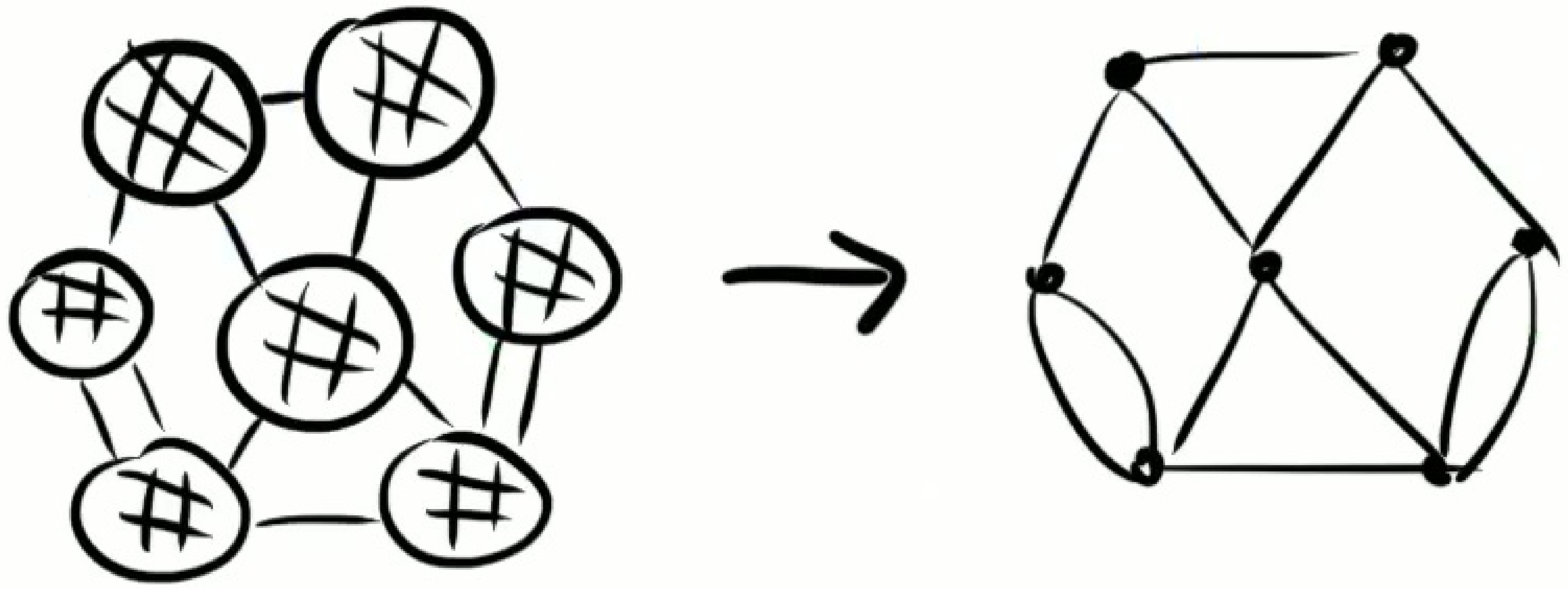
- [Kawarabayashi-Thorup'15]

  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

  - no nontrivial mincut has any edges contracted

    at least two vertices on each side 

  - number of vertices is $\tilde{O}(n/\lambda)$



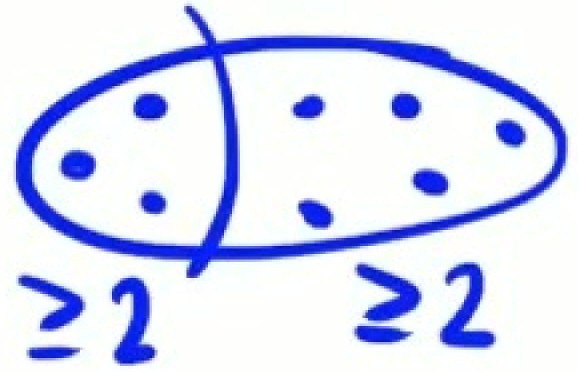
Vertex Sparsification

- [Kawarabayashi-Thorup'15]

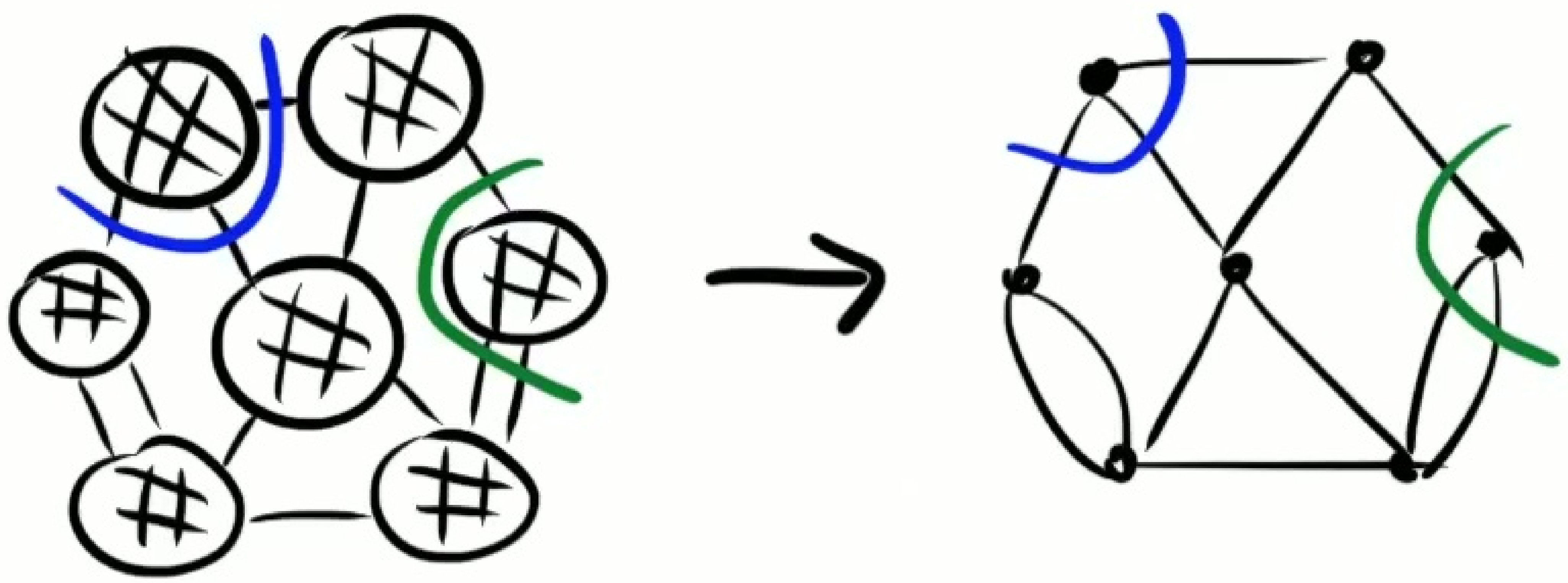
  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

  • no nontrivial mincut has any edges contracted

  at least two vertices on each side



  • number of vertices is $\tilde{O}(n/\lambda)$



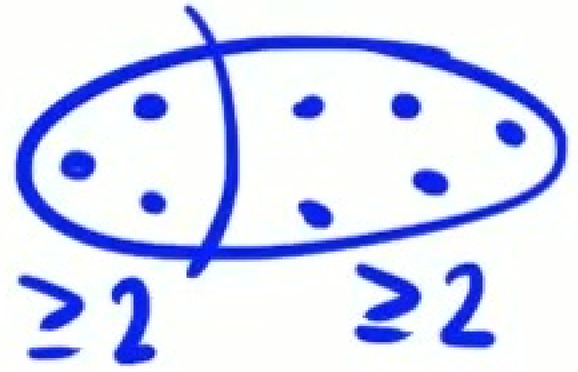
Vertex Sparsification

- [Kawarabayashi-Thorup'15]

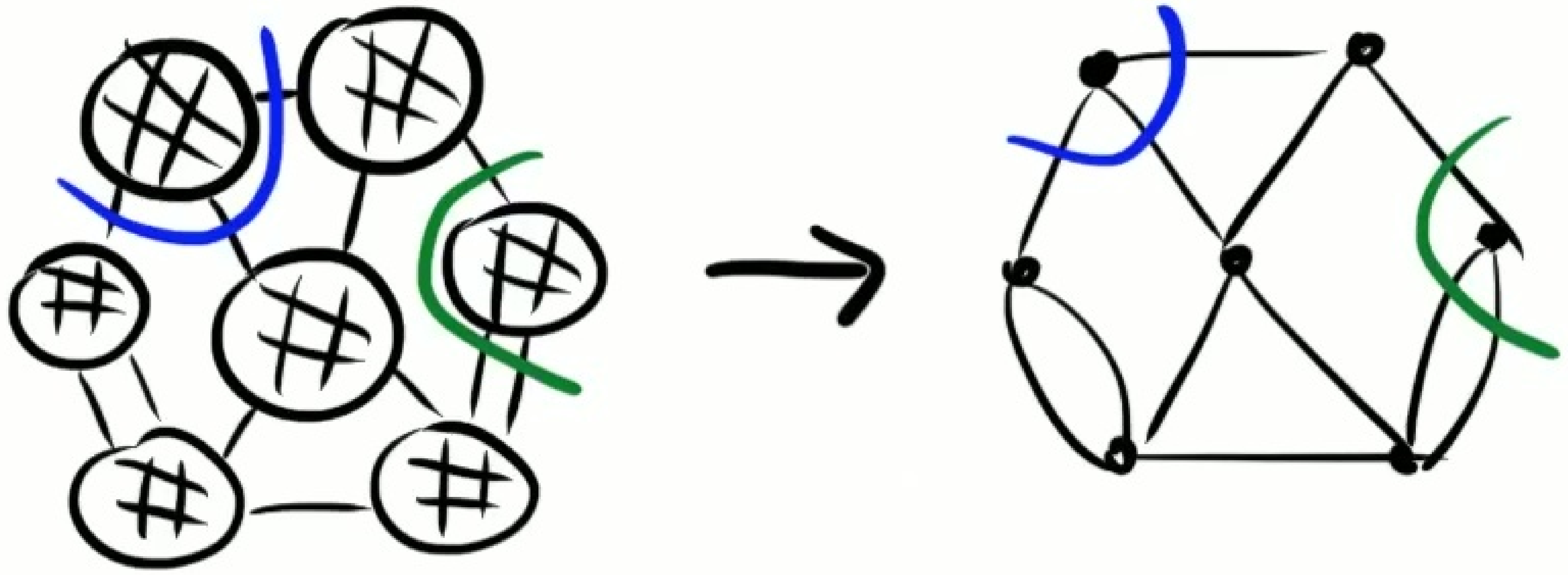
  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

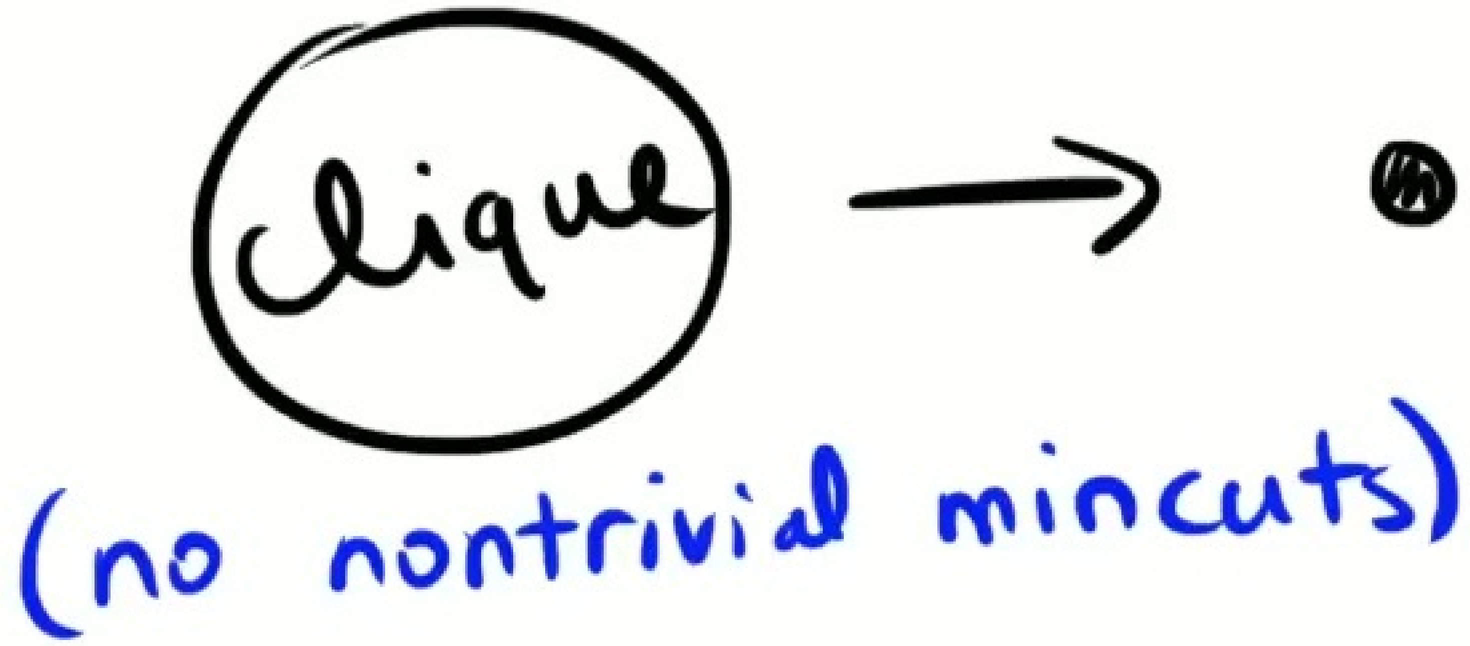
  - no nontrivial mincut has any edges contracted

    at least two vertices on each side



  - number of vertices is $\tilde{O}(n/\lambda)$



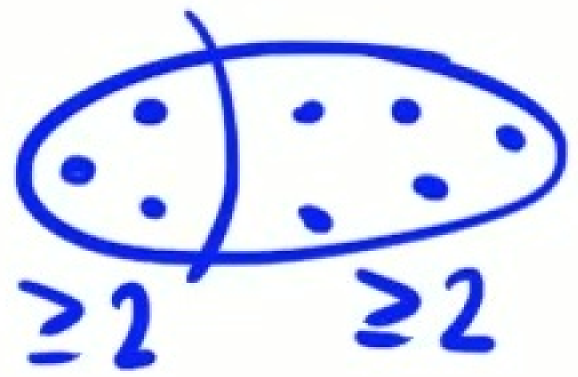


Vertex Sparsification

- [Kawarabayashi-Thorup'15]

  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

  - no nontrivial mincut has any edges contracted

    at least two vertices on each side 

  - number of vertices is $\tilde{O}(n/\lambda)$
  - Used for deterministic mincut: sparsify into $\overline{m} = \tilde{O}(m/\lambda)$ and run $\lambda\overline{m} = \tilde{O}(m)$ algorithm

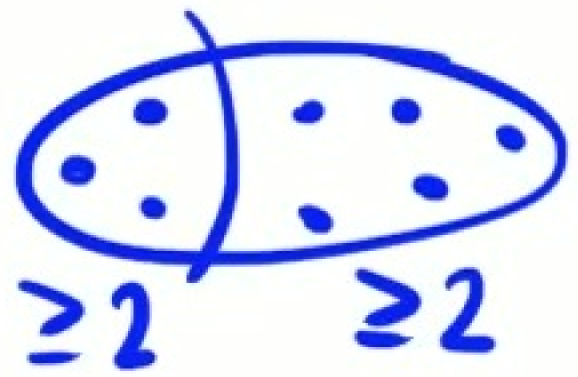
Vertex Sparsification

- [Kawarabayashi-Thorup'15]

  For any simple graph G, can contract edges into multigraph $\overline{G}$ s.t.

  - no nontrivial mincut has any edges contracted

    at least two vertices on each side



  - number of vertices is $\tilde{O}(n/\lambda)$
  - Used for deterministic mincut: sparsify into $\overline{m} = \tilde{O}(m/\lambda)$ and run $\lambda\overline{m} = \tilde{O}(m)$ algorithm

  Trivial mincuts easy

Vertex Sparsification

- [Kawarabayashi-Thorup'15] [This work]

For any simple graph G with (min k-cut) $=: \lambda_k \le f(k)\cdot\lambda$, can contract edges into multigraph $\bar{G}$ s.t.
(regularity)

- no nontrivial ~~mincut~~ min k-cut has any edges contracted

  at least two vertices on each side

- number of vertices is $\tilde{O}_k(n/\lambda_k)$
- Used for ~~deterministic~~ ~~mincut~~ min k-cut: sparsify into $\cancel{\bar{m}=\tilde{O}(m/\lambda)}$ $\bar{n}=\tilde{O}(n/\lambda_k)$

  and run $\cancel{\lambda\bar{m}=\tilde{O}(m)}$ $\boxed{\lambda_k^k\bar{n}^{k+o(k)}=n^{k+o(k)}}$ algorithm

Trivial mincuts easy

Vertex Sparsification

- [Kawarabayashi-Thorup'15] [This work]

  For any simple graph G with (min k-cut) =: $\lambda_k \le f(k)\cdot\lambda$, (regularity) can contract edges into multigraph $\bar{G}$ s.t.

