Local weighted optimizations and open problems

Benoît Corsini
with L. Addario-Berry
and J. Barrett

Table of contents

Table of contents

Motivation

Motivation

The fundamental question

The fundamental question

When considering real-life networks, it is often impossible to access the whole graph at once.

Motivation

The fundamental question

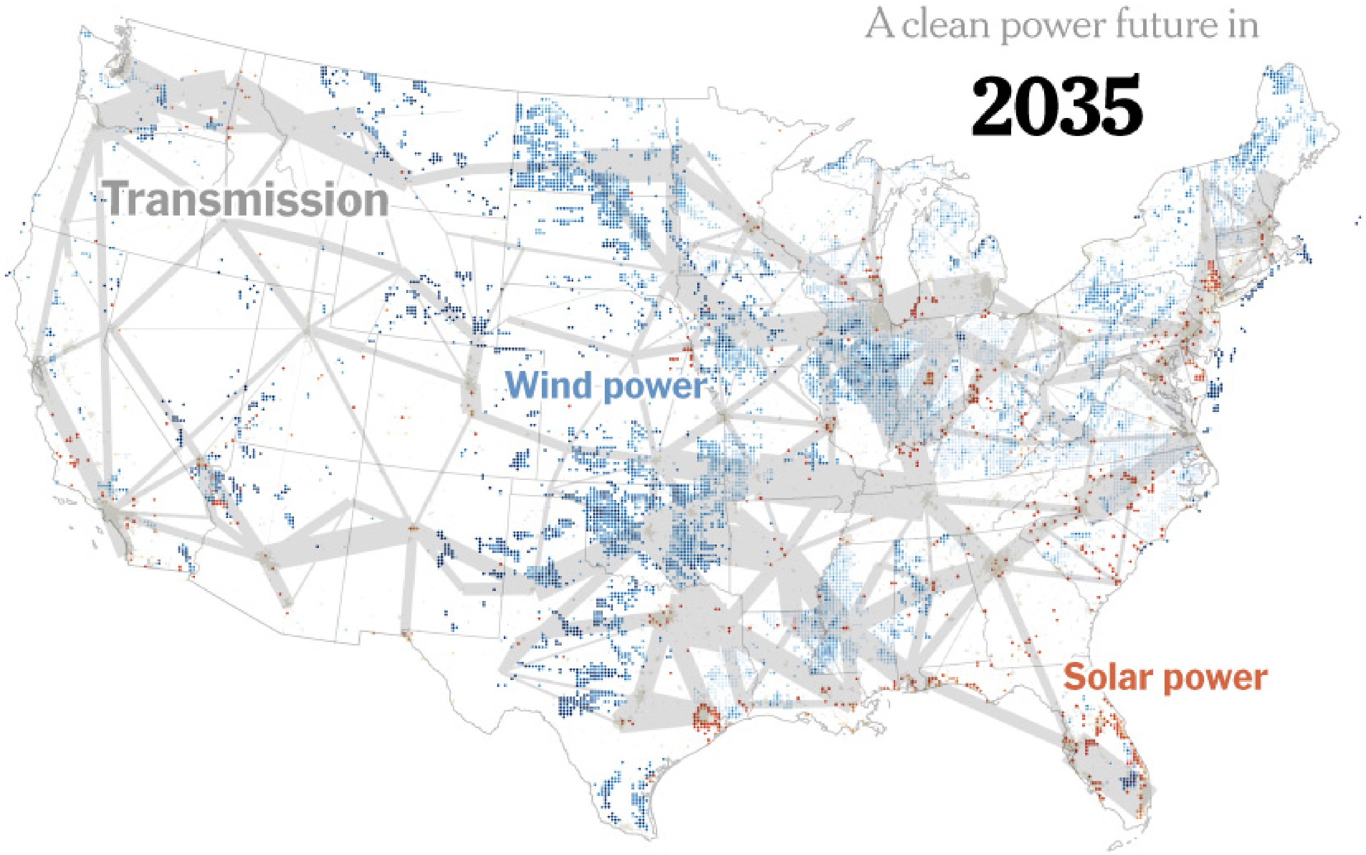
When considering real-life networks, it is often impossible to access the whole graph at once.
So if we further want to modify the graph, this becomes even more complicated...

The fundamental question

When considering real-life networks, it is often impossible to access the whole graph at once.
So if we further want to modify the graph, this becomes even more complicated...

Assume we have a target property that we want our network to satisfy; can we operate "local" modifications eventually leading to the global graph satisfying this property?

Motivation



Source: mivolink.blogspot.com

Modernizing Canada's Aging Power Grid

by Powertec Electric | Apr 20, 2019 | Electrical Power, Electricians, Hiring Electricians | 0 comments

In the 70s and 80s, there was a lot of investment into electrical infrastructure in Canada. New technologies were demanding higher electrical capacity in homes, and the growth of Canada's large urban centres meant that demand was sure to remain high. The surge of investment into the grid was so monumental that supply actually ended up outweighing demand, and electricity could be bought on the cheap for many years. These investments have sustained us for quite some time, but we may now be reaching the breaking point of our electric grid.

Source: powertec.ca

Out setup

Out setup

In the case of the electrical grid, we

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and
- were hoping to make it become “more robust”.

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and
- were hoping to make it become “more robust”.

We adapt this example and now consider

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and
- were hoping to make it become “more robust”.

We adapt this example and now consider

- nodes with iid distances;

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and
- were hoping to make it become “more robust”.

We adapt this example and now consider

- nodes with iid distances;
- an arbitrary starting graph;

Out setup

In the case of the electrical grid, we

- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and
- were hoping to make it become “more robust”.

We adapt this example and now consider

- nodes with iid distances;
- an arbitrary starting graph;
- a budget corresponding to the weight of the graph; and

Out setup

In the case of the electrical grid, we

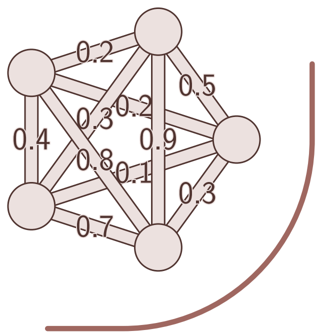
- were operating on a 2-dimensional Euclidian plan;
- started with a given network on this plan;
- had a yearly budget allowing us to modify only parts of our graph; and
- were hoping to make it become "more robust".

We adapt this example and now consider

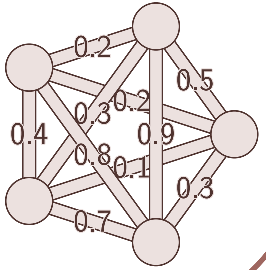
- nodes with iid distances;
- an arbitrary starting graph;
- a budget corresponding to the weight of the graph; and
- the minimum spanning tree as the target graph.