  - no nontrivial ~~mincut~~ min k-cut has any edges contracted

    at least two vertices on each side

  - number of vertices is $\tilde{O}_k(n/\lambda_k)$
  - Used for ~~deterministic~~ ~~mincut~~ min k-cut: sparsify into ~~$\bar{m} = \tilde{O}(m/\lambda)$~~ $\bar{n} = \tilde{O}(n/\lambda_k)$ and run ~~$\lambda\bar{m} = \tilde{O}(m)$~~ $\boxed{\lambda_k^k \bar{n}^{k+o(k)} = n^{k+o(k)}}$ algorithm

  Trivial min k-cuts easy (guess one vertex and recurse)

Solving on Sparsified Graph

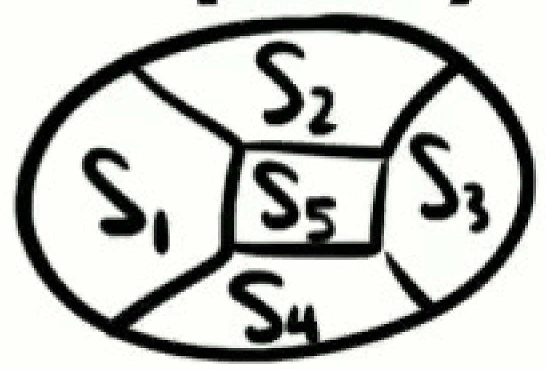
Goal: solve in time $\lambda^k n^{k+o(k)}$

Solving on Sparsified Graph

Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]:

Given weighted graph, exists collection $\mathcal{T}$ of poly(n) spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \leq 2k-2$

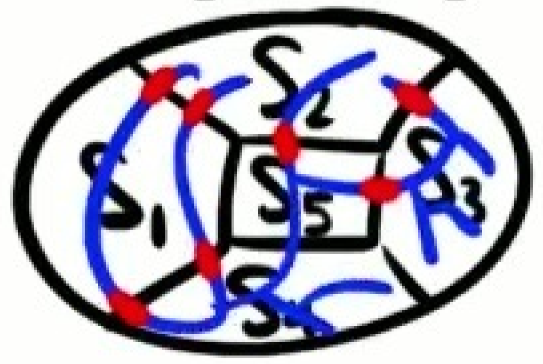


Solving on Sparsified Graph

Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]:

Given weighted graph, exists collection $\mathcal{T}$ of poly(n) spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$



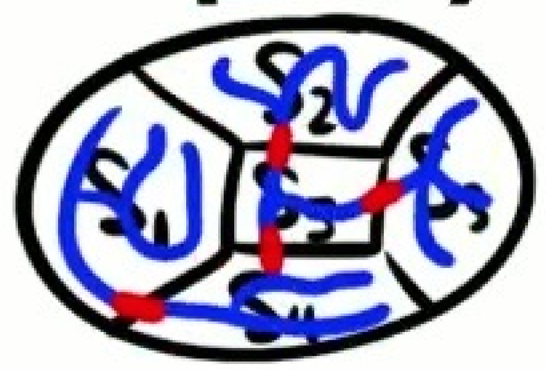
exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \leq 2k-2$

Solving on Sparsified Graph

Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \le \cancel{2k-2}$ $k-1$

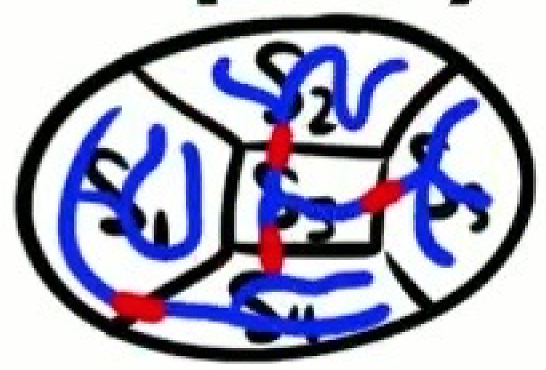


Solving on Sparsified Graph

Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \dots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \dots, S_k]| \leq \cancel{2k-2}\ k-1$



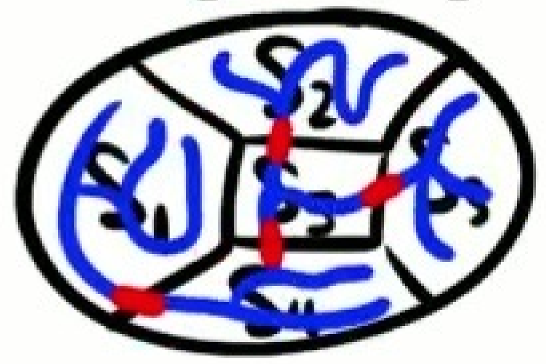
Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

Solving on Sparsified Graph

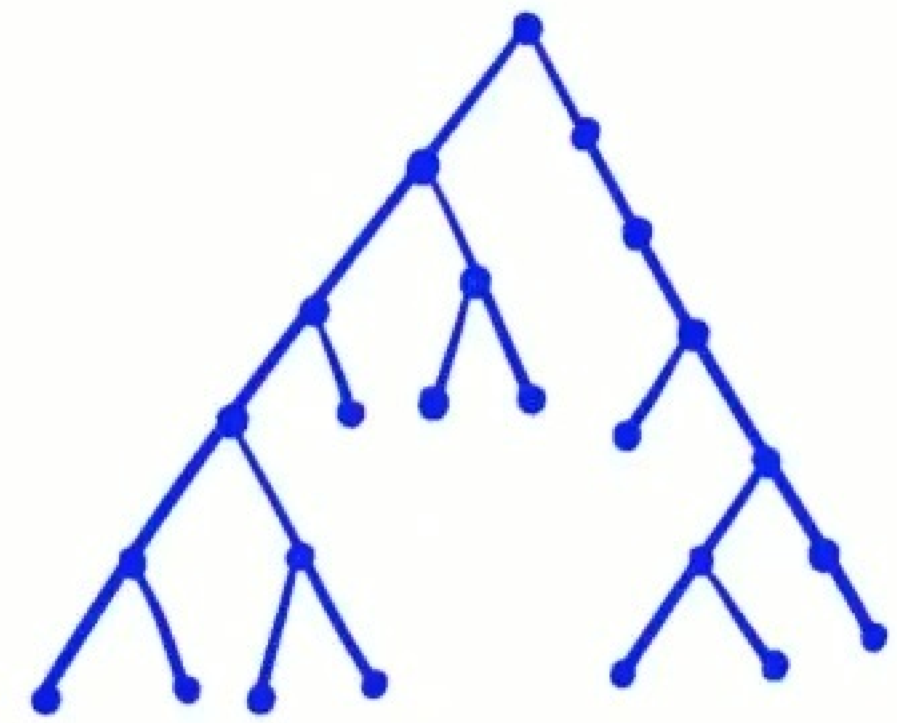
Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \le \cancel{2k-2}\ k-1$



Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

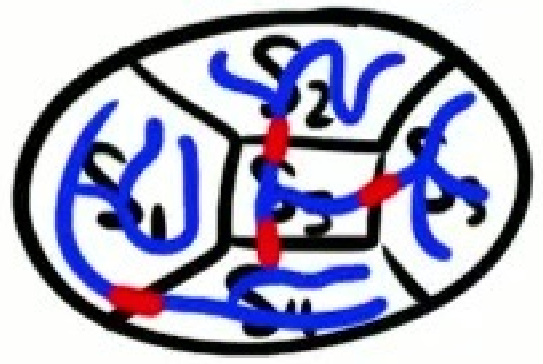


Solving on Sparsified Graph

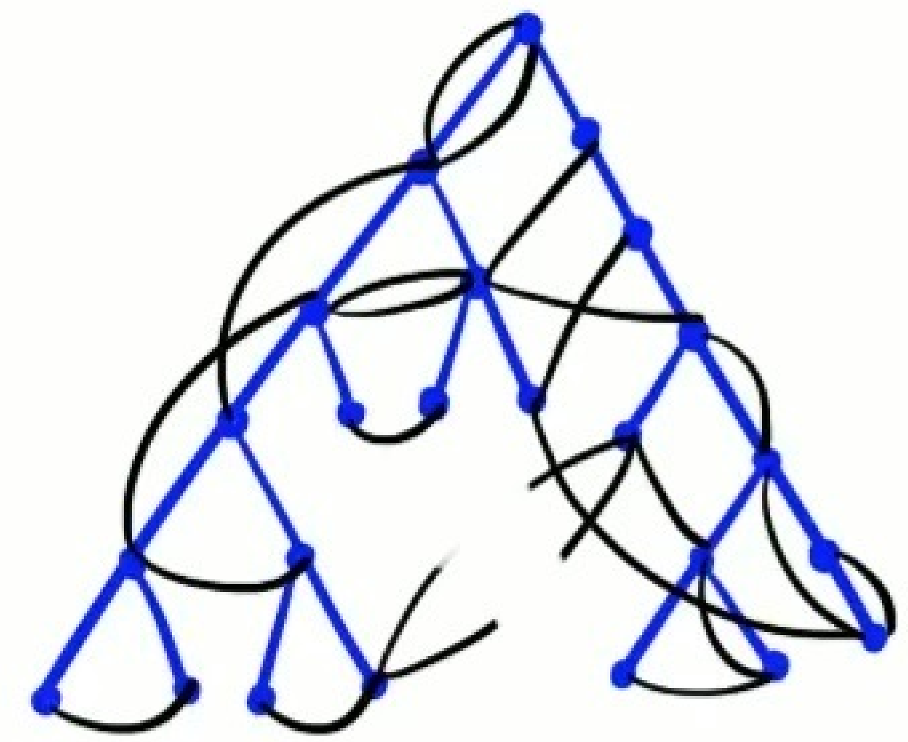
Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \le \cancel{2k-2}\ k-1$



Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

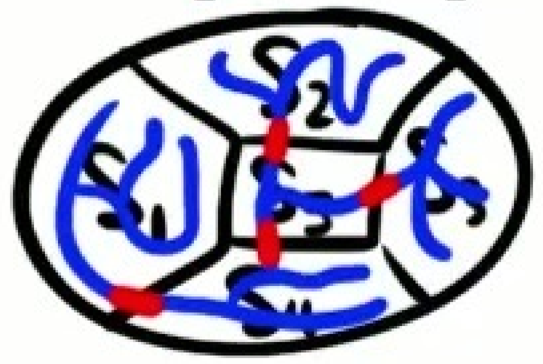


Solving on Sparsified Graph

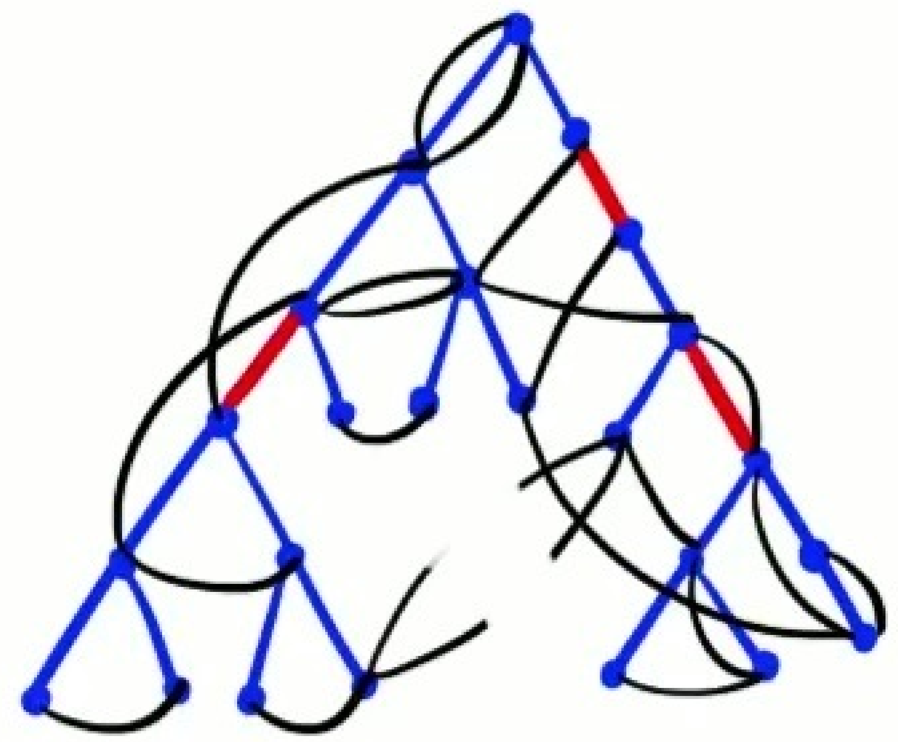
Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \leq \cancel{2k-2}\ k-1$



Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

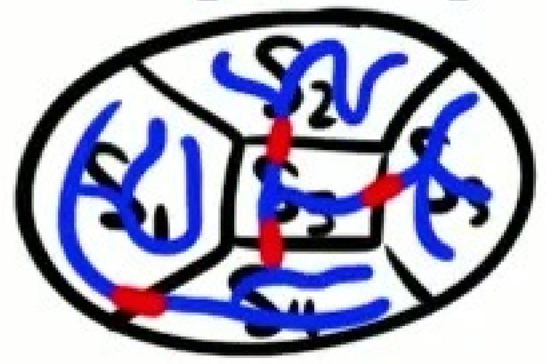


Solving on Sparsified Graph

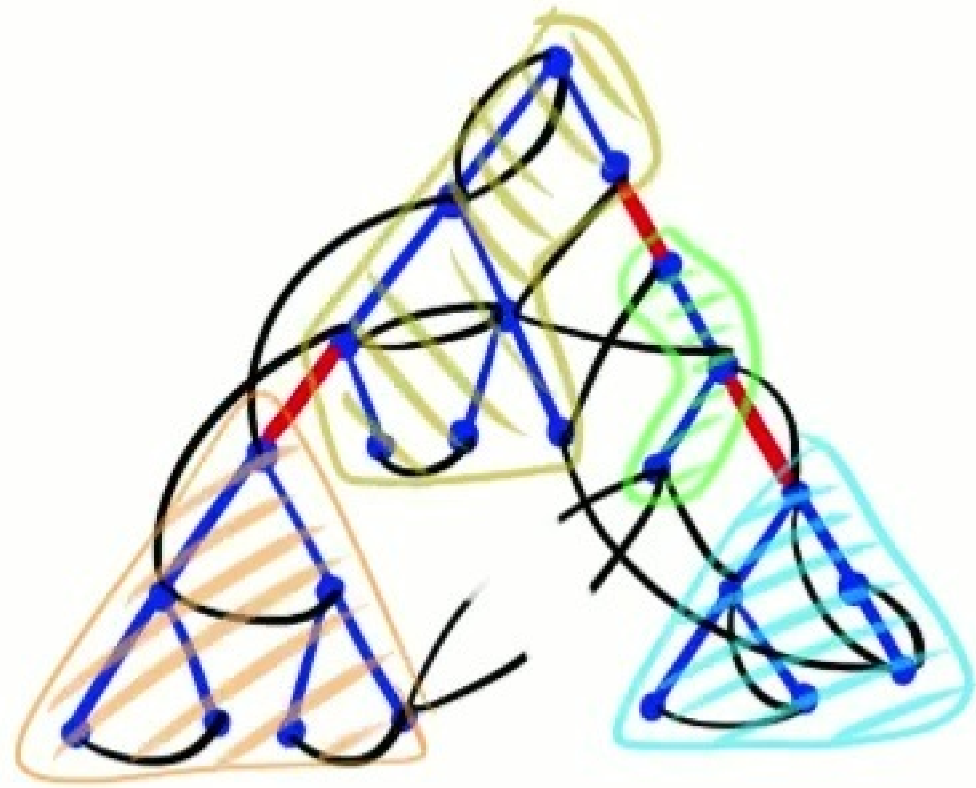
Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \leq \cancel{2k-2}\ k-1$



Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

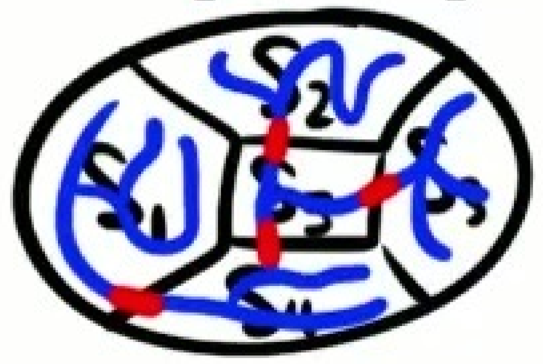


Solving on Sparsified Graph

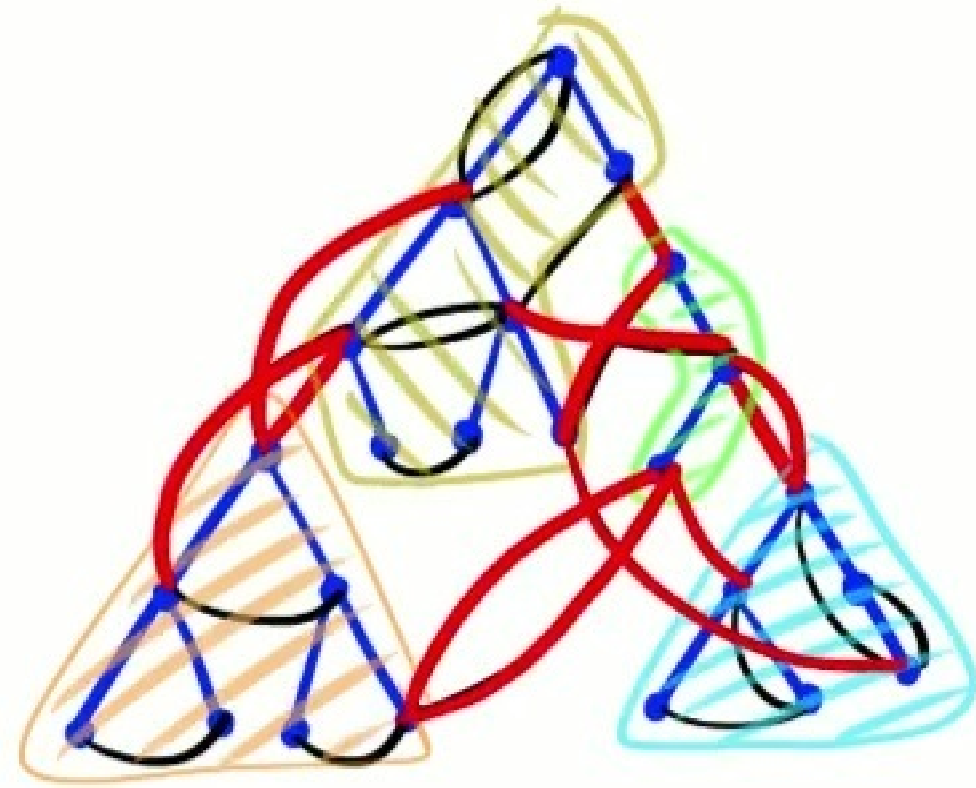
Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \dots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \dots, S_k]| \le \cancel{2k-2}$ $k-1$



Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

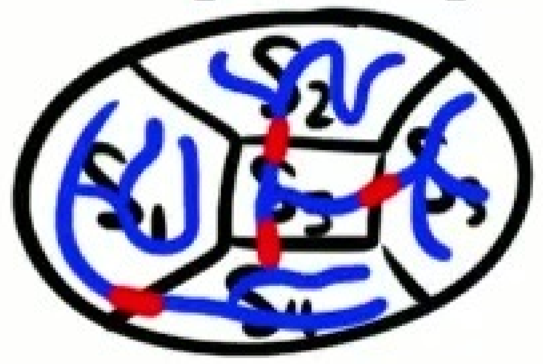


Solving on Sparsified Graph

Goal: solve in time $\lambda^k n^{k+o(k)}$

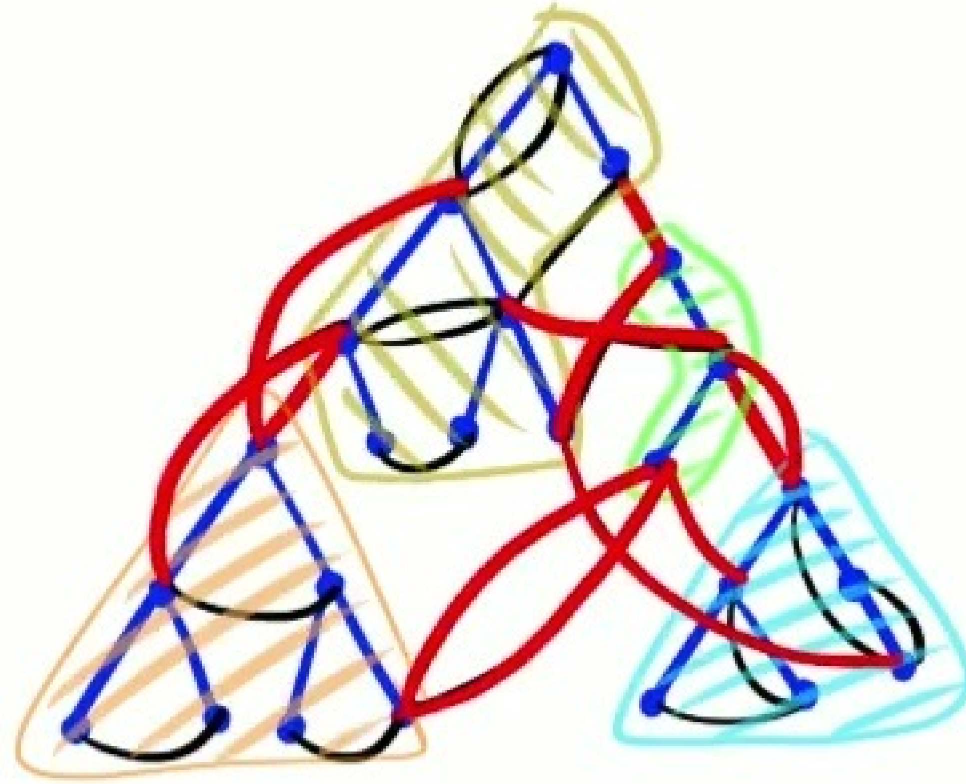
Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \ldots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \ldots, S_k]| \le \cancel{2k-2}$ $k-1$



Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$

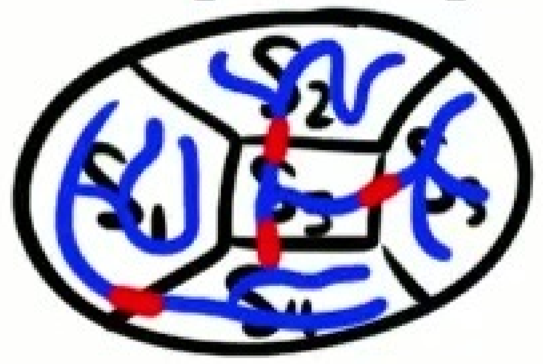


Solving on Sparsified Graph

Goal: solve in time $\lambda^k n^{k+o(k)}$

Tree Packing [Thorup]: [GLL'18]:

Given weighted graph, exists collection $\mathcal{T}$ of ~~poly(n)~~ n^k spanning trees of G s.t. for any min k-cut $S_1, S_2, \dots, S_k \subseteq V$ exists tree $T \in \mathcal{T}$ s.t. $|E_T[S_1, S_2, \dots, S_k]| \le \cancel{2k-2}\ k-1$

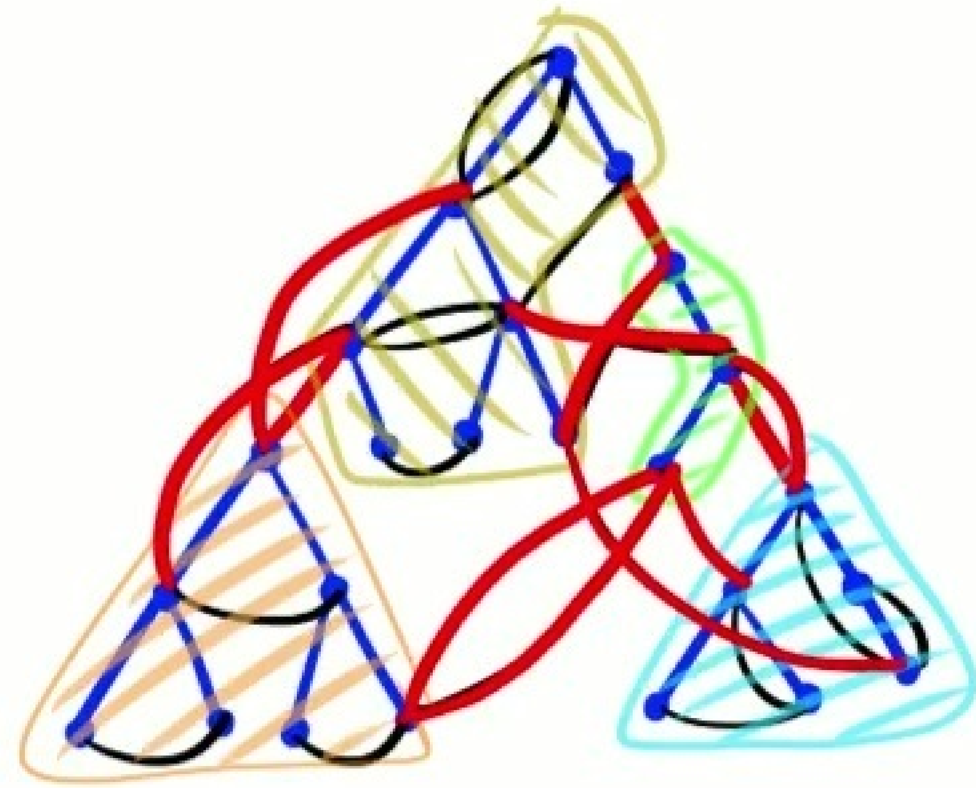


Suffices to solve problem: given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$

λ_k^k is FPT in λ_k and k,

but particular dependency matters!



Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

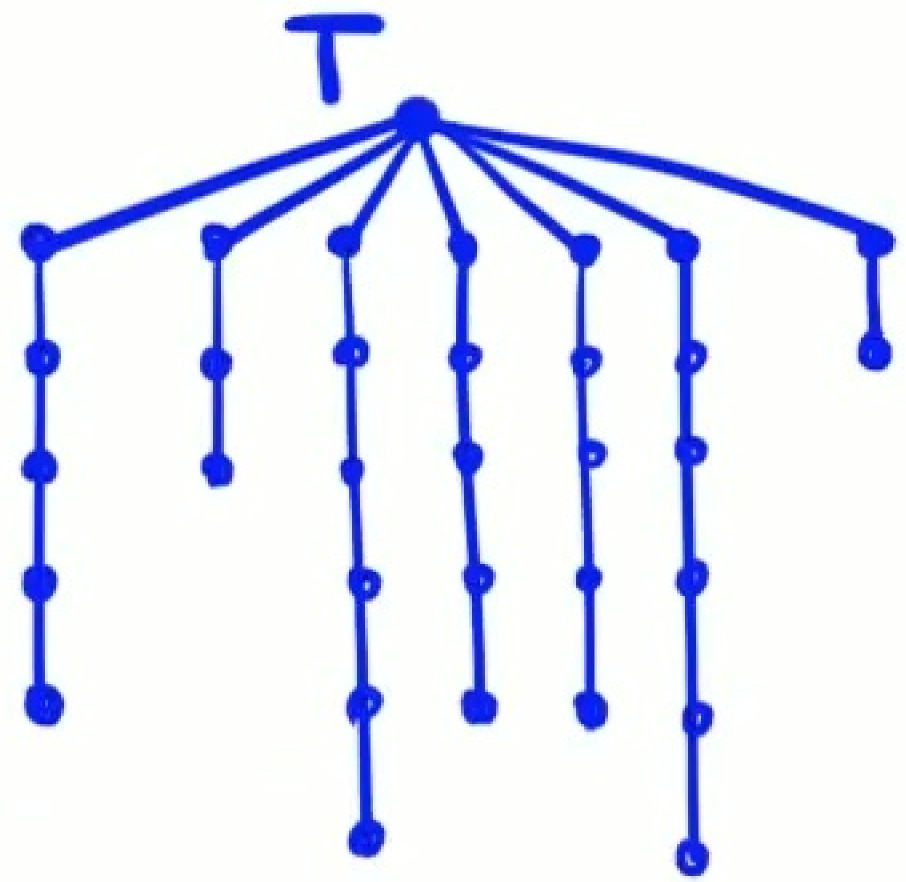
Time $\tilde{O}(\lambda_k^k n^{o(k)})$

Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$

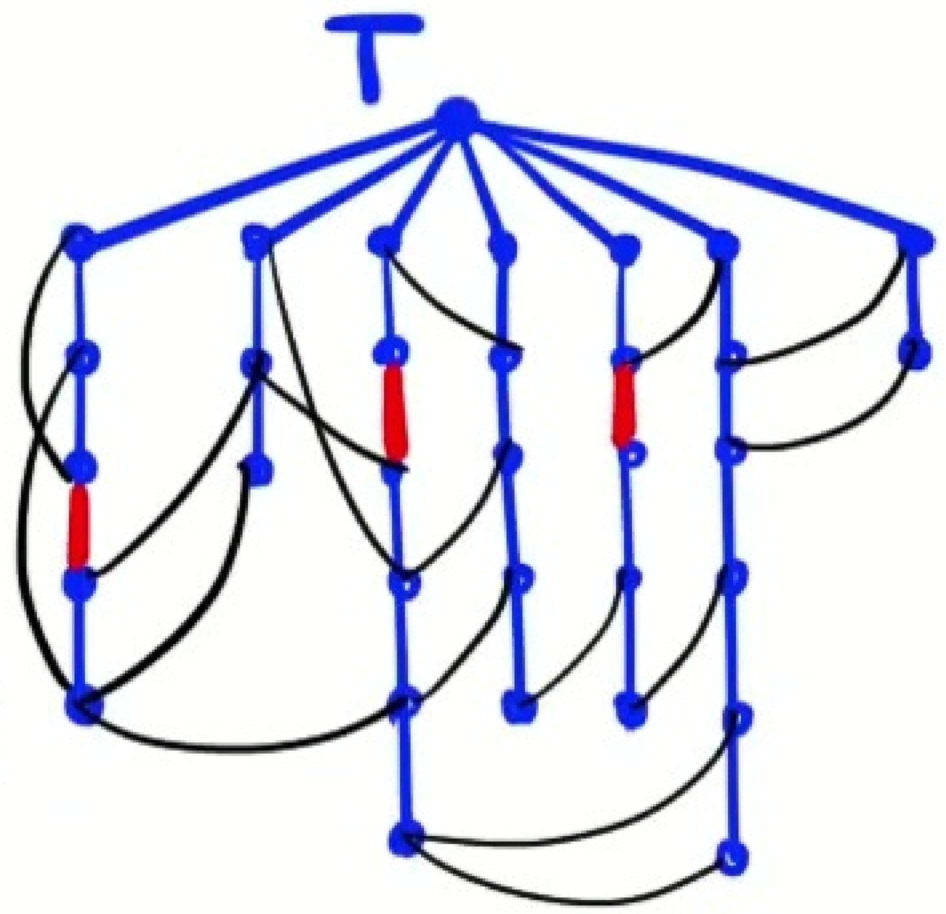
This talk: when T is a "spider"



Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$



This talk: when T is a “spider”

- OPT cuts at most one edge per branch

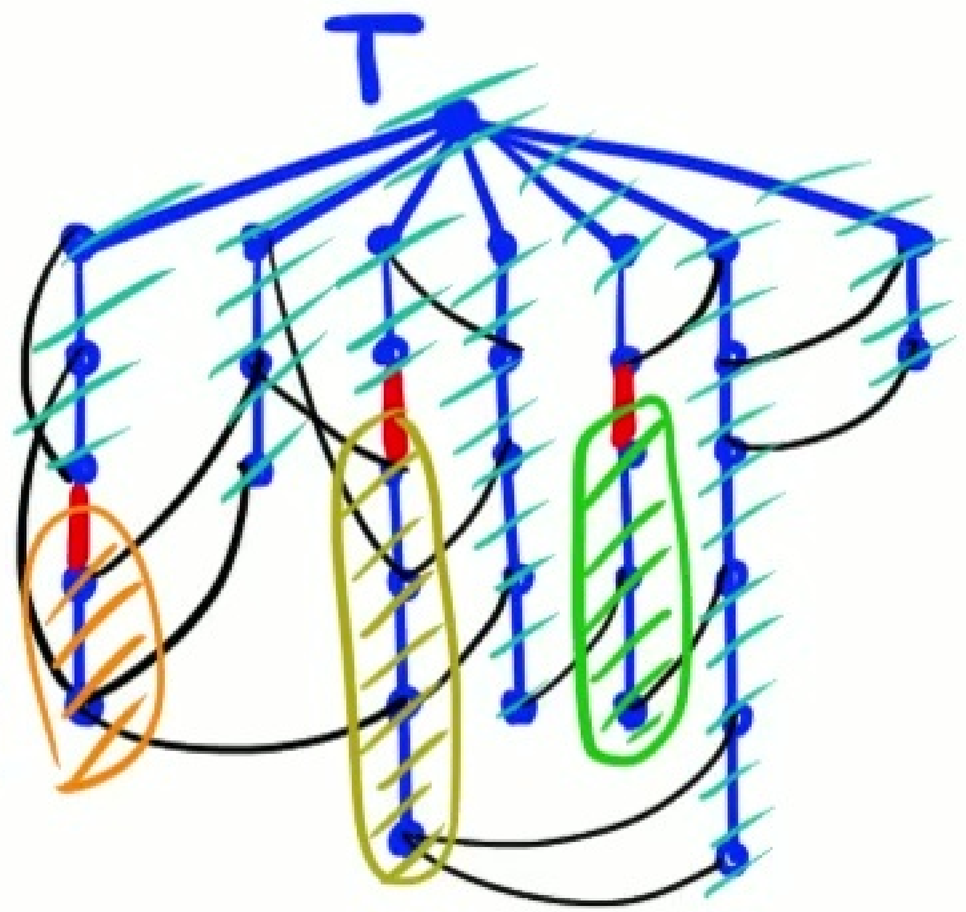
Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$

This talk: when T is a "spider"

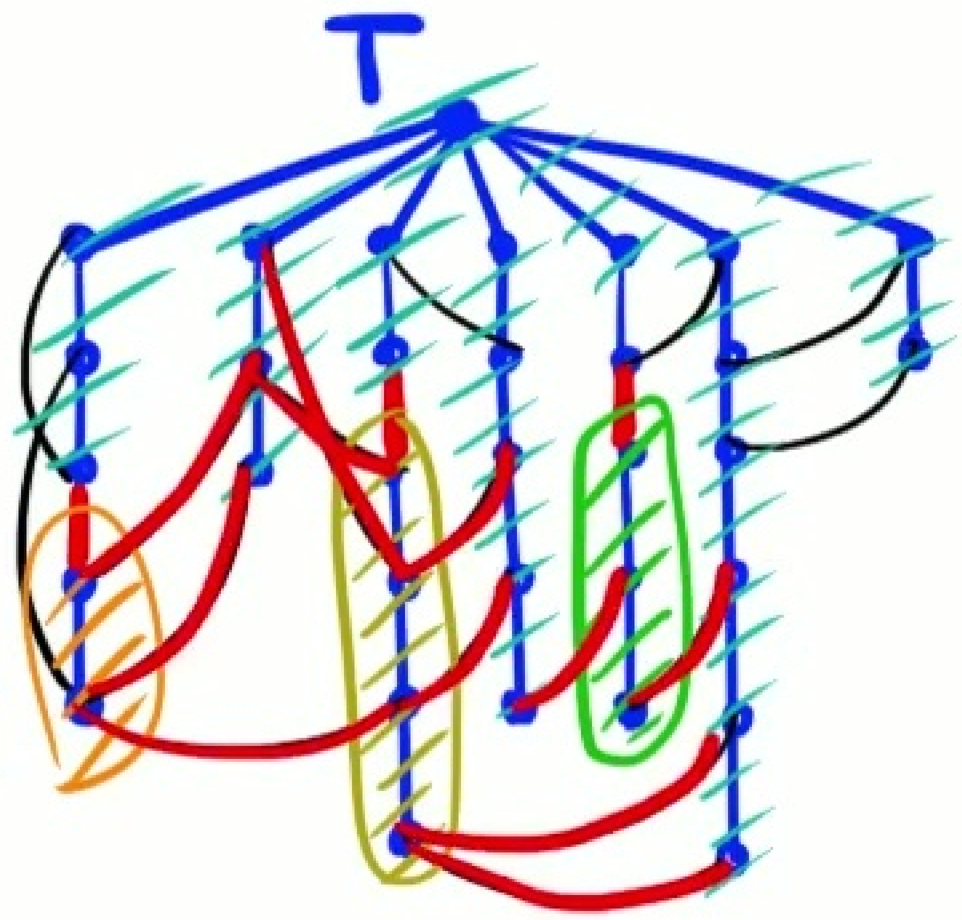
- OPT cuts at most one edge per branch



Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$



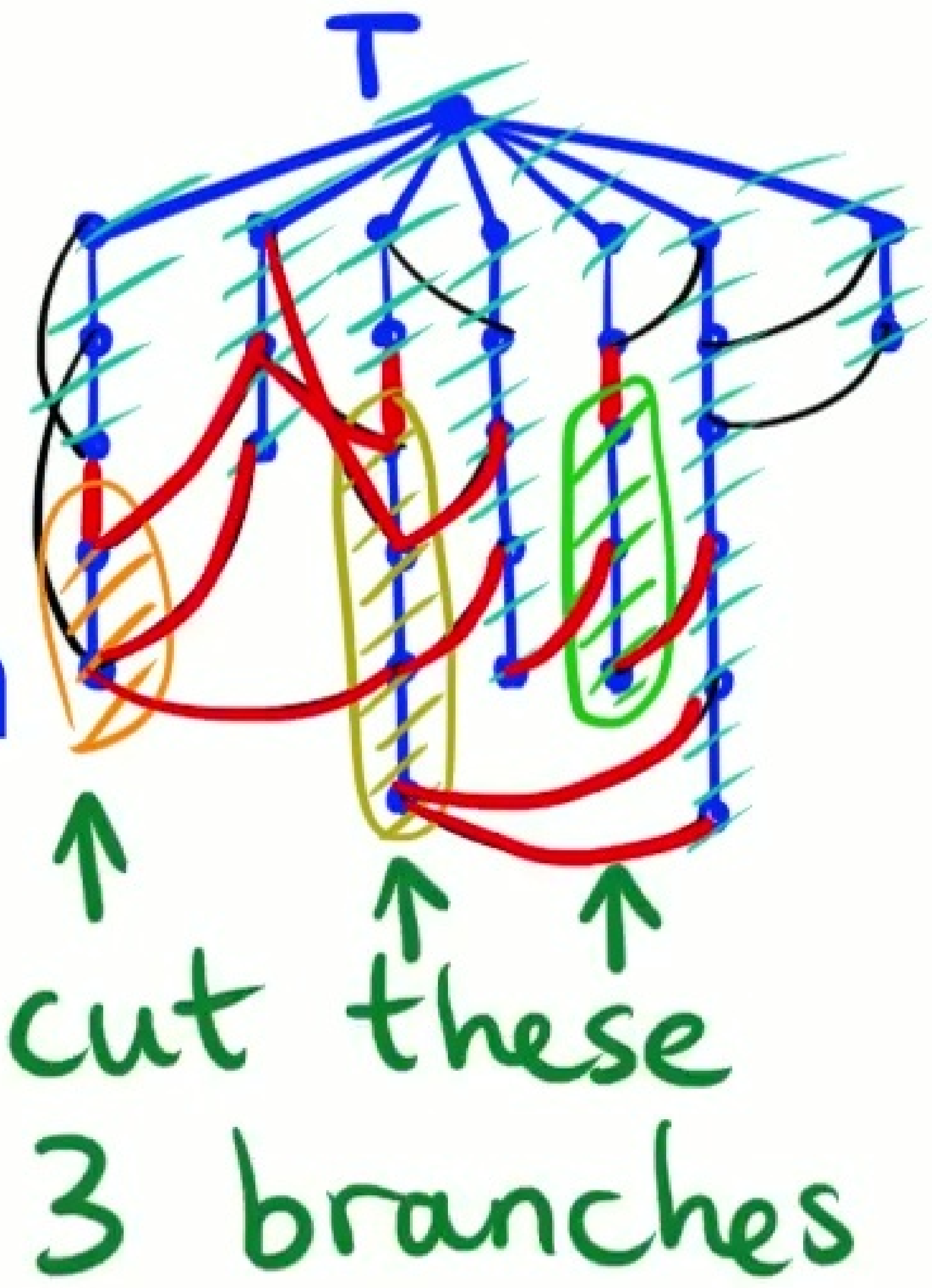
This talk: when T is a “spider”