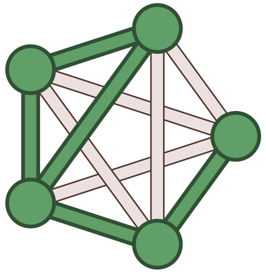
Example

Example

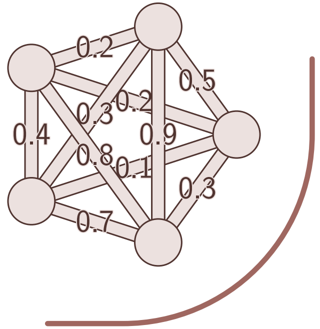


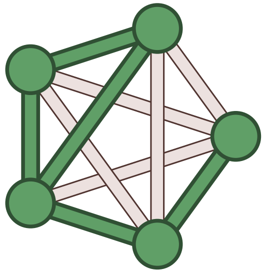
Example

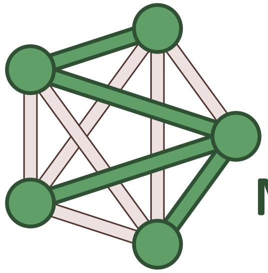


$H =$  $, \ \lambda = 1$

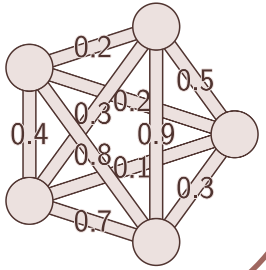
Example

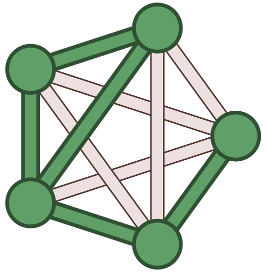
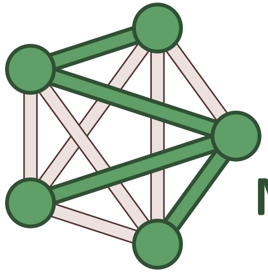


$H =$  $, \ \lambda = 1$

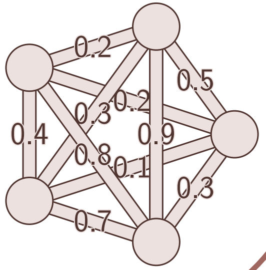


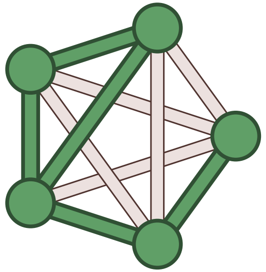
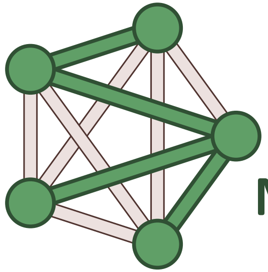
Example

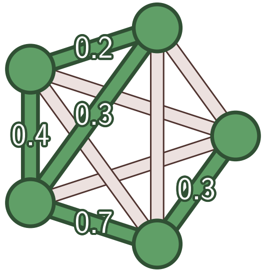


$H =$  $, \quad \lambda = 1 \quad \xrightarrow{\quad ? \quad}$  **MST**

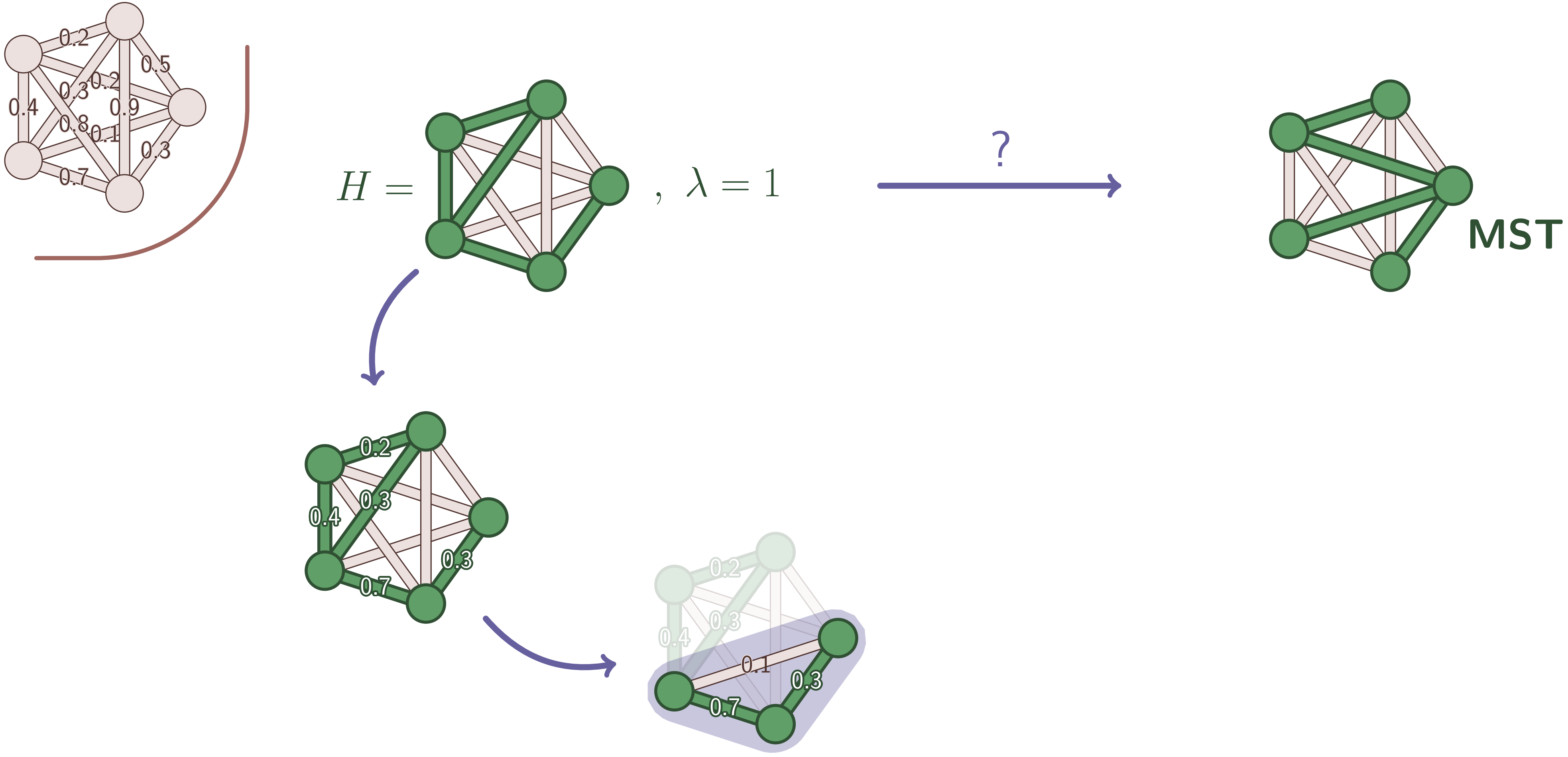
Example



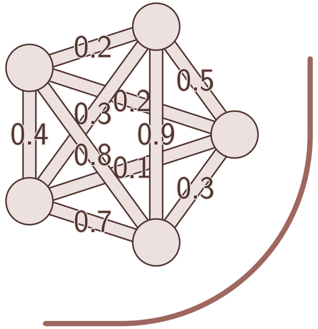
$H =$  $, \quad \lambda = 1$ $\xrightarrow{?}$  **MST**

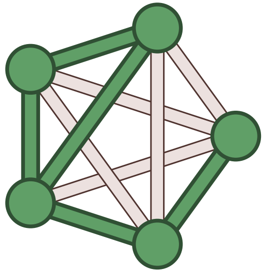


Example

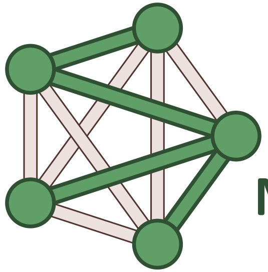

0.2
0.5
0.3
0.2
0.4
0.9
0.8
0.1
0.3
0.7
$H =$
, $\lambda = 1$
?
MST
0.2
0.3
0.4
0.3
0.7
0.2
0.3
0.4
0.1
0.3
0.7

Example

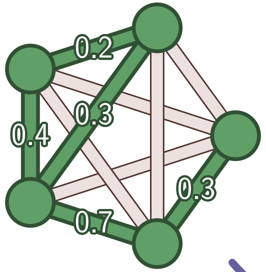


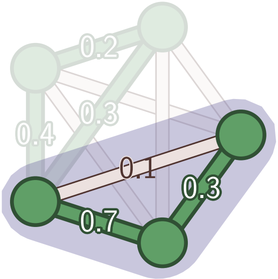
$H =$  $, \quad \lambda = 1$

?