- OPT cuts at most one edge per branch

Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$



This talk: when T is a "spider"

- OPT cuts at most one edge per branch

① Find which branches to cut: $\binom{\#\text{branches}}{k-1}$ choices naively

Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

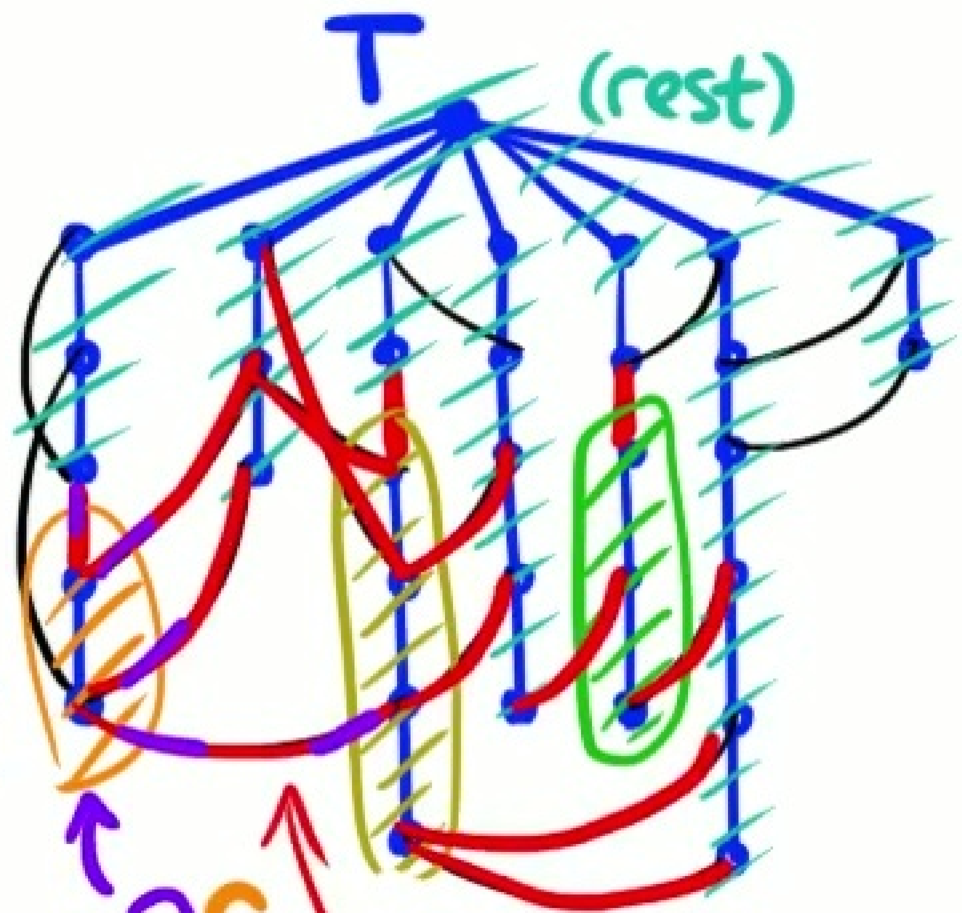
Time $\tilde{O}(\lambda_k^k n^{o(k)})$

This talk: when T is a "spider"

- OPT cuts at most one edge per branch

① Find which branches to cut: $\binom{\#\text{branches}}{k-1}$ choices naively

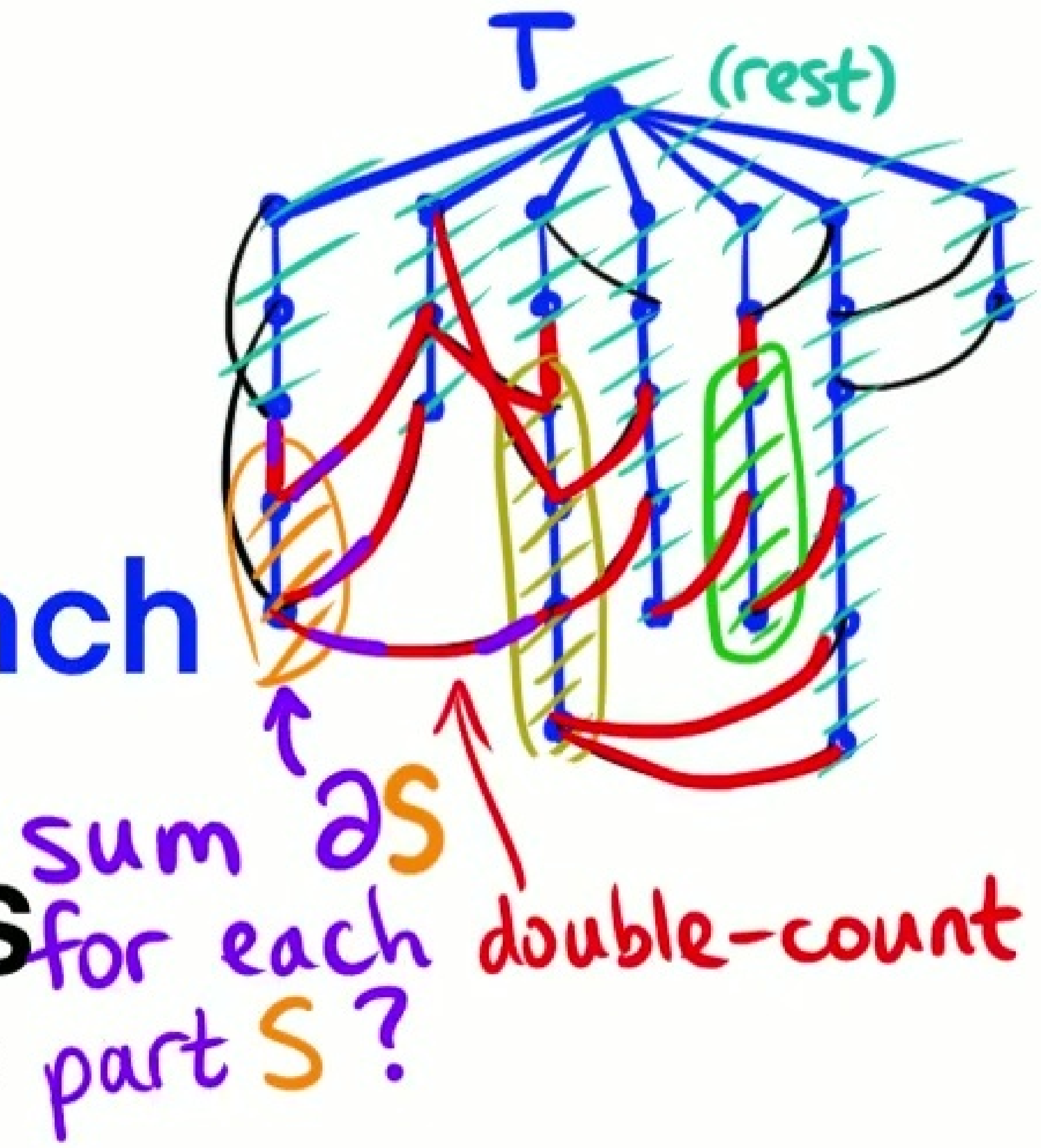
② Deal with double-counting



Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$



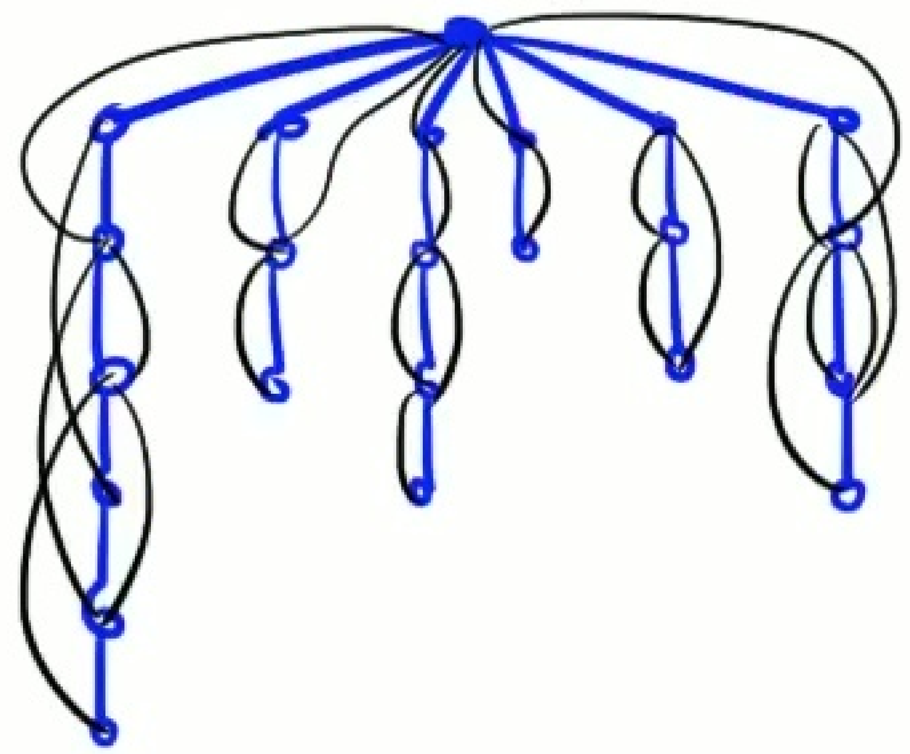
This talk: when T is a "spider"

- OPT cuts at most one edge per branch

① Find which branches to cut: $\binom{\#\text{branches}}{k-1}$ choices naively

② Deal with double-counting

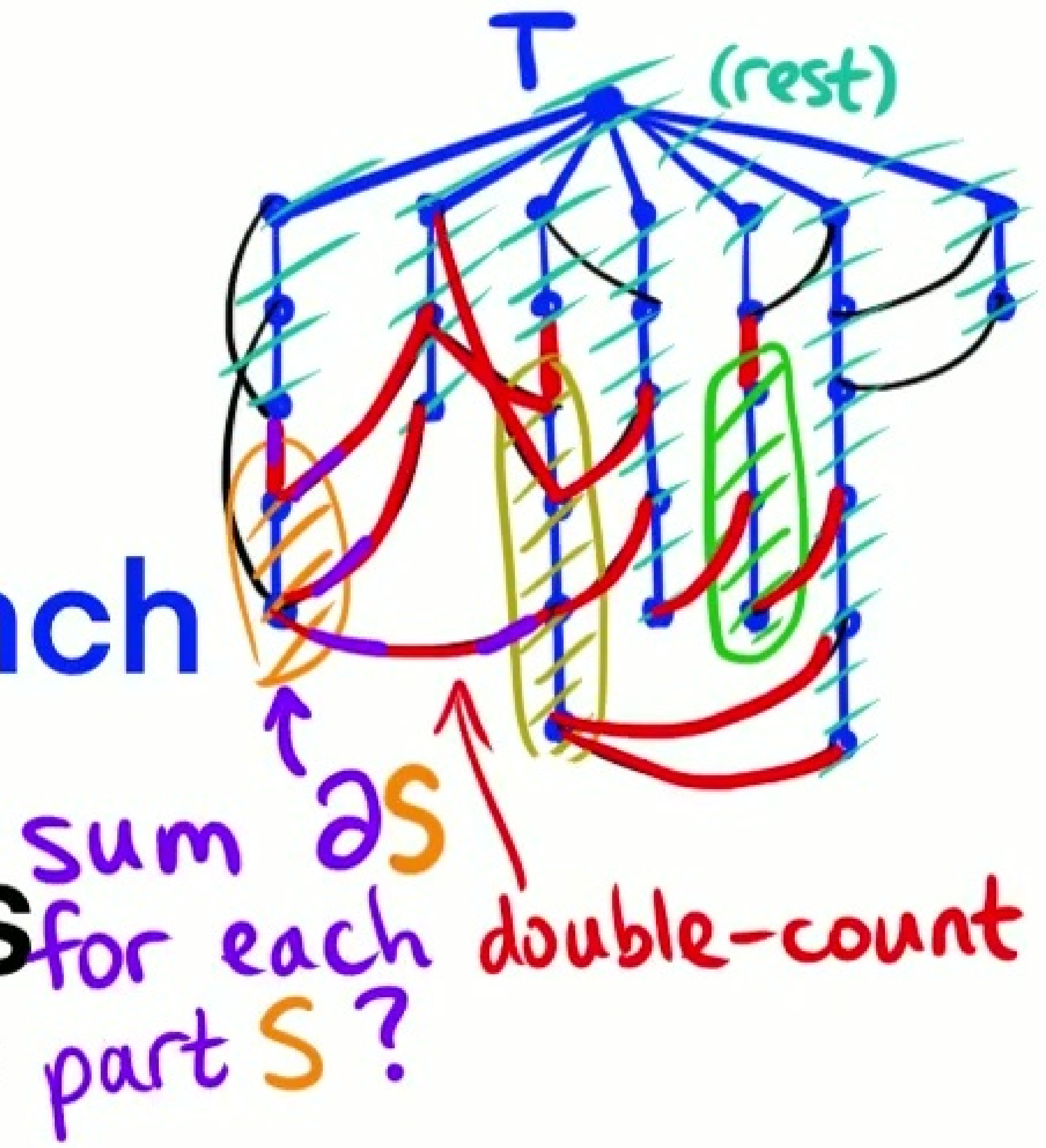
② Extreme case: no edges between different branches



Restricted Problem to Solve

given (tight) tree T, delete best k-1 edges to form smallest k-cut

Time $\tilde{O}(\lambda_k^k n^{o(k)})$

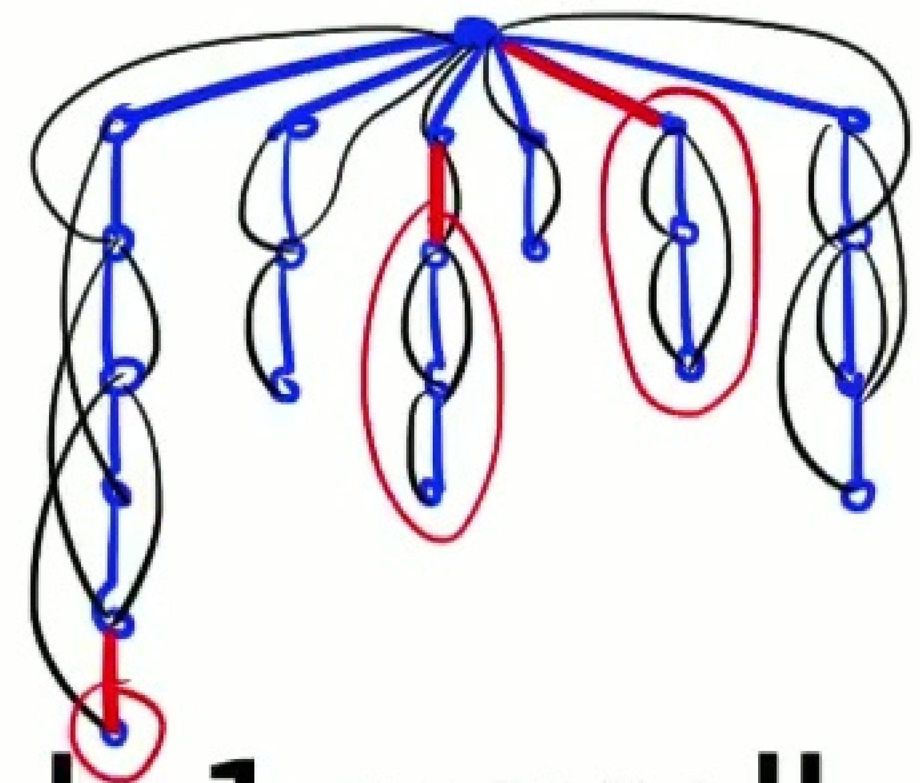


This talk: when T is a "spider"