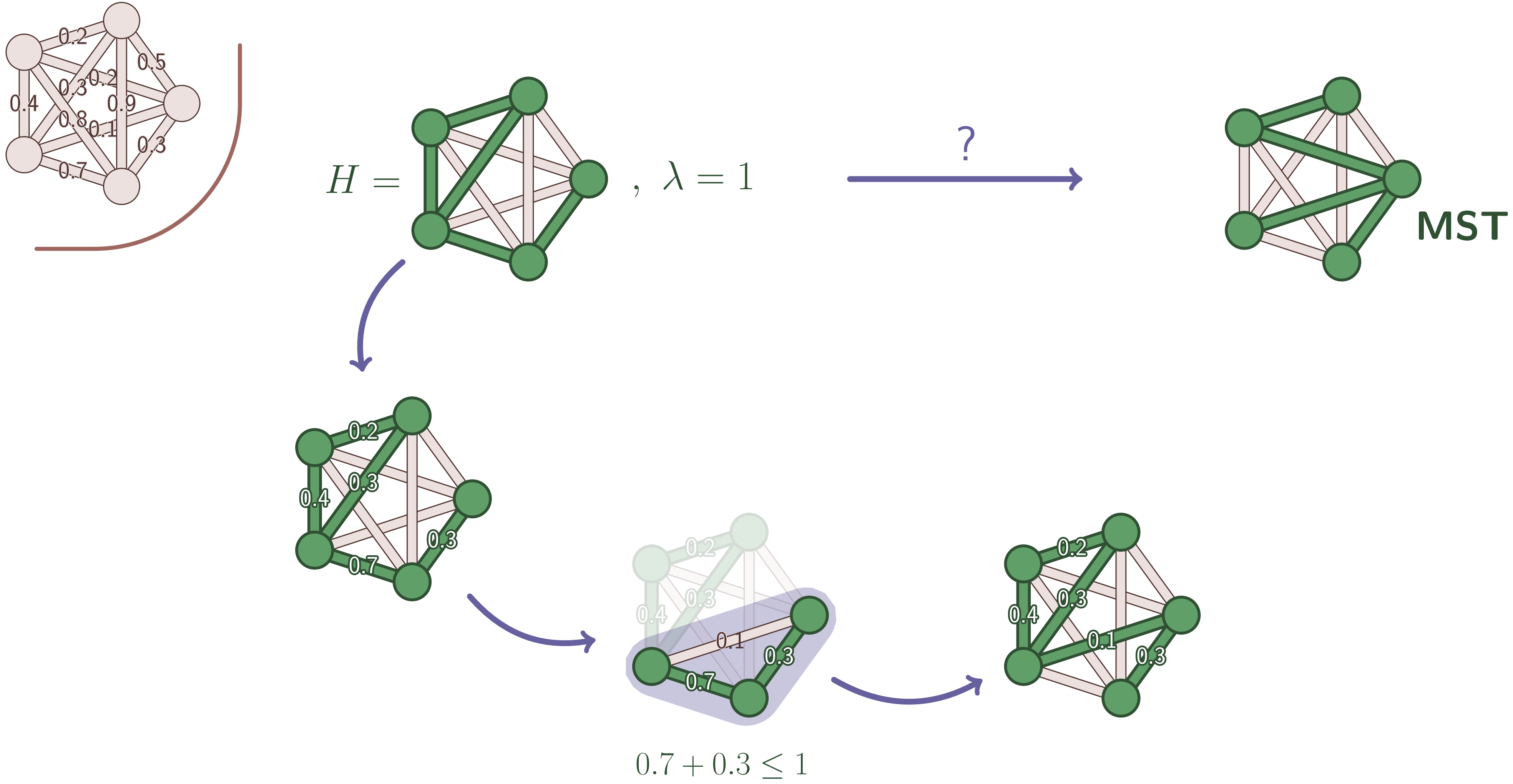
MST



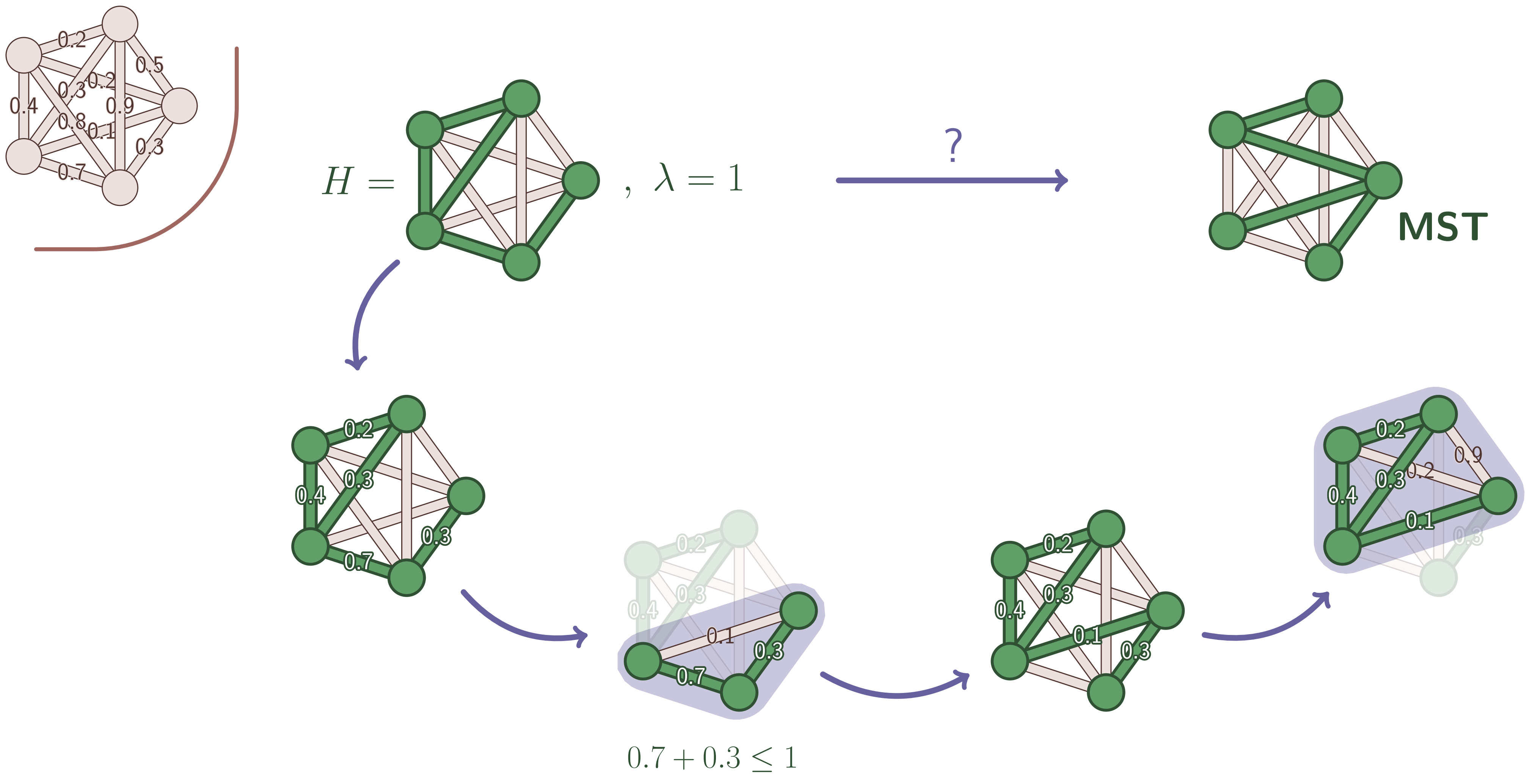


$$0.7 + 0.3 \leq 1$$

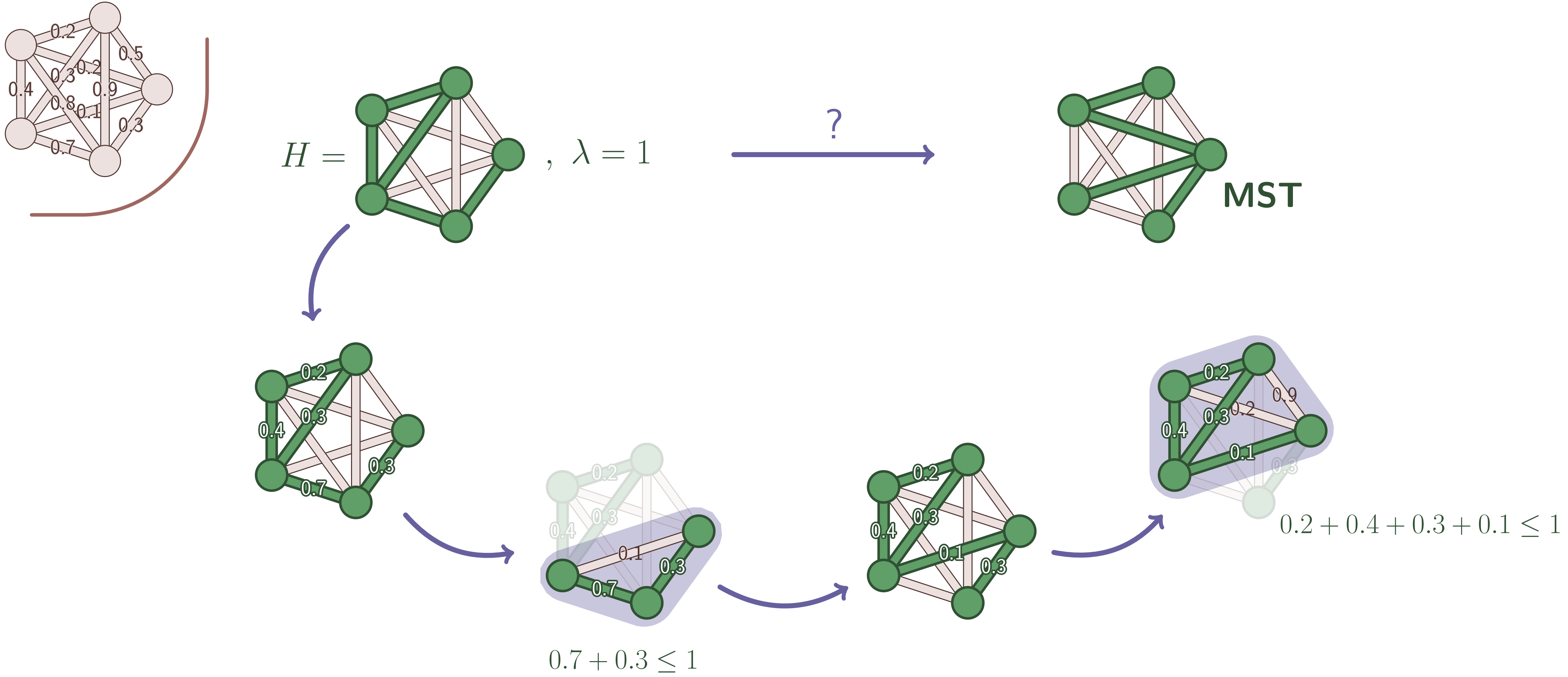
Example



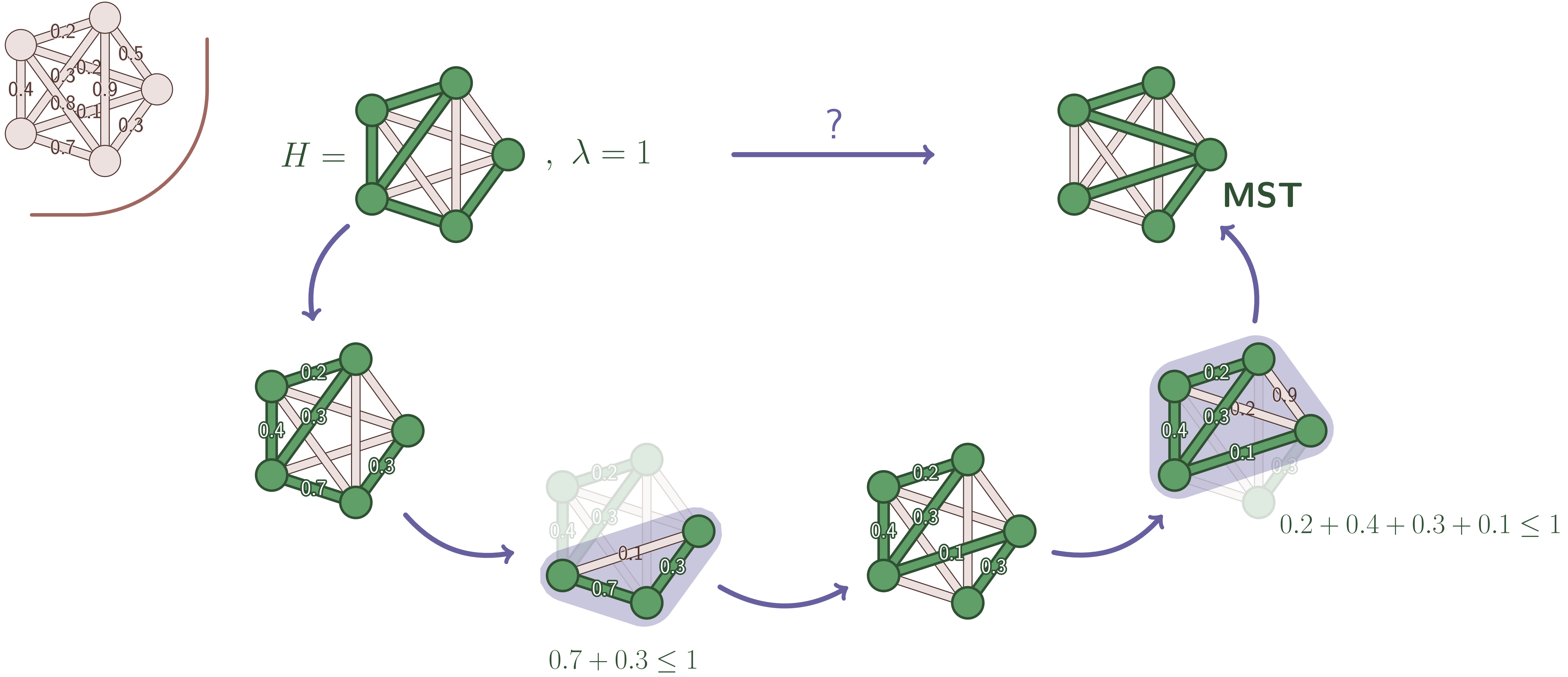
Example



Example



Example


0.2
0.5
0.3
0.2
0.4
0.9
0.8
0.1
0.3
0.7
$H =$
$, \lambda = 1$
?
MST
$0.7 + 0.3 \leq 1$
$0.2 + 0.4 + 0.3 + 0.1 \leq 1$

Definition

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

- $H_{n,0} = H_n$; and

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

- $H_{n,0} = H_n$; and
- $H_{n,i}$ is obtained by replacing $H_{n,i-1}[S_i]$ on $H_{n,i-1}$ by its (local) minimum spanning tree.

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

- $H_{n,0} = H_n$; and
- $H_{n,i}$ is obtained by replacing $H_{n,i-1}[S_i]$ on $H_{n,i-1}$ by its (local) minimum spanning tree.

Definition (Optimization)

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

- $H_{n,0} = H_n$; and
- $H_{n,i}$ is obtained by replacing $H_{n,i-1}[S_i]$ on $H_{n,i-1}$ by its (local) minimum spanning tree.

Definition (Optimization)

Say that $\mathbb{S} = (S_1, \ldots, S_k)$ is an optimization with respect to (H_n, λ) if

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

- $H_{n,0} = H_n$; and
- $H_{n,i}$ is obtained by replacing $H_{n,i-1}[S_i]$ on $H_{n,i-1}$ by its (local) minimum spanning tree.

Definition (Optimization)

Say that $\mathbb{S} = (S_1, \ldots, S_k)$ is an optimization with respect to (H_n, λ) if

- $H_{n,k}$ is the (global) minimum spanning tree of $\mathbb{K}_n$; and

Definition

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent $\textsc{Uniform}([0,1])$ random edge weights. Let H_n be a spanning subgraph of K_n and $\lambda > 0$ be a positive number.

For any sequence of sets $\mathbb{S} = (S_1, \ldots, S_k)$, define $(H_{n,0}, \ldots, H_{n,k})$ as follows.

- $H_{n,0} = H_n$; and
- $H_{n,i}$ is obtained by replacing $H_{n,i-1}[S_i]$ on $H_{n,i-1}$ by its (local) minimum spanning tree.

Definition (Optimization)

Say that $\mathbb{S} = (S_1, \ldots, S_k)$ is an optimization with respect to (H_n, λ) if

- $H_{n,k}$ is the (global) minimum spanning tree of $\mathbb{K}_n$; and
- for any i, the weight of $H_{n,i-1}[S_i]$ is less than λ.

Properties

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.
- This process can always end up on a "local minimum" (by adding more sets to the sequence).

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.
- This process can always end up on a "local minimum" (by adding more sets to the sequence).
- Given $\mathbb{K}_n$ and H_n, there exists a threshold $\lambda_{\mathrm{thr}} = \lambda_{\mathrm{thr}}(H_n; \mathbb{K}_n)$ such that

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.
- This process can always end up on a "local minimum" (by adding more sets to the sequence).
- Given $\mathbb{K}_n$ and H_n, there exists a threshold $\lambda_{\text{thr}} = \lambda_{\text{thr}}(H_n; \mathbb{K}_n)$ such that
    - if $\lambda < \lambda_{\text{thr}}$, then there exists no optimization with respect to (H_n, λ); and

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.
- This process can always end up on a "local minimum" (by adding more sets to the sequence).
- Given $\mathbb{K}_n$ and H_n, there exists a threshold $\lambda_{\text{thr}} = \lambda_{\text{thr}}(H_n; \mathbb{K}_n)$ such that
    - if $\lambda < \lambda_{\text{thr}}$, then there exists no optimization with respect to (H_n, λ); and
    - if $\lambda > \lambda_{\text{thr}}$, then there exists an optimization with respect to (H_n, λ).

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.
- This process can always end up on a "local minimum" (by adding more sets to the sequence).
- Given $\mathbb{K}_n$ and H_n, there exists a threshold $\lambda_{\text{thr}} = \lambda_{\text{thr}}(H_n; \mathbb{K}_n)$ such that
  - if $\lambda < \lambda_{\text{thr}}$, then there exists no optimization with respect to (H_n, λ); and
  - if $\lambda > \lambda_{\text{thr}}$, then there exists an optimization with respect to (H_n, λ).
- It is easy to check that λ_{thr} is larger than the heaviest edge in H_n not in the (global) minimum spanning tree and smaller than the total weight of H_n.

Properties

The existence of an optimization with respect to (H_n, λ) means that it is possible to transform H_n into the (global) minimum spanning tree on $\mathbb{K}_n$ by inductively replacing subgraphs of weight less than λ into (local) optimally weighted trees.