- **OPT cuts at most one edge per branch**

①Find which branches to cut: $\binom{\#\text{branches}}{k-1}$ choices naively

②Deal with double-counting



② Extreme case: no edges between different branches

For each branch, take best cut; take best k-1 overall

Identifying the Branches

Other extreme: suppose many edges between OPT's branches

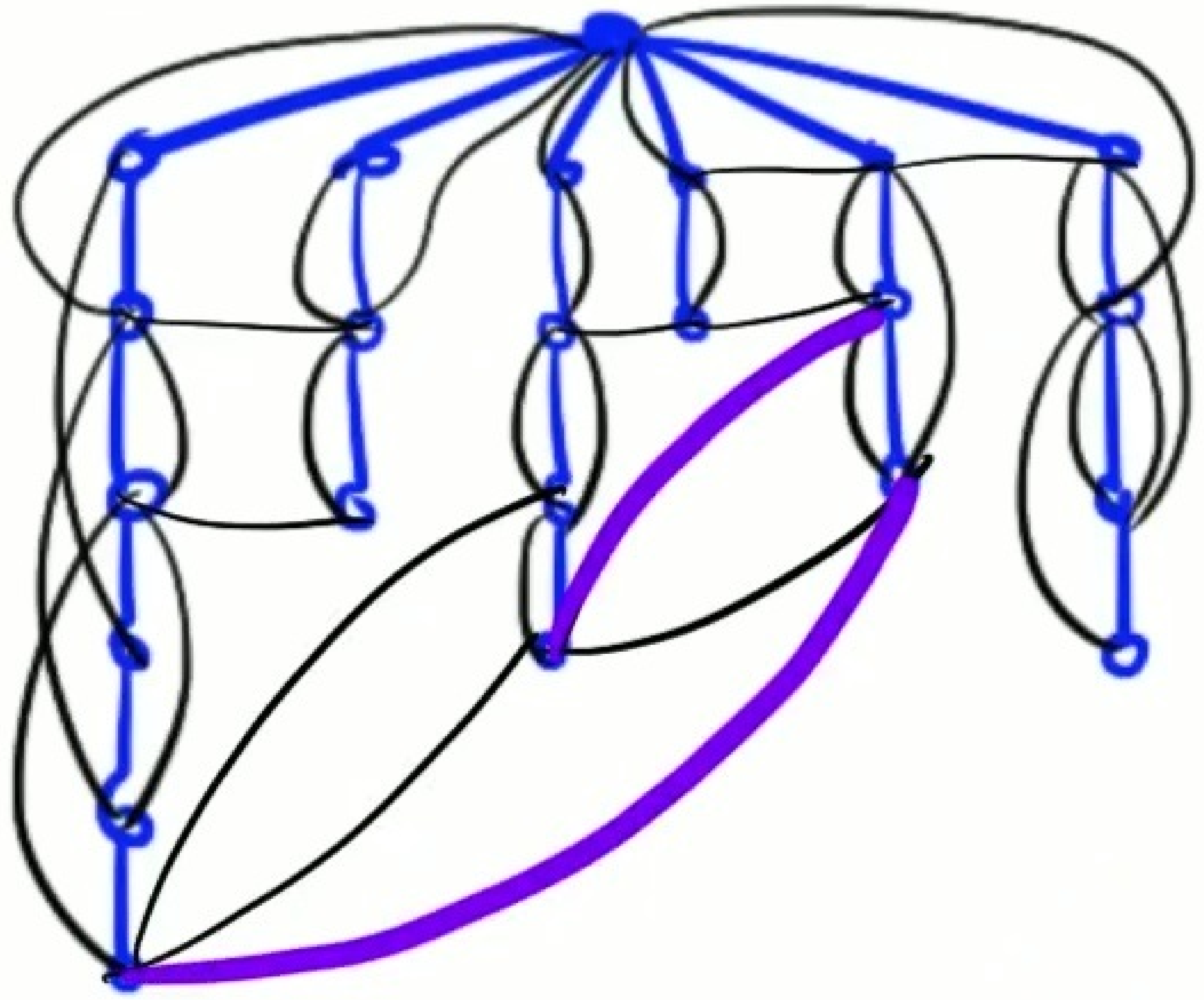
Identifying the Branches

Other extreme: suppose many edges between OPT's branches

Idea: color-code a spanning tree connecting these branches

Identifying the Branches

Other extreme: suppose many edges between OPT's branches

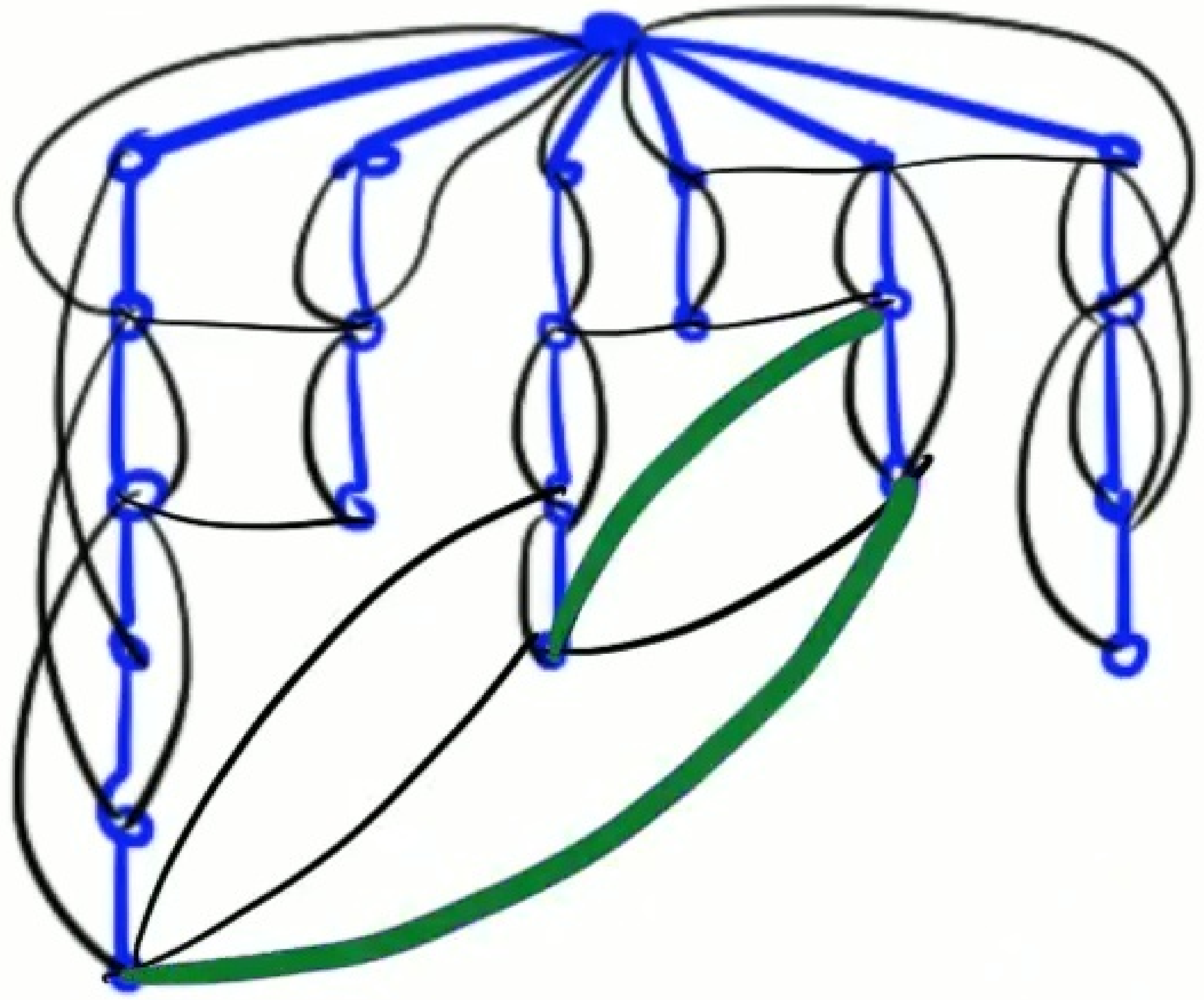


Idea: color-code a spanning tree connecting these branches

Identifying the Branches

Other extreme: suppose many edges between OPT's branches

Idea: color-code a spanning tree connecting these branches

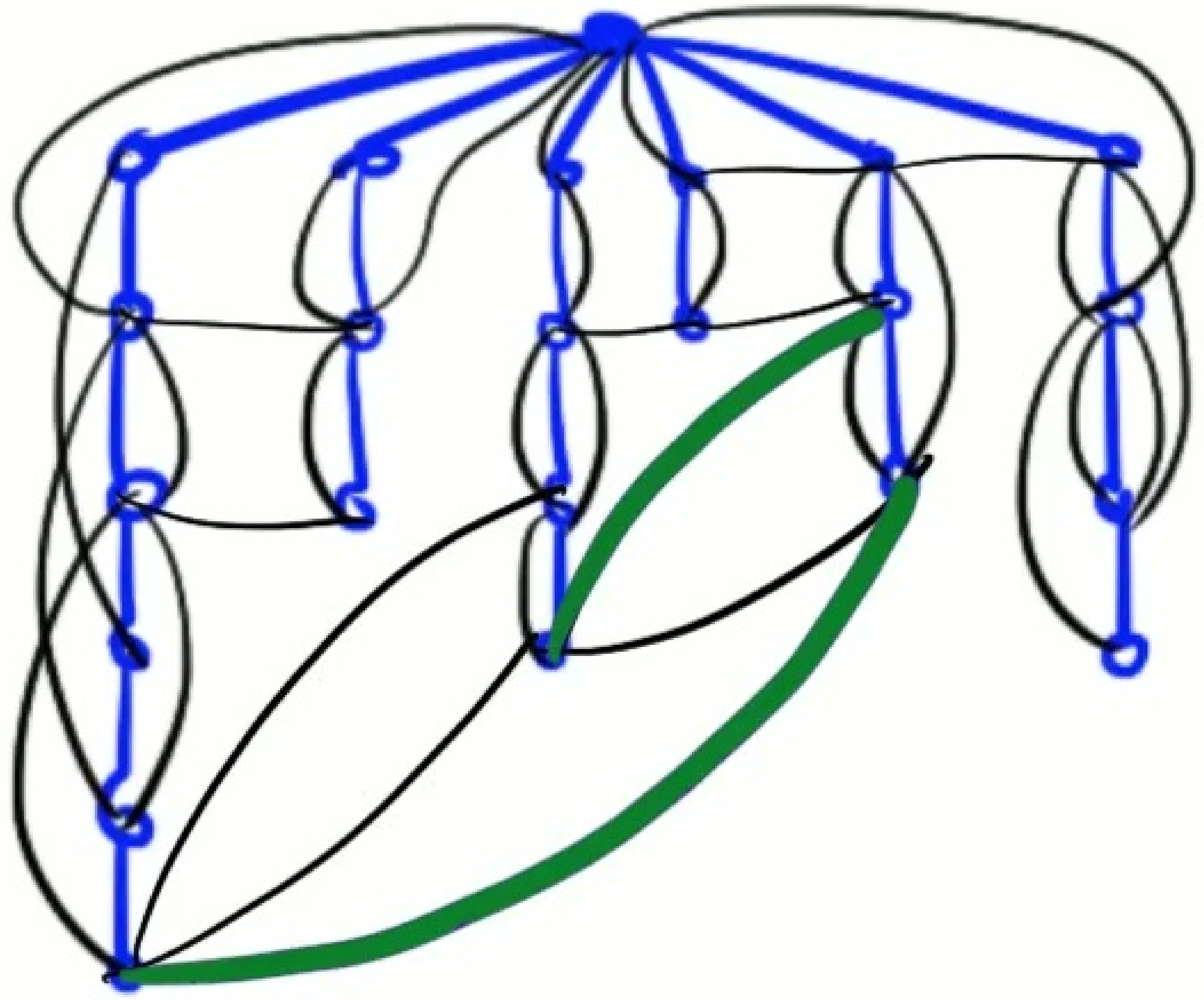


Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green

Identifying the Branches

Other extreme: suppose many edges between OPT's branches



Idea: color-code a spanning tree connecting these branches

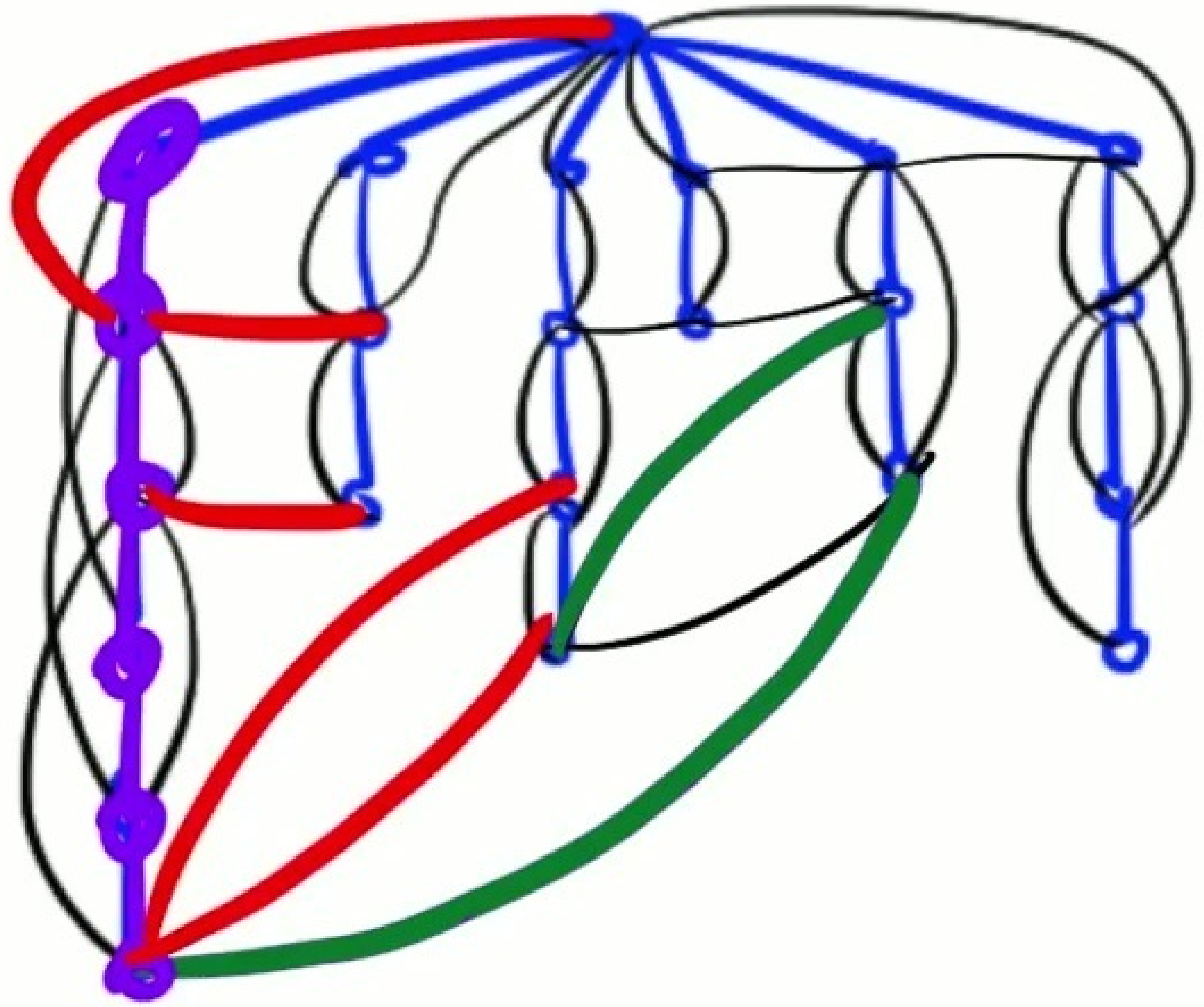
Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green

For each of OPT's branches, remaining boundary edges red

Identifying the Branches

Other extreme: suppose many edges between OPT's branches



Idea: color-code a spanning tree connecting these branches

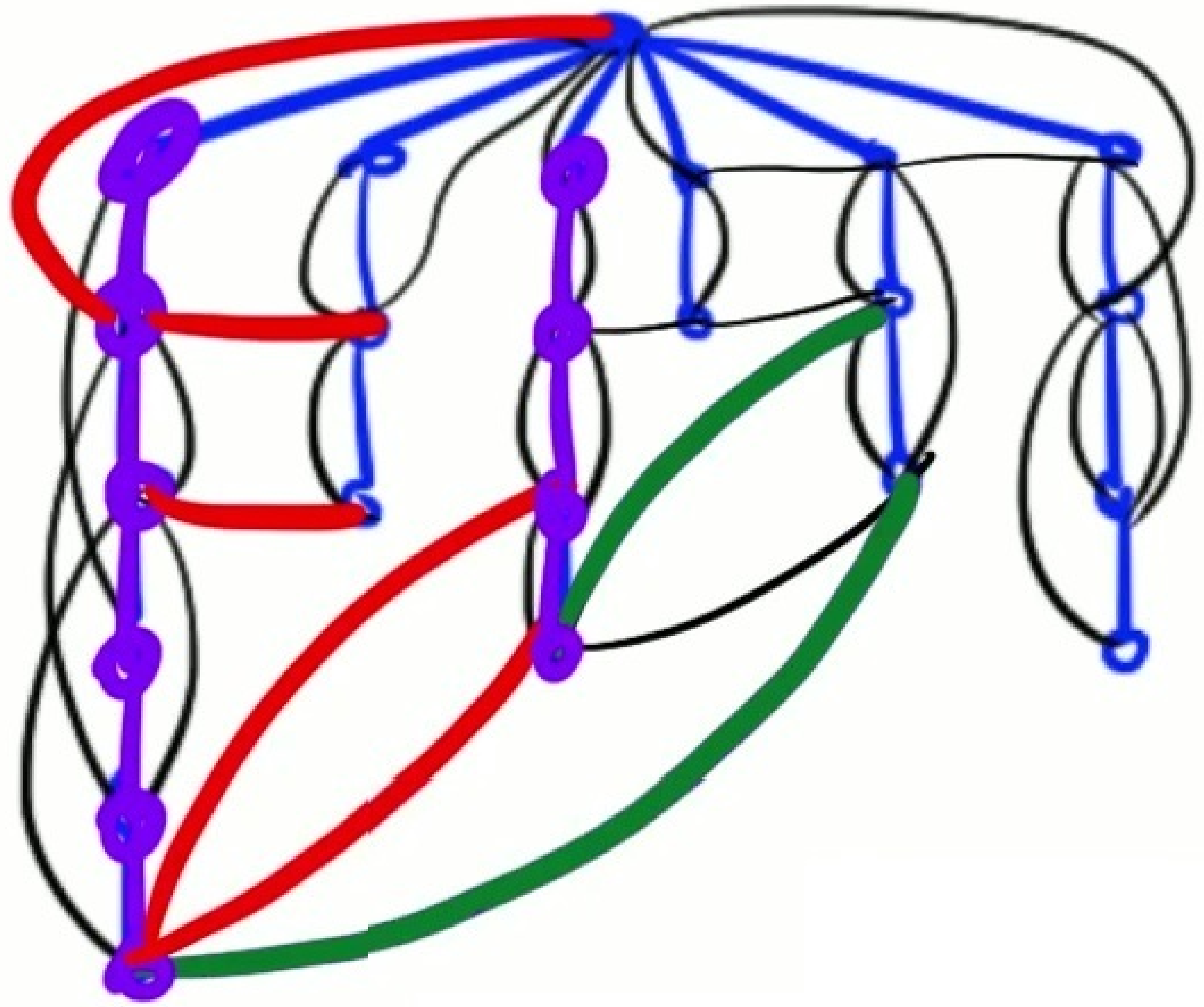
Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green

For each of OPT's branches, remaining boundary edges red

Identifying the Branches

Other extreme: suppose many edges between OPT's branches



Idea: color-code a spanning tree connecting these branches

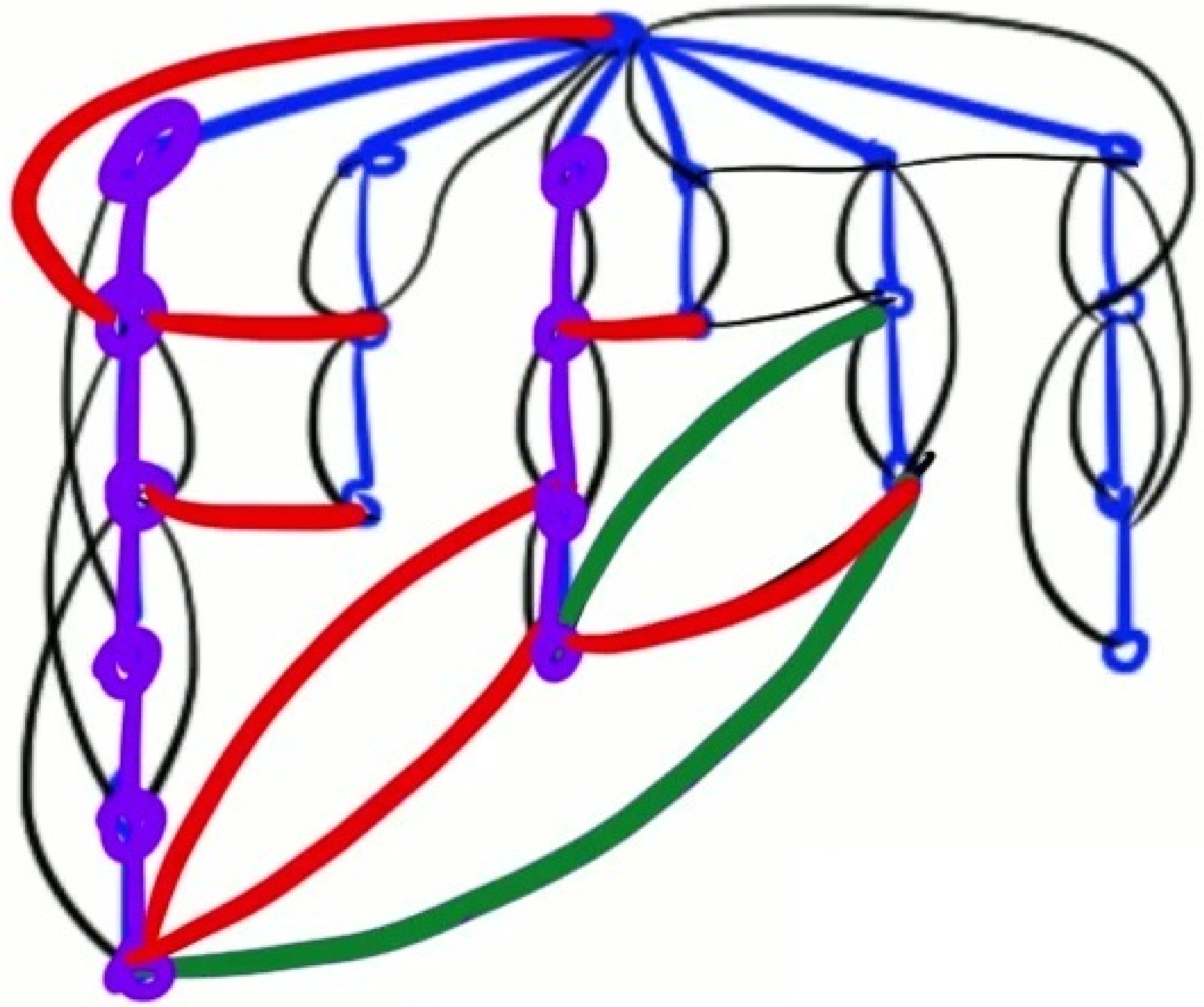
Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green