- In the previous example, there exists an optimization with respect to $(H, 1)$.
- If H_n is not a tree, its number of edges will regularly decrease in the process.
- This process can always end up on a "local minimum" (by adding more sets to the sequence).
- Given $\mathbb{K}_n$ and H_n, there exists a threshold $\lambda_{\text{thr}} = \lambda_{\text{thr}}(H_n; \mathbb{K}_n)$ such that
  - if $\lambda < \lambda_{\text{thr}}$, then there exists no optimization with respect to (H_n, λ); and
  - if $\lambda > \lambda_{\text{thr}}$, then there exists an optimization with respect to (H_n, λ).
- It is easy to check that λ_{thr} is larger than the heaviest edge in H_n not in the (global) minimum spanning tree and smaller than the total weight of H_n.

→ We hope to characterize λ_{thr} when n is large for various choices of H_n.

Table of contents

The ONE result

The ONE result

Theorem (Addario-Berry, Barrett, 🧍 [2022])

The ONE result

Theorem (Addario-Berry, Barrett, [2022])

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent uniform edge weights, H_n be a spanning subgraph of K_n chosen independently of $\mathbb{U}$, and $\varepsilon > 0$.

The ONE result

Theorem (Addario-Berry, Barrett, [2022])

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent uniform edge weights, H_n be a spanning subgraph of K_n chosen independently of $\mathbb{U}$, and $\varepsilon > 0$.

Then, with high probability as n goes to infinity:

The ONE result

Theorem (Addario-Berry, Barrett, [2022])

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent uniform edge weights, H_n be a spanning subgraph of K_n chosen independently of $\mathbb{U}$, and $\varepsilon > 0$.

Then, with high probability as n goes to infinity:

- there exists an optimization with respect to $(H_n, 1+\varepsilon)$; and

The ONE result

Theorem (Addario-Berry, Barrett, [2022])

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent uniform edge weights, H_n be a spanning subgraph of K_n chosen independently of $\mathbb{U}$, and $\varepsilon > 0$.