For each of OPT's branches, remaining boundary edges red

Identifying the Branches

Other extreme: suppose many edges between OPT's branches



Idea: color-code a spanning tree connecting these branches

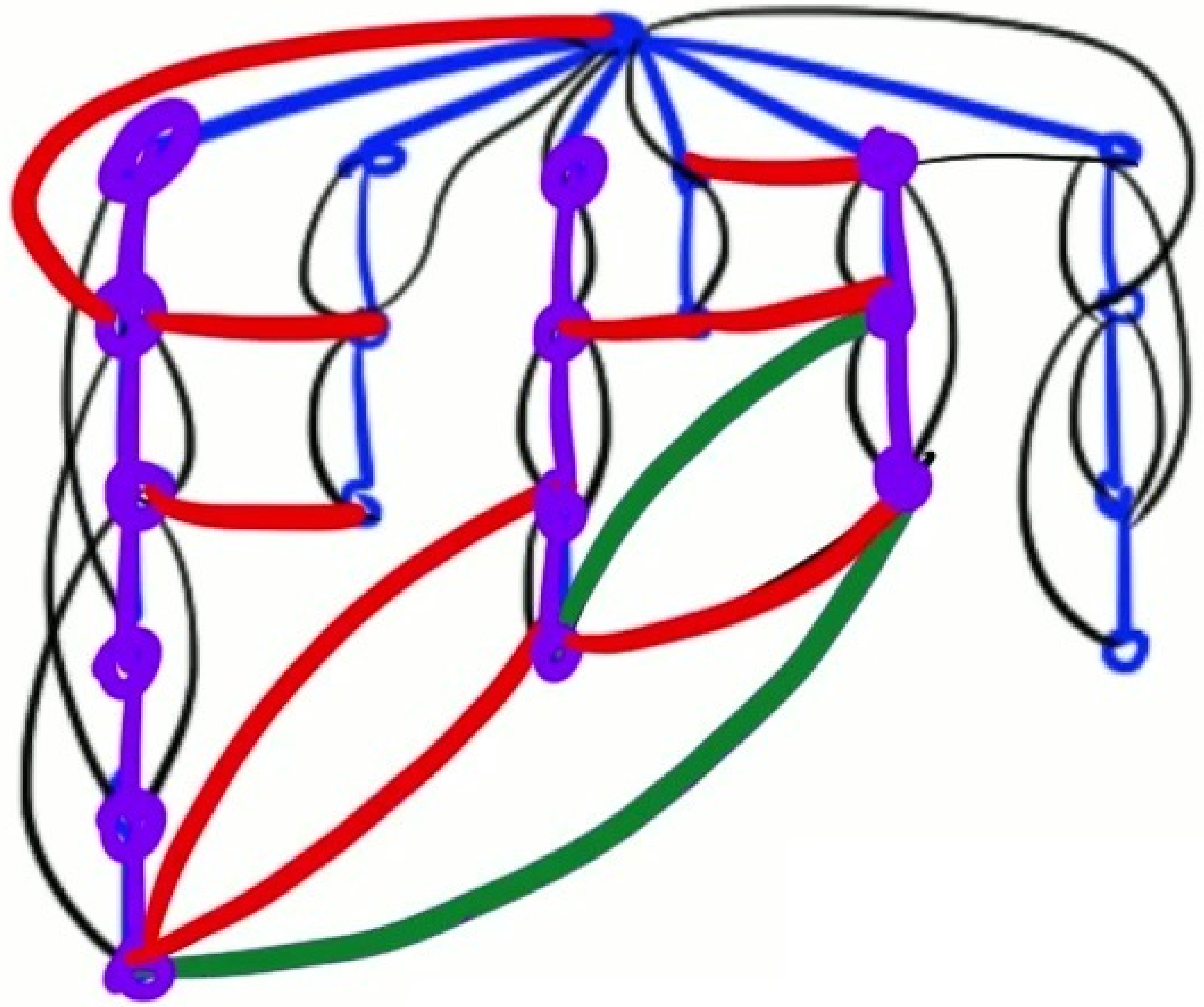
Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green

For each of OPT's branches, remaining boundary edges red

Identifying the Branches

Other extreme: suppose many edges between OPT's branches



Idea: color-code a spanning tree connecting these branches

Color each edge green w.p. p and red w.p. 1-p

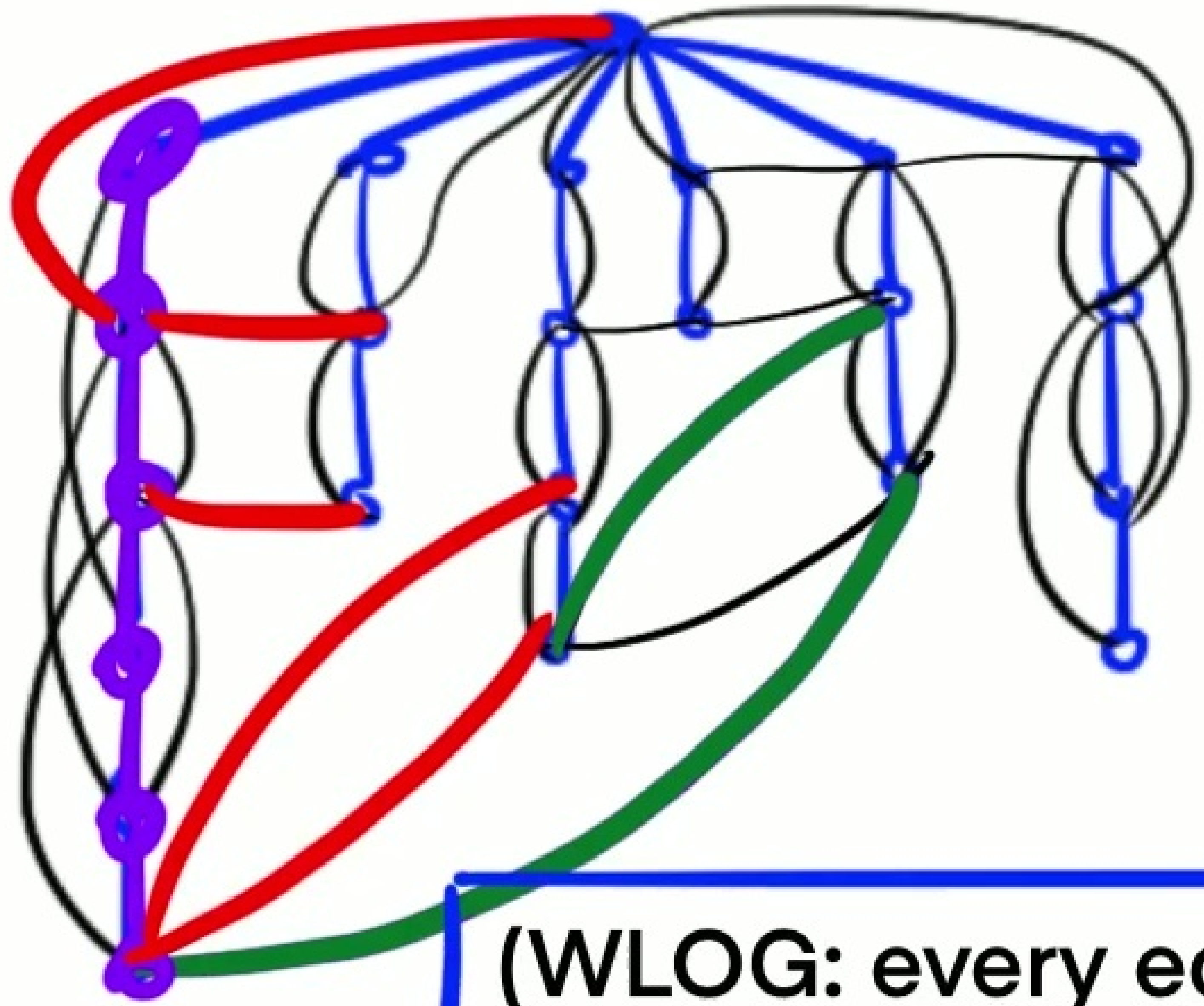
Want: fixed spanning tree all green

For each of OPT's branches, remaining boundary edges red

Identifying the Branches

Other extreme: suppose many edges between OPT's branches

Idea: color-code a spanning tree connecting these branches



Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green

(WLOG: every edge cut $\leq \lambda_k$
Otherwise OPT can't pick that edge, so contract)

For each of OPT's branches, remaining boundary edges red $\leq \lambda_k$ per branch $\Rightarrow \leq k\lambda_k$ total

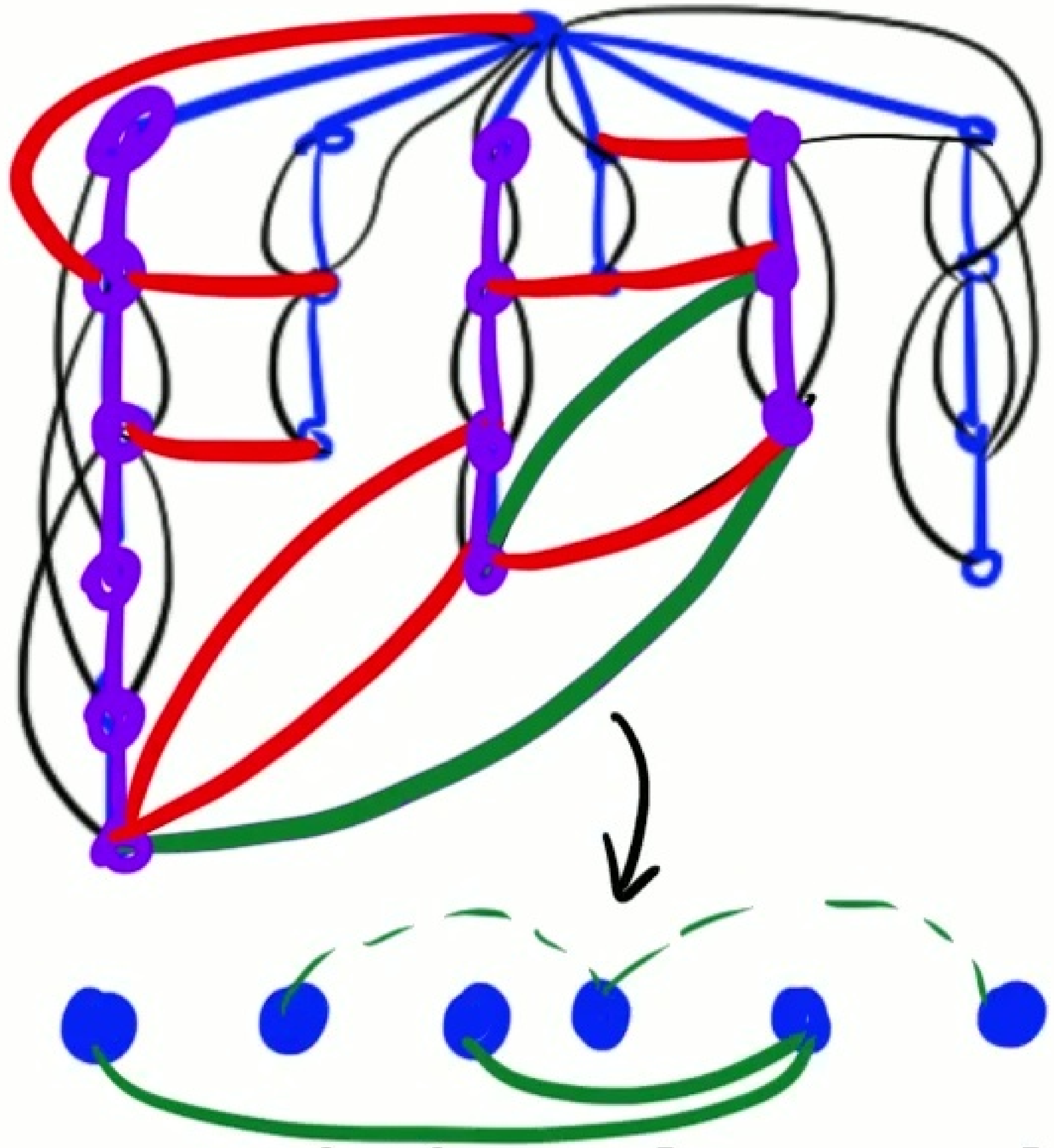
Identifying the Branches

Other extreme: suppose many edges between OPT's branches

Idea: color-code a spanning tree connecting these branches

Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green



For each of OPT's branches, remaining boundary edges red $\leq \lambda_k$ per branch $\Rightarrow \leq k\lambda_k$ total

Contract each branch; OPT's branches is one connected component

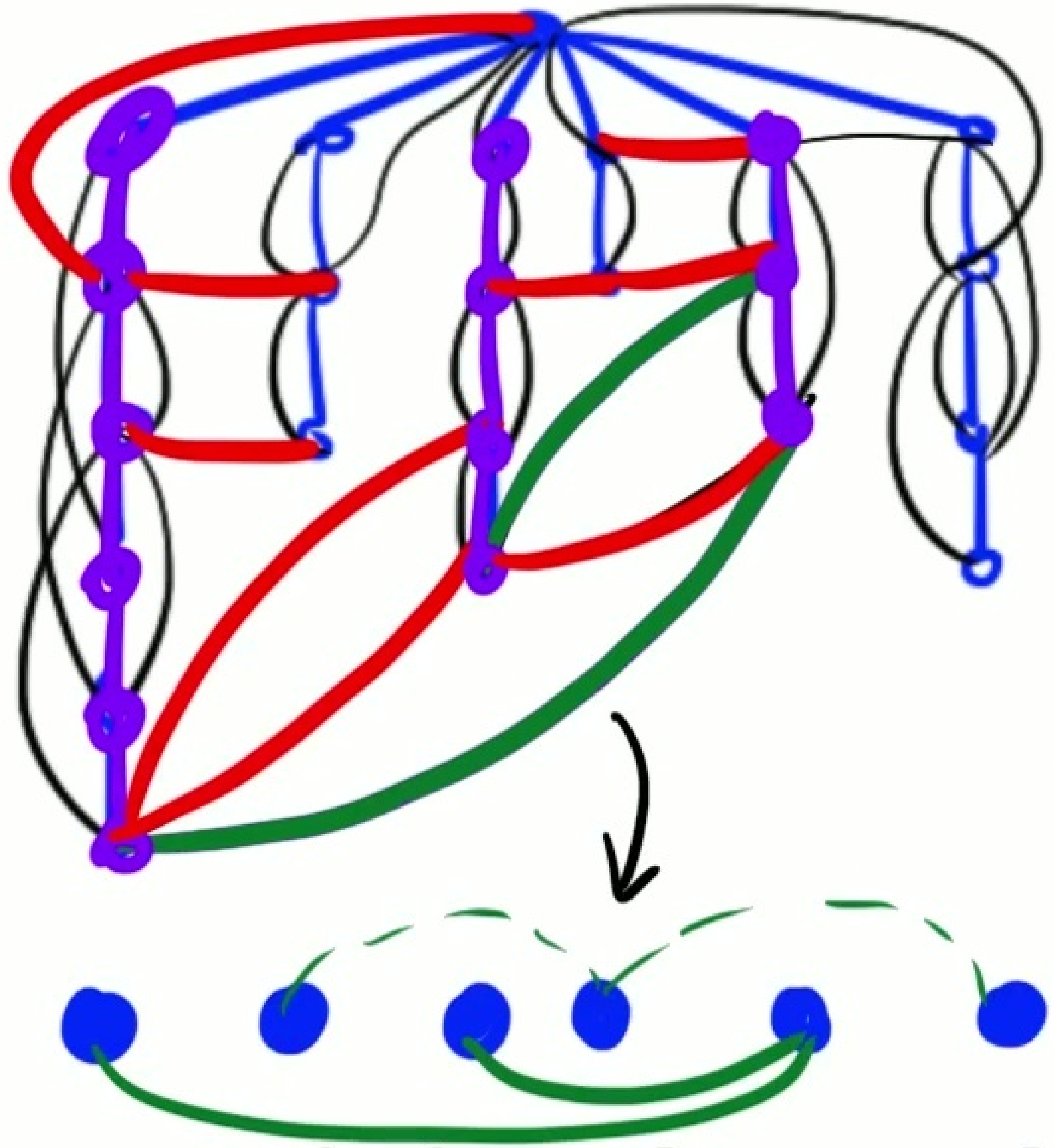
Identifying the Branches

Other extreme: suppose many edges between OPT's branches

Idea: color-code a spanning tree connecting these branches

Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green



success $p^k(1-p)^{k\lambda_k}$

For each of OPT's branches, remaining boundary edges red $\leq \lambda_k$ per branch $\Rightarrow \leq k\lambda_k$ total