Then, with high probability as n goes to infinity:

- there exists an optimization with respect to $(H_n, 1+\varepsilon)$; and
- there does not exist any optimization with respect to $(H_n, 1-\varepsilon)$.

The ONE result

Theorem (Addario-Berry, Barrett, [2022])

Let $\mathbb{K}_n = (K_n, \mathbb{U})$ be the complete weighted graph with independent uniform edge weights, H_n be a spanning subgraph of K_n chosen independently of $\mathbb{U}$, and $\varepsilon > 0$.

Then, with high probability as n goes to infinity:

- there exists an optimization with respect to $(H_n, 1 + \varepsilon)$; and
- there does not exist any optimization with respect to $(H_n, 1 - \varepsilon)$.

→ There is a universal threshold at 1, no matter the structure (or density) of H_n.

Table of contents

Lower bound

Lower bound

The lower bound is actually straightforward to check.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.
- If none of the sets contain both ends of e, then e belongs to the final graph.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.
- If none of the sets contain both ends of e, then e belongs to the final graph.
- Otherwise, the first subgraph containing e has weight at least $1 - \epsilon$.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.
- If none of the sets contain both ends of e, then e belongs to the final graph.
- Otherwise, the first subgraph containing e has weight at least $1 - \epsilon$.

→ There does not exist an optimization with respect to $(H_n, 1 - \epsilon)$.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.
- If none of the sets contain both ends of e, then e belongs to the final graph.
- Otherwise, the first subgraph containing e has weight at least $1 - \epsilon$.

$\rightarrow$ There does not exist an optimization with respect to $(H_n, 1 - \epsilon)$.

I now focus on the upper bound, more technical, but more constructive.

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.
- If none of the sets contain both ends of e, then e belongs to the final graph.
- Otherwise, the first subgraph containing e has weight at least $1 - \epsilon$.

→ There does not exist an optimization with respect to $(H_n, 1 - \epsilon)$.

I now focus on the upper bound, more technical, but more constructive.

→ Given a graph H_n, can we find a sequence of sets transforming H_n into the (global) MST?

Lower bound

The lower bound is actually straightforward to check.

- Since H_n is chosen independently of $\mathbb{U}$, it has an edge e with weight $1 - o_{\mathbb{P}}(1) \geq 1 - \epsilon$.
- This edge is likely not in the (global) MST.
- If none of the sets contain both ends of e, then e belongs to the final graph.
- Otherwise, the first subgraph containing e has weight at least $1 - \epsilon$.

→ There does not exist an optimization with respect to $(H_n, 1 - \epsilon)$.

I now focus on the upper bound, more technical, but more constructive.

→ Given a graph H_n, can we find a sequence of sets transforming H_n into the (global) MST?

→ Can we show that the maximal weight of these sets is not too large (i.e. $\leq 1 + \epsilon$)?

Upper bound proof structure

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.
- A case-by-case proof for these three cases.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.
- A case-by-case proof for these three cases.

→ The last argument is the most complex and detail-oriented.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.
- A case-by-case proof for these three cases.

→ The last argument is the most complex and detail-oriented.

→ I will only explain the first two points.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.
- A case-by-case proof for these three cases.

→ The last argument is the most complex and detail-oriented.

→ I will only explain the first two points.

  - For simplicity, I now drop the subscript n on H_n.

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.
- A case-by-case proof for these three cases.

→ The last argument is the most complex and detail-oriented.

→ I will only explain the first two points.

  - For simplicity, I now drop the subscript n on H_n.
  - I will keep assuming that things are "large enough".

Upper bound proof structure

The proof of the upper bound can be decomposed in three parts.

- The "eating algorithm", a method for locally growing MST.
- A Ramsey-like argument to reduce the study from any H_n to only three cases.
- A case-by-case proof for these three cases.

→ The last argument is the most complex and detail-oriented.

→ I will only explain the first two points.

  - For simplicity, I now drop the subscript n on H_n.
  - I will keep assuming that things are "large enough".
  - Every pair of nodes has an independent uniform weight, even those not part of H.

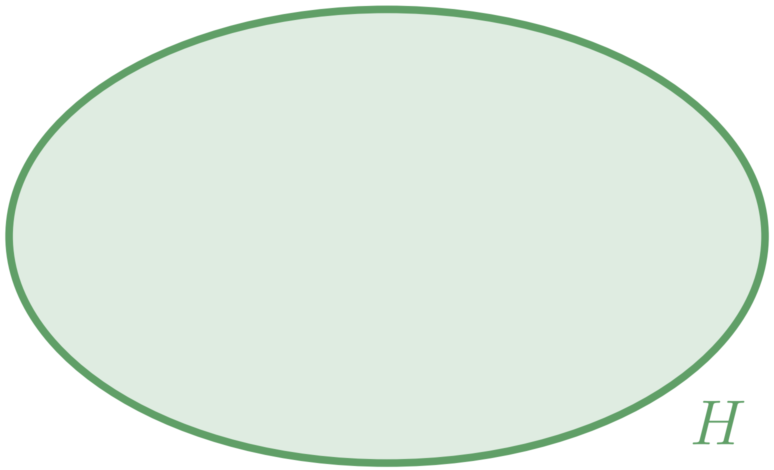
The eating algorithm

The eating algorithm

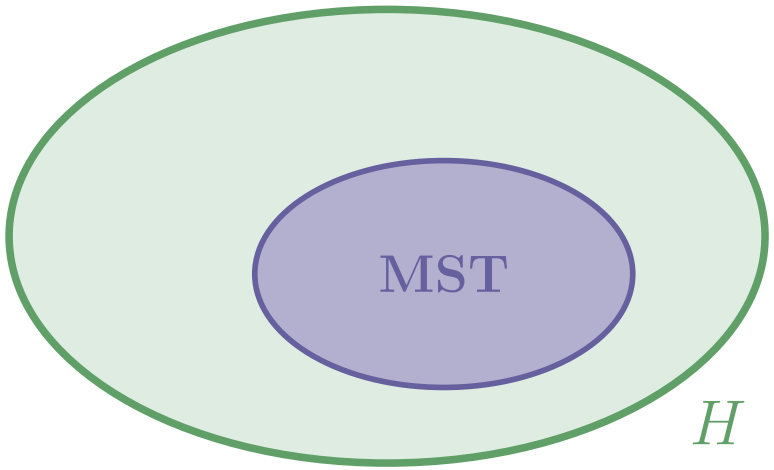
Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.



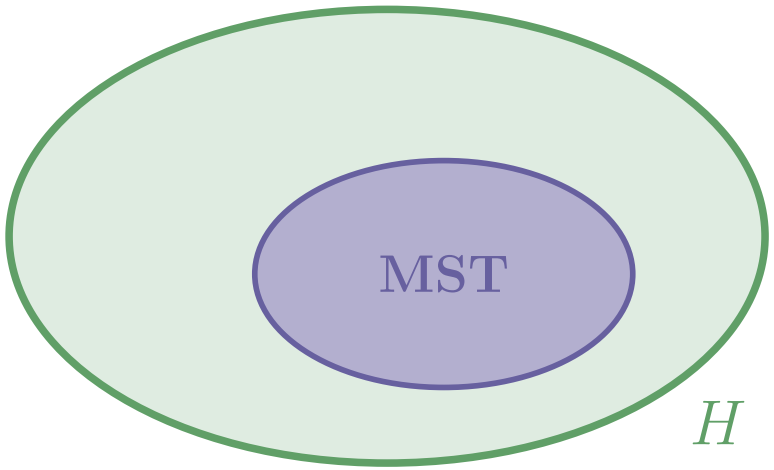
Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.