Contract each branch; OPT's branches is one connected component

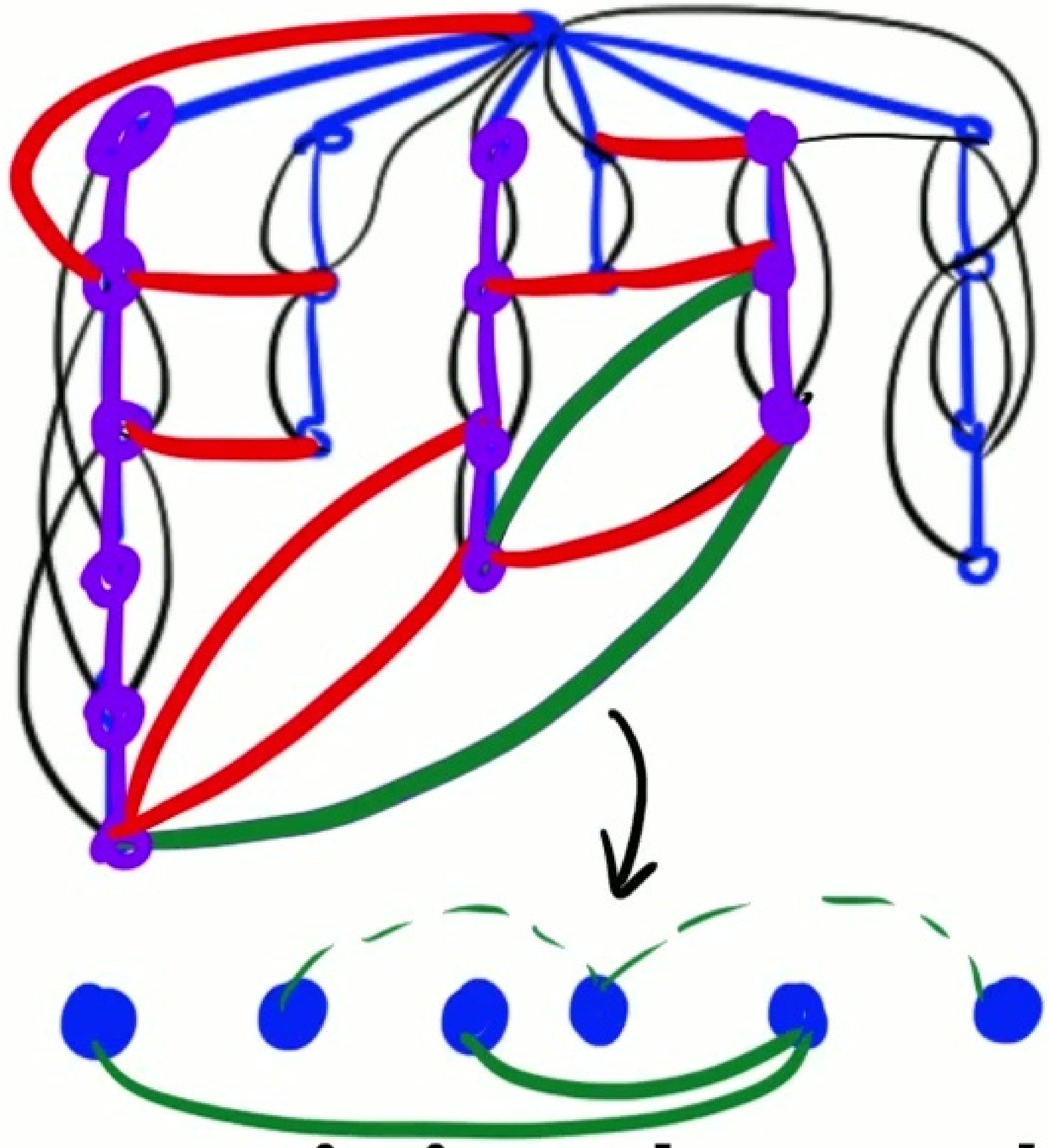
Identifying the Branches

Other extreme: suppose many edges between OPT's branches

Idea: color-code a spanning tree connecting these branches

Color each edge green w.p. p and red w.p. 1-p

Want: fixed spanning tree all green



success

$p^k (1-p)^{k\lambda_k}$

$p = 1/\lambda_k$

$\frac{1}{\lambda_k^k} e^{-k}$

For each of OPT's branches, remaining boundary edges red $\leq \lambda_k$ per branch $\Rightarrow \leq k\lambda_k$ total

Contract each branch; OPT's branches is one connected component

Identifying the Branches

Other extrem... sup... e many edges between OPT's

bran...

Highlight small (size k) structure

Idea... e

conn...

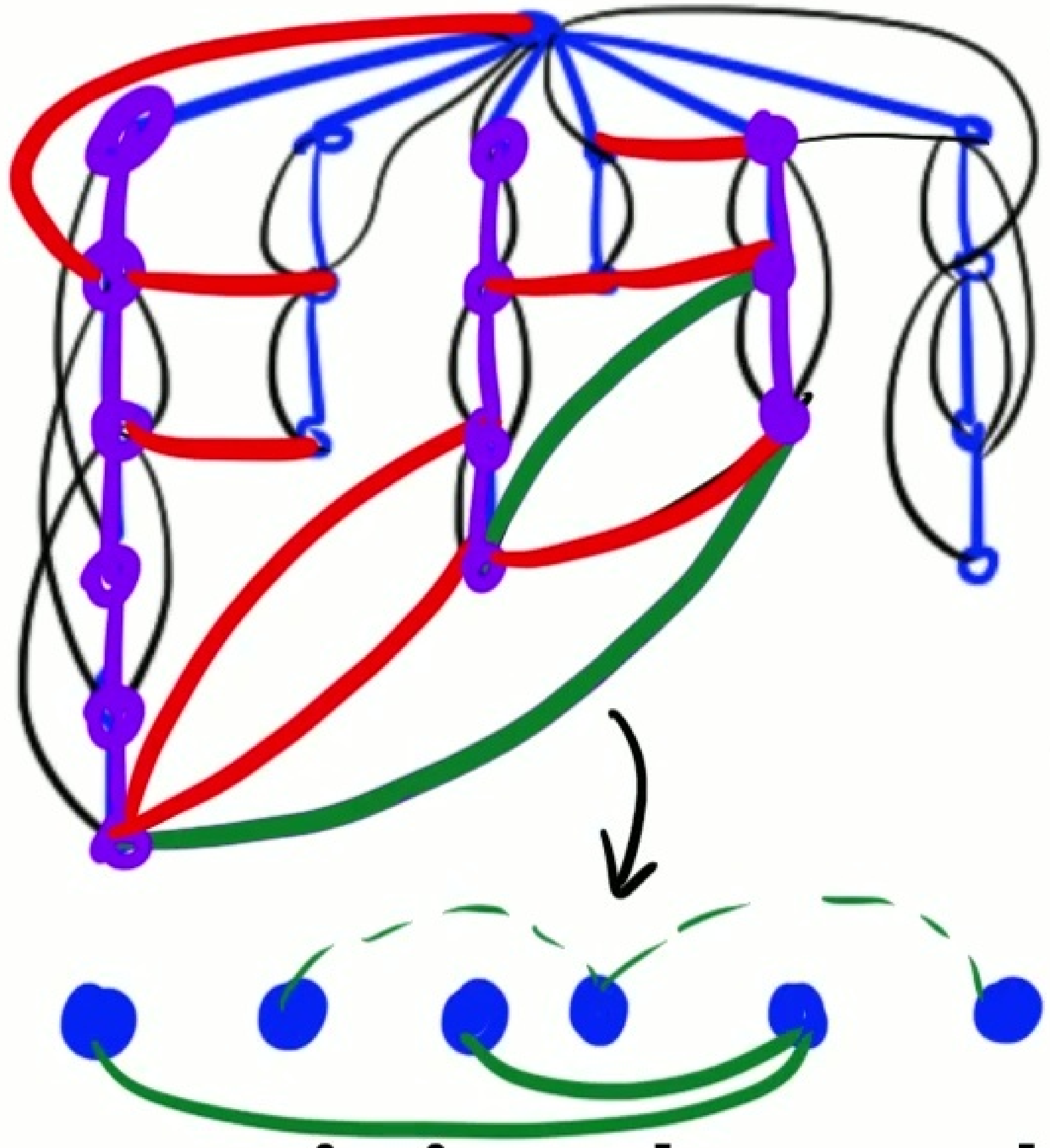
Color each e... g... w.p. p and

red w.p. 1-p

Want: fixed spanning tree all green

For each of OPT's branches, remaining boundary edges red $\leq \lambda_k$ per branch $\Rightarrow \leq k\lambda_k$ total

Contract each branch; OPT's branches is one connected component



success

$p^k (1-p)^{k\lambda_k}$

$p = 1/\lambda_k$

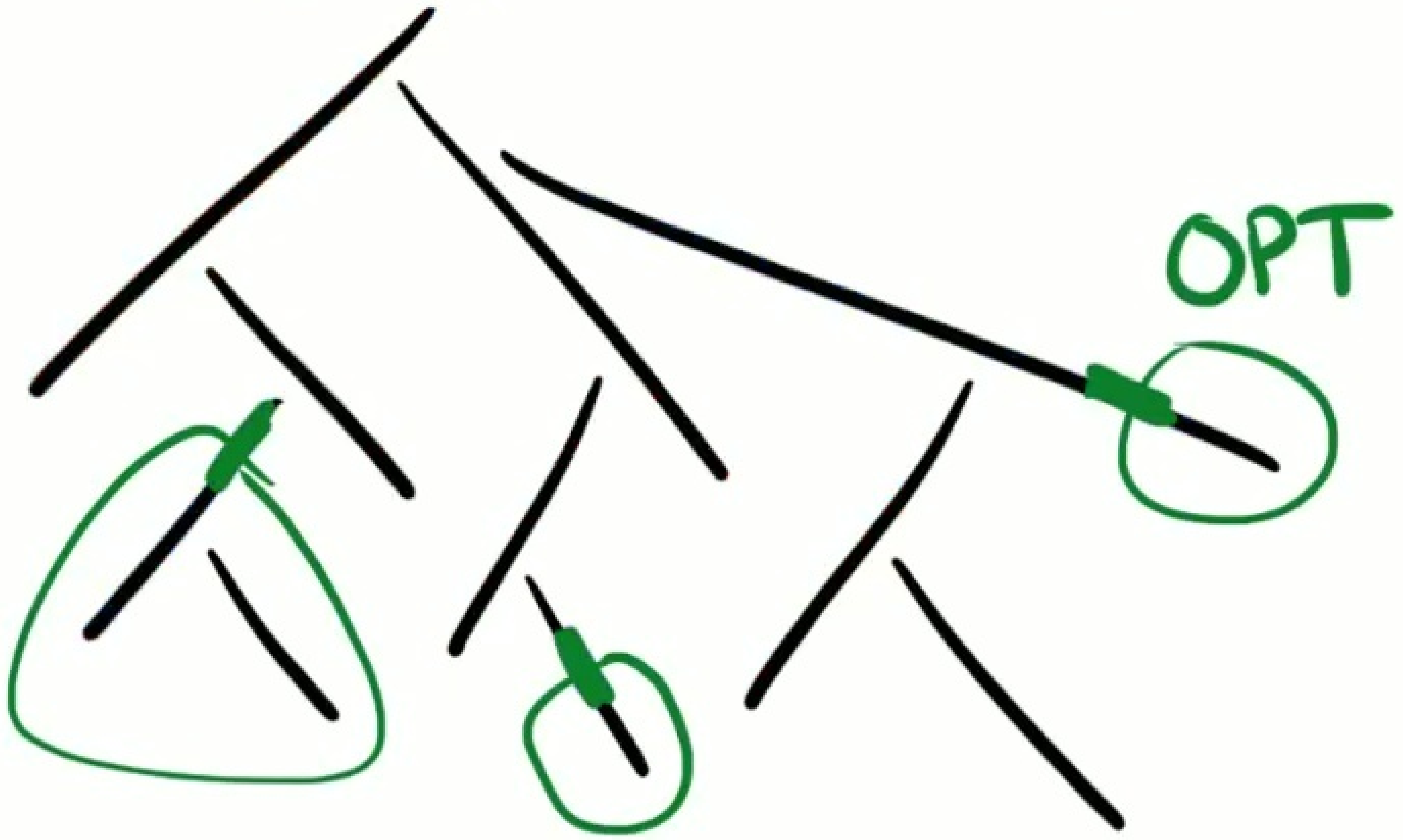
$\frac{1}{\lambda_k^k} e^{-k}$

General Trees

Heavy-light decomposition into branches, color-code red/blue

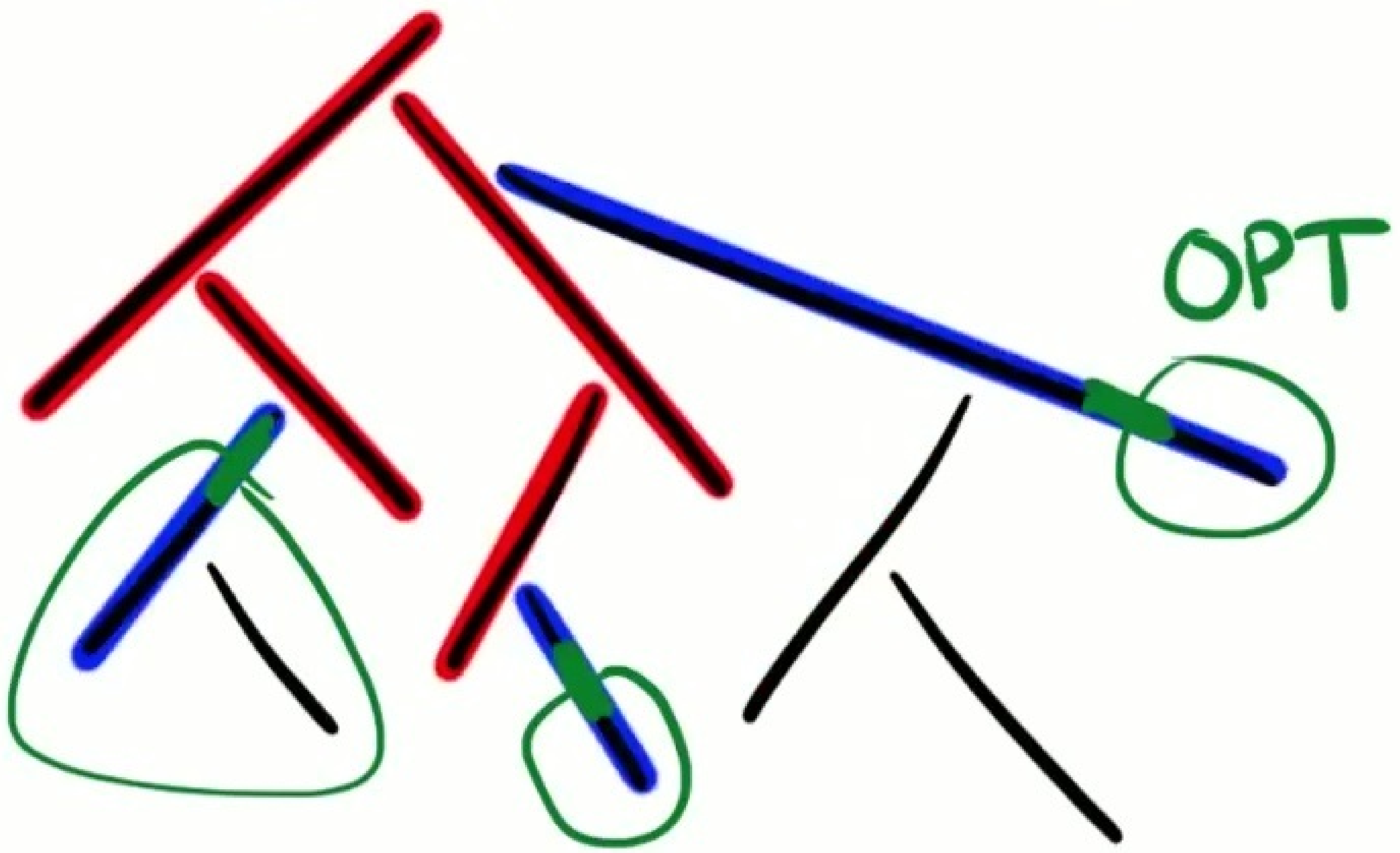
General Trees

Heavy-light decomposition into branches, color-code red/blue



General Trees

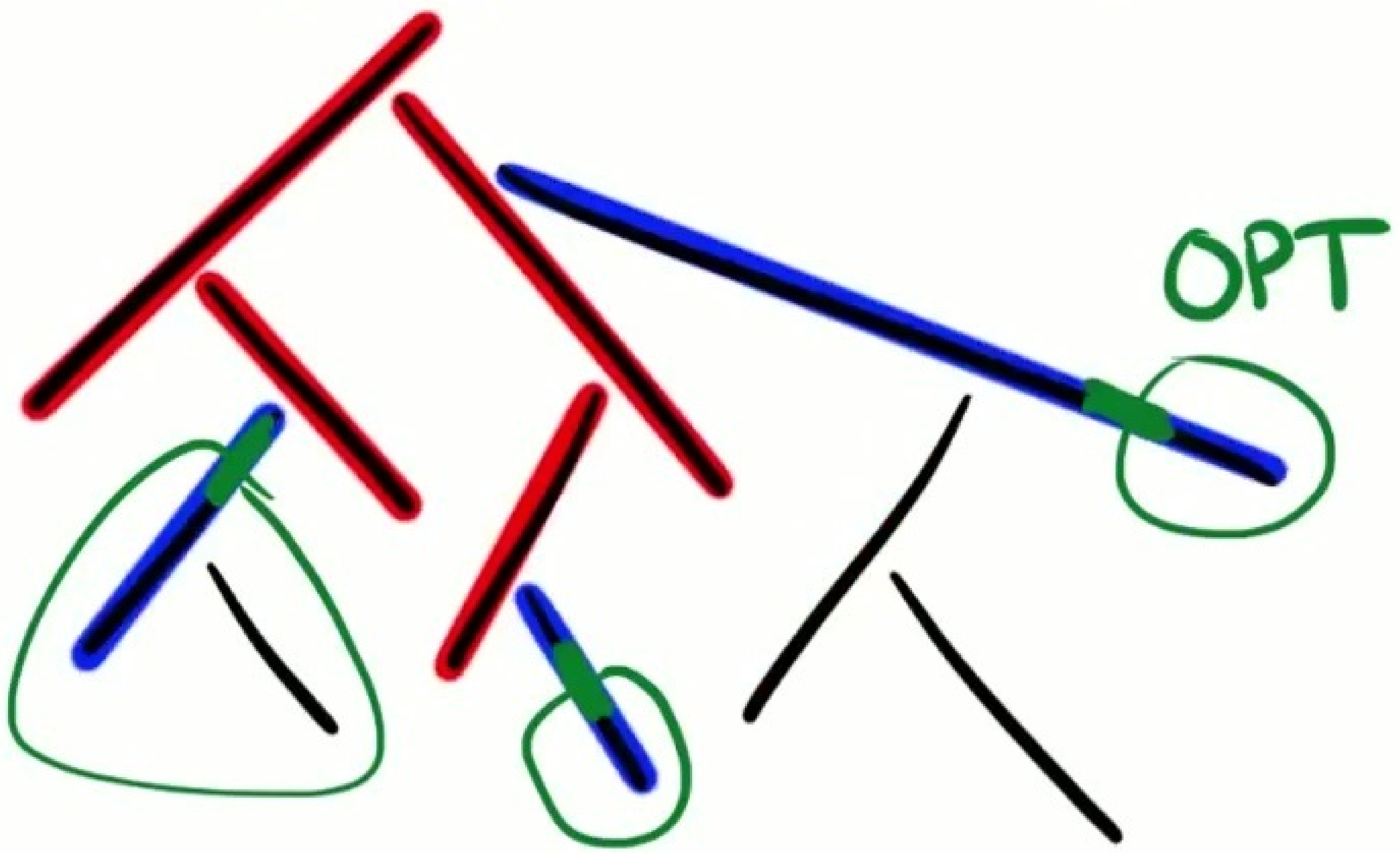
Heavy-light decomposition into branches, color-code red/blue