The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

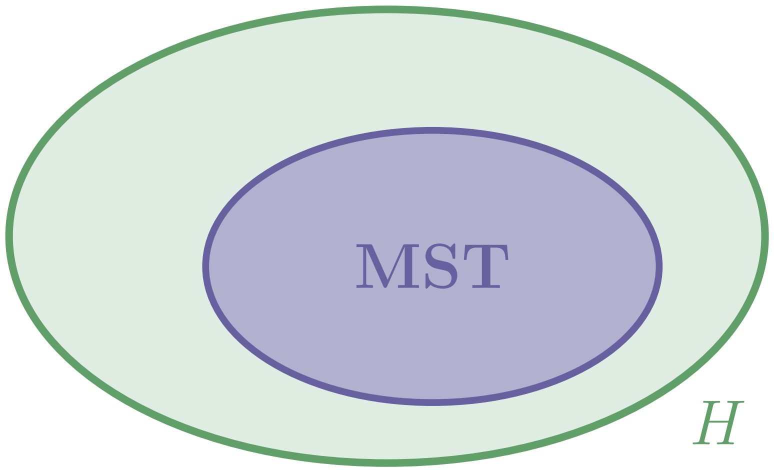
→ Can we extend this MST so that it keeps "eating" nodes?



The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

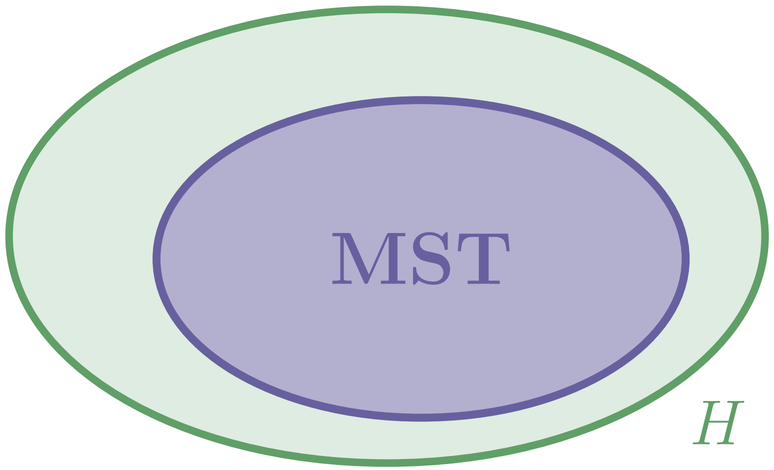
→ Can we extend this MST so that it keeps "eating" nodes?



The eating algorithm

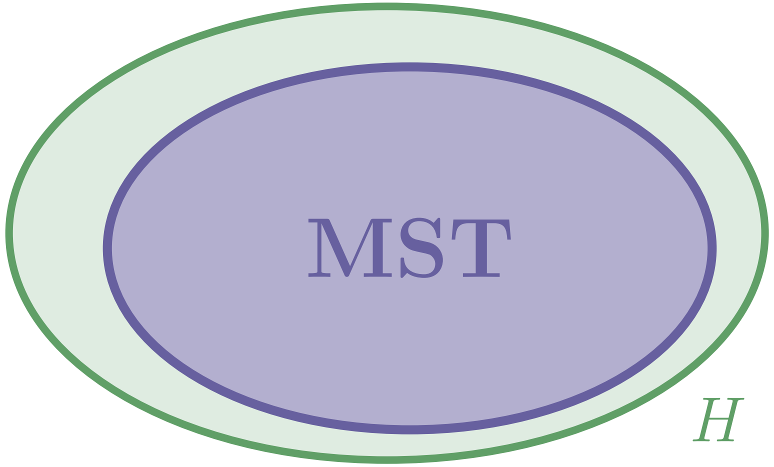
Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

$\rightarrow$ Can we extend this MST so that it keeps "eating" nodes?



Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

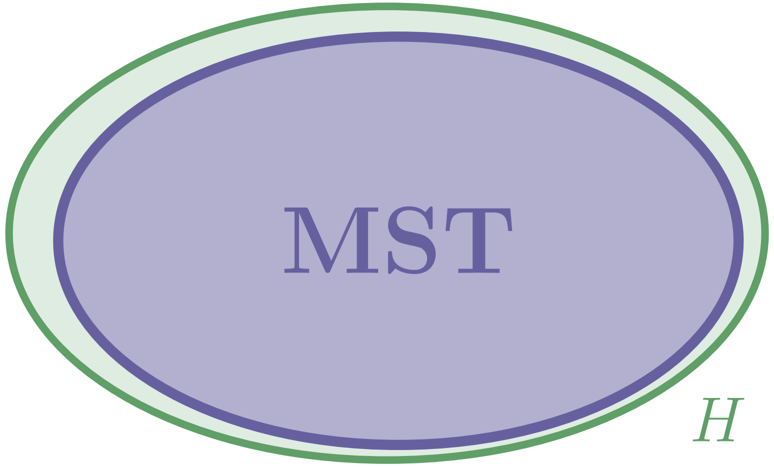
$\rightarrow$ Can we extend this MST so that it keeps "eating" nodes?



The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

$\rightarrow$ Can we extend this MST so that it keeps “eating” nodes?



Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

→ Can we extend this MST so that it keeps “eating” nodes?

The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

→ Can we extend this MST so that it keeps "eating" nodes?

→ If we have a MST on $n-1$ nodes, can we extend it to n nodes?

The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

→ Can we extend this MST so that it keeps "eating" nodes?

→ If we have a MST on $n-1$ nodes, can we extend it to n nodes?

Useful facts about the MST on $\mathbb{K}_n$:

The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

→ Can we extend this MST so that it keeps "eating" nodes?

→ If we have a MST on $n-1$ nodes, can we extend it to n nodes?

Useful facts about the MST on $\mathbb{K}_n$:

- Its total weight is $\zeta(3) + o_{\mathbb{P}}(1)$. (F'85)

The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

$\rightarrow$ Can we extend this MST so that it keeps "eating" nodes?

$\rightarrow$ If we have a MST on $n-1$ nodes, can we extend it to n nodes?

Useful facts about the MST on $\mathbb{K}_n$:

- Its total weight is $\zeta(3) + o_{\mathbb{P}}(1)$. (F'85)
- Its edges have weight $O_{\mathbb{P}}(\log n/n)$. (ABBC'22)

The eating algorithm

Imagine the scenario where we started from H arbitrary and managed to replace a large subgraph by its (local) MST.

$\rightarrow$ Can we extend this MST so that it keeps "eating" nodes?

$\rightarrow$ If we have a MST on $n-1$ nodes, can we extend it to n nodes?

Useful facts about the MST on $\mathbb{K}_n$:

- Its total weight is $\zeta(3) + o_{\mathbb{P}}(1)$. (F'85)
- Its edges have weight $O_{\mathbb{P}}(\log n/n)$. (ABBC'22)
- Its diameter is $\Theta_{\mathbb{P}}(n^{1/3})$. (ABBR'06)

The eating algorithm (easy case)

The eating algorithm (easy case)

Consider the following (easier) scenario.