- Each branch containing an edge cut by OPT is blue
- Their ancestor branches are red
- O(log n) anc. branches per edge (by HLD)

General Trees

Heavy-light decomposition into branches, color-code red/blue

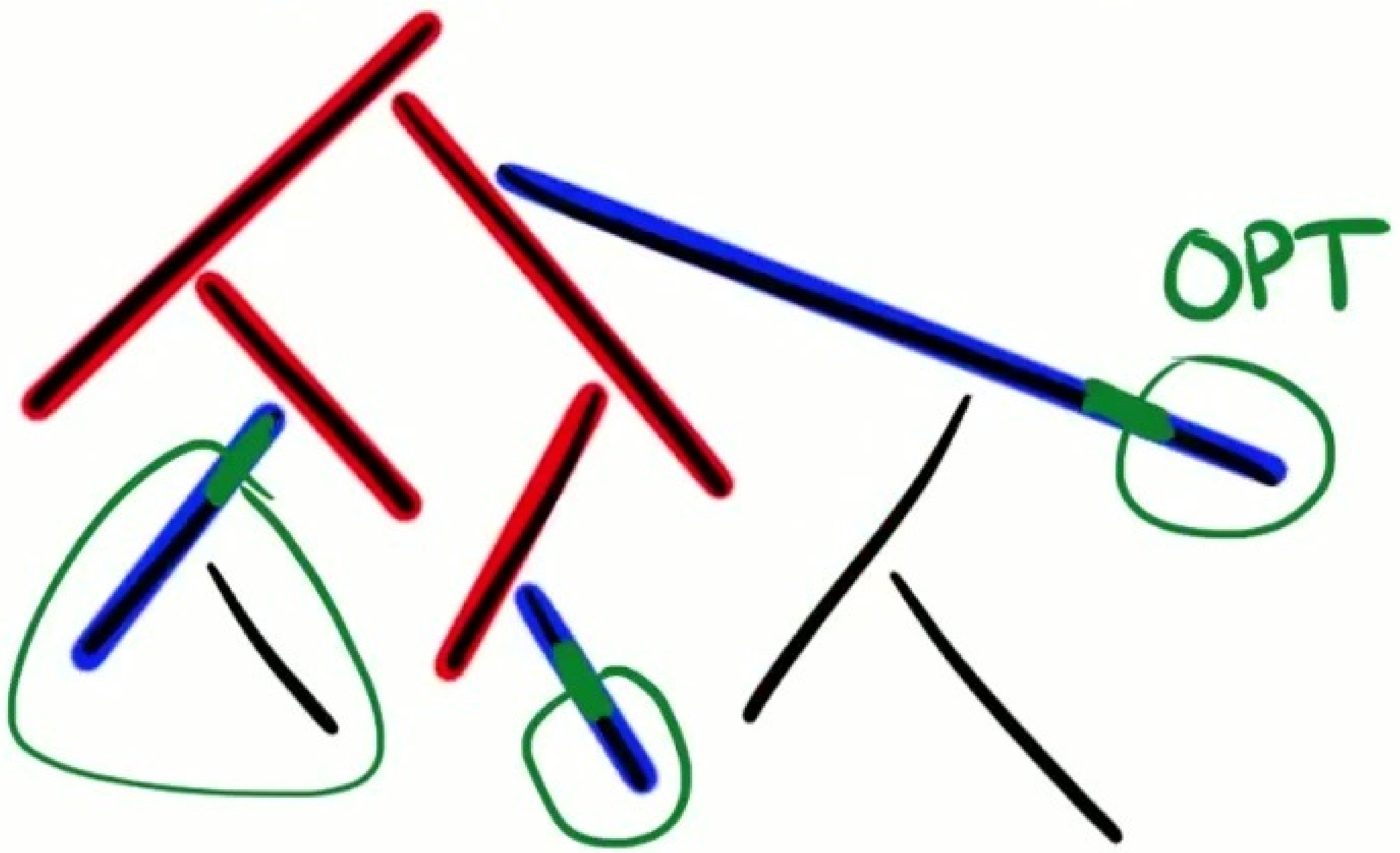


- Each branch containing an edge cut by OPT is blue
- Their ancestor branches are red
- O(log n) anc. branches per edge (by HLD)
- Color blue w.p. 1/logn:

$$\left(\frac{1}{\log n}\right)^k \left(1 - \frac{1}{\log n}\right)^{O(k \log n)} = \frac{1}{\log^k n} e^{-O(k)}$$

General Trees

Heavy-light decomposition into branches, color-code red/blue

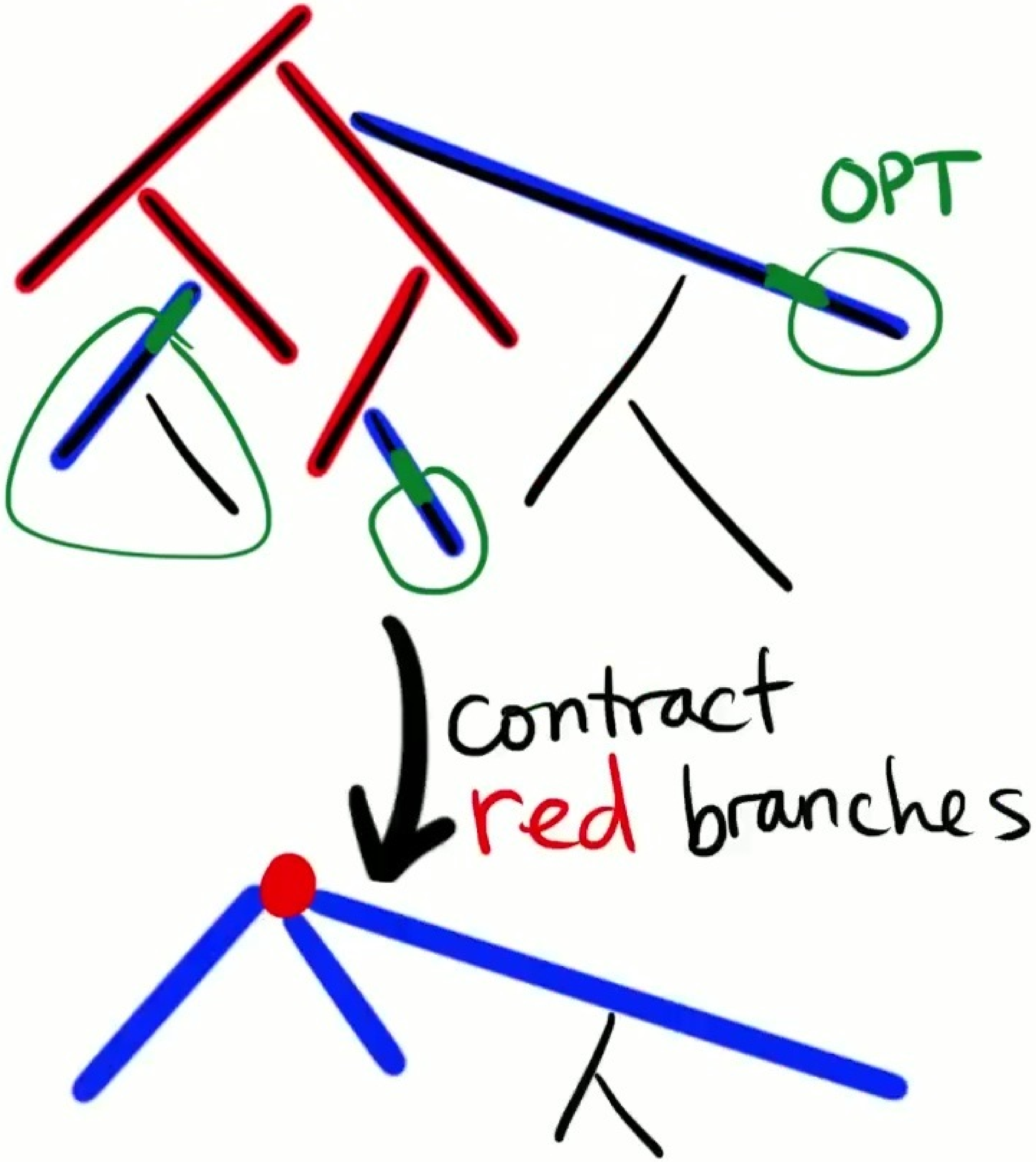


- Each branch containing an edge cut by OPT is blue
- Their ancestor branches are red
- O(log n) anc. branches per edge (by HLD)
- Color blue w.p. 1/logn:

$$\left(\frac{1}{\log n}\right)^k \left(1 - \frac{1}{\log n}\right)^{O(k \log n)} = \frac{1}{\boxed{\log^k n}} e^{-O(k)} \leftarrow \text{FPT(k)}$$

General Trees

Heavy-light decomposition into branches, color-code red/blue

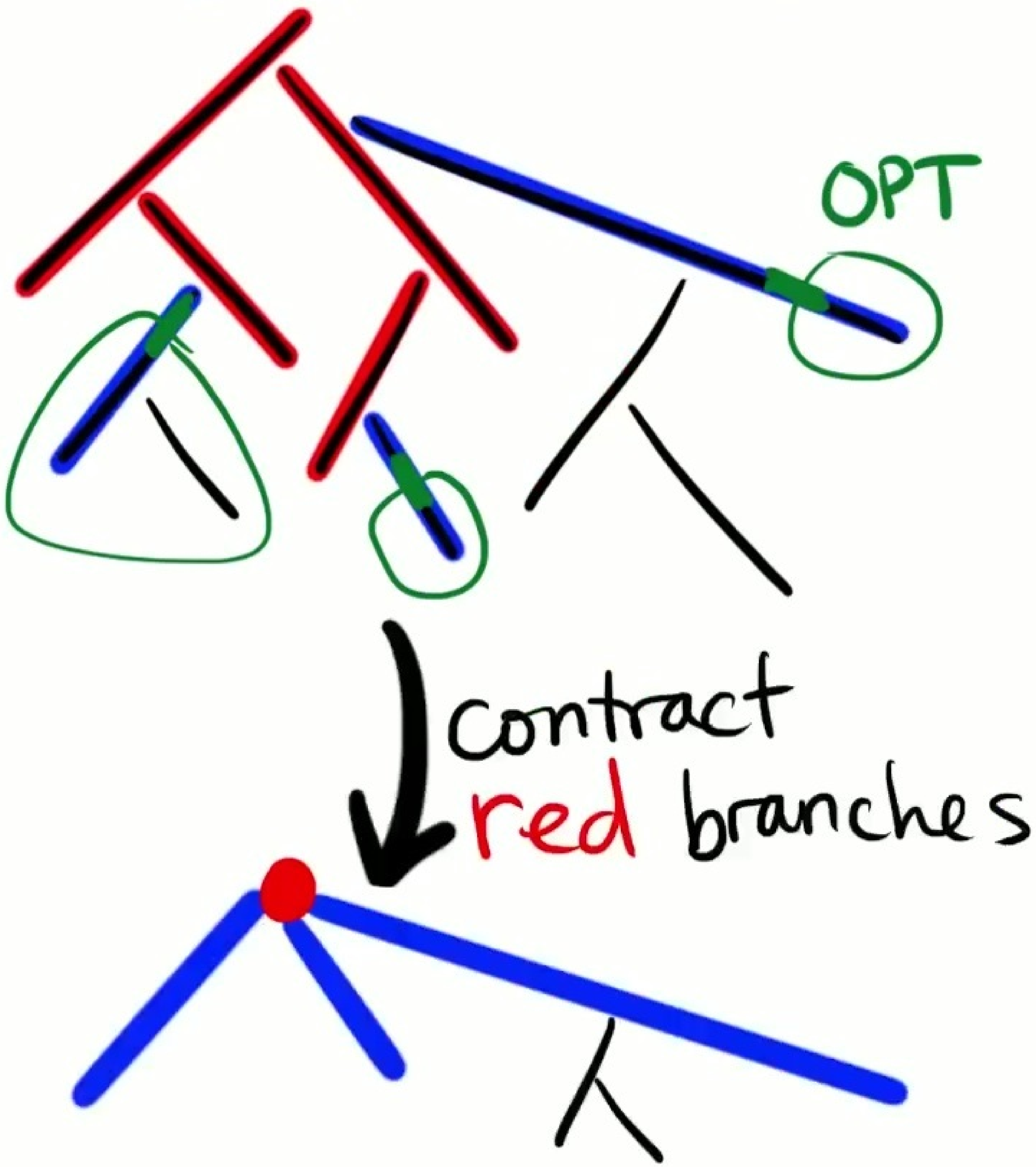


- Each branch containing an edge cut by OPT is blue
- Their ancestor branches are red
- O(log n) anc. branches per edge (by HLD)
- Color blue w.p. 1/logn:

$$\left(\frac{1}{\log n}\right)^{k}\left(1-\frac{1}{\log n}\right)^{O(k \log n)} = \frac{1}{\boxed{\log^k n}} e^{-O(k)} \leftarrow \text{FPT}(k)$$

General Trees

Heavy-light decomposition into branches, color-code red/blue



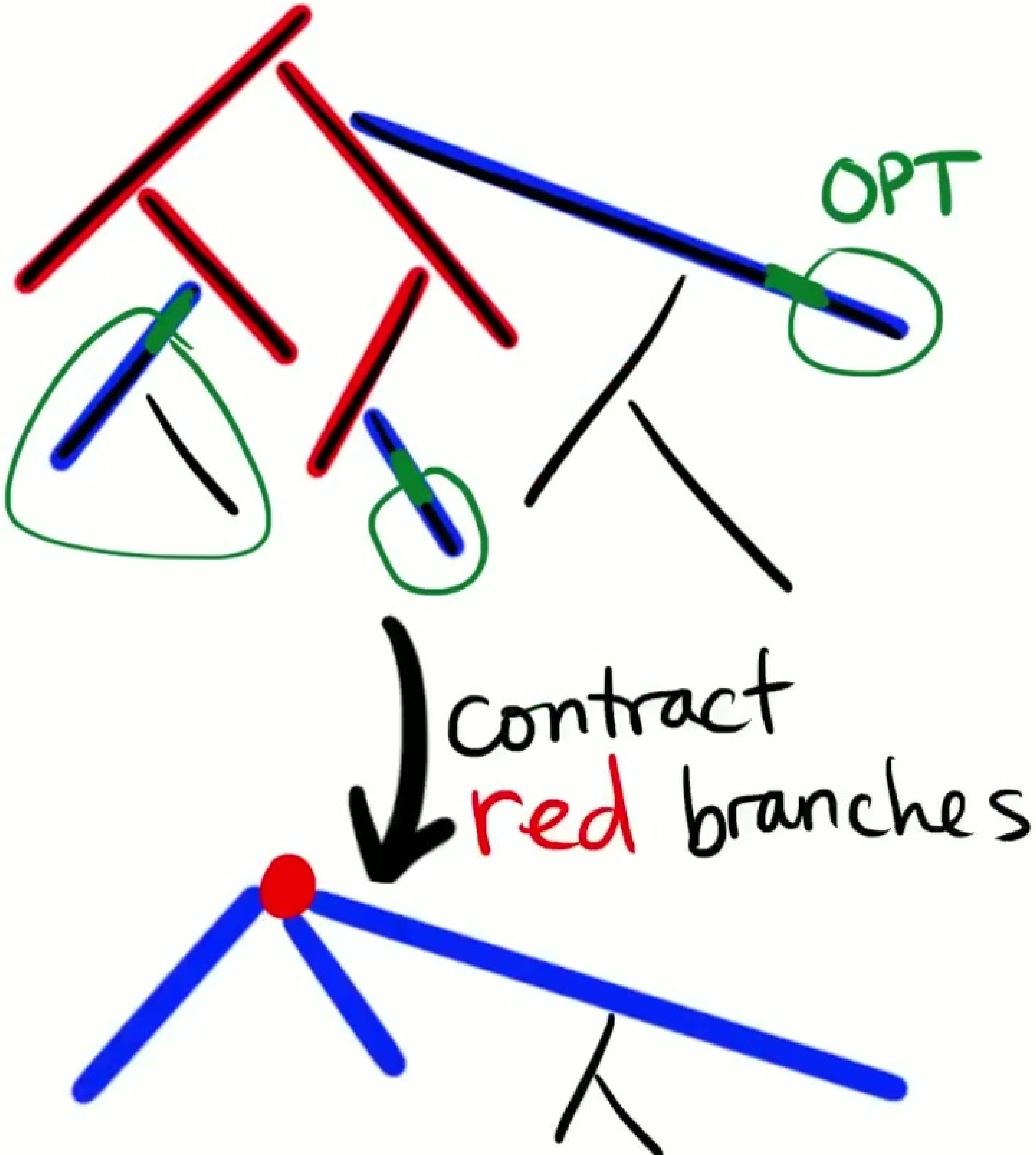
- Each branch containing an edge cut by OPT is blue
- Their ancestor branches are red
- O(log n) anc. branches per edge (by HLD)
- Color blue w.p. 1/logn:

$$\left(\frac{1}{\log n}\right)^k \left(1 - \frac{1}{\log n}\right)^{O(k \log n)} = \frac{1}{\boxed{\log^k n}} e^{-O(k)} \leftarrow \text{FPT}(k)$$

  - Assumes OPT edges "incomparable"

General Trees

Heavy-light decomposition into branches, color-code red/blue



- Each branch containing an edge cut by OPT is blue
- Their ancestor branches are red
- O(log n) anc. branches per edge (by HLD)
- Color blue w.p. 1/logn:

$$\left(\frac{1}{\log n}\right)^{k}\left(1-\frac{1}{\log n}\right)^{O(k\log n)} = \frac{1}{\boxed{\log^k n}}e^{-O(k)} \leftarrow \text{FPT}(k)$$

  - Assumes OPT edges "incomparable"

Reduce to incomparable: Dynamic program on subtrees [GLL18]

Future directions?

Faster min k-cut on a weighted graph?

Future directions?

Faster min k-cut on a weighted graph?

[Gupta Lee L] min k-cut in time $n^k 2^{O(\log\log n)^2}$

Future directions?

Faster min k-cut on a weighted graph?

[Gupta Lee L] min k-cut in time $n^k 2^{O(\log\log n)^2}$

- The Karger-Stein algorithm outputs any fixed min k-cut with probability $n^{-k} 2^{-O(\log\log n)^2}$!

[Improve from n^{-2k}]

Future directions?

Faster min k-cut on a weighted graph?

[Gupta Lee L] min k-cut in time $n^k 2^{O((\log\log n)^2)}$

(TCS+ talk, Nov. 20)

- The Karger-Stein algorithm outputs any fixed min k-cut with probability $n^{-k} 2^{-O((\log\log n)^2)}$!

[Improve from n^{-2k}]

Future directions?

Faster min k-cut on a weighted graph?

[Gupta Lee L] min k-cut in time $n^k 2^{O(\log\log n)^2}$

(TCS+ talk, Nov. 20)

- The Karger-Stein algorithm outputs any fixed min k-cut with probability $n^{-k} 2^{-O(\log\log n)^2}$!

[Improve from n^{-2k}]

Deterministic n^k time?