The eating algorithm (easy case)

Consider the following (easier) scenario.

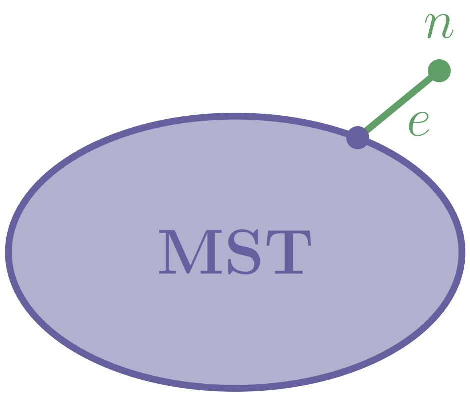
→ We have the MST on $n-1$ nodes.

The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.



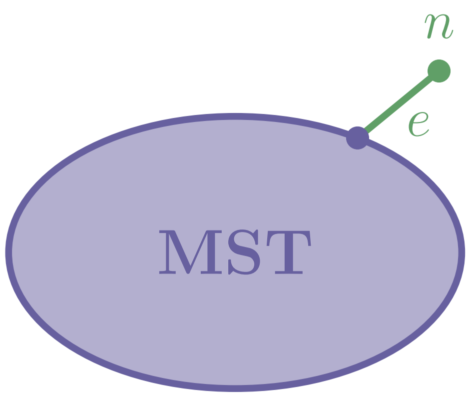
The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n - 1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.



The eating algorithm (easy case)

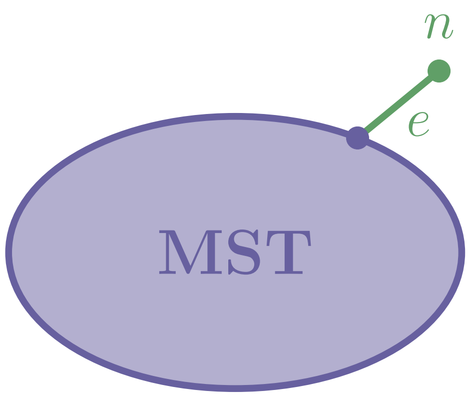
Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.



The eating algorithm (easy case)

Consider the following (easier) scenario.

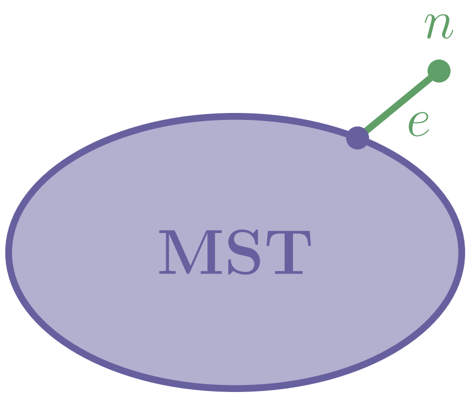
→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

→ We need to do better!



The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

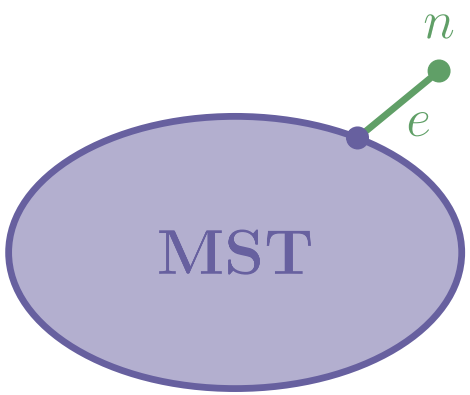
→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

→ We need to do better!

Instead we consider the paths from n to $1, 2, \ldots, n-1$.



The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.

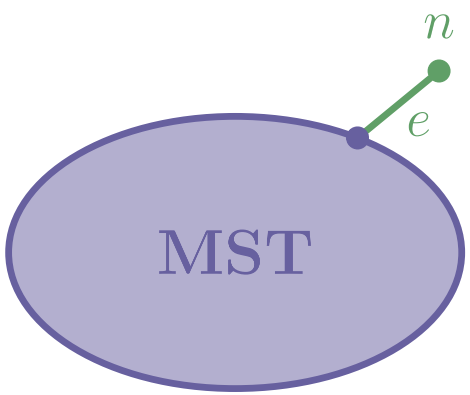
If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

→ We need to do better!

Instead we consider the paths from n to $1, 2, \ldots, n-1$.

→ The weight of a path in the MST is $o_{\mathbb{P}}(1)$.



The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

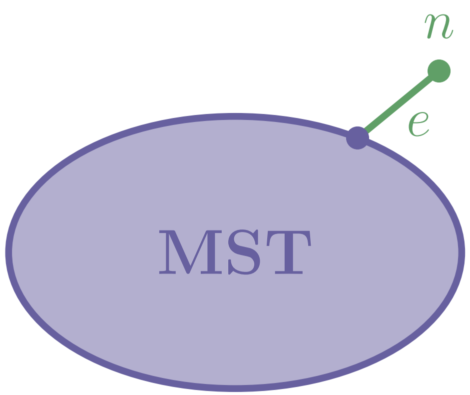
→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

→ We need to do better!

Instead we consider the paths from n to $1, 2, \ldots, n-1$.

→ The weight of a path in the MST is $o_{\mathbb{P}}(1)$.

→ The path from n to i has weight $U_e + o_{\mathbb{P}}(1) \leq 1 + \epsilon$.



The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

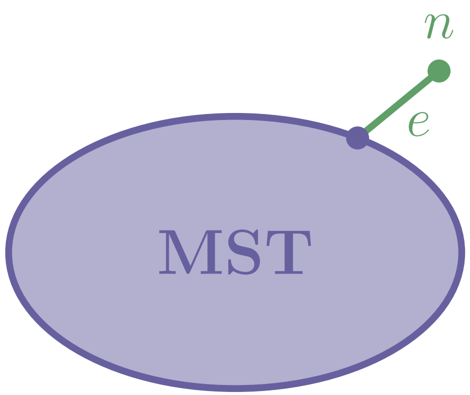
→ We need to do better!

Instead we consider the paths from n to $1, 2, \ldots, n-1$.

→ The weight of a path in the MST is $o_{\mathbb{P}}(1)$.

→ The path from n to i has weight $U_e + o_{\mathbb{P}}(1) \leq 1 + \epsilon$.

→ After considering all such paths, we have the (global) MST.



The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n - 1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

→ We need to do better!

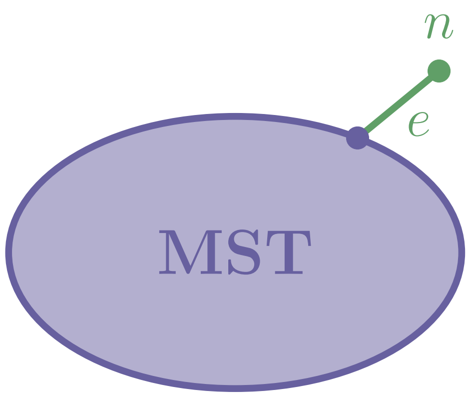
Instead we consider the paths from n to $1, 2, \ldots, n-1$.

→ The weight of a path in the MST is $o_{\mathbb{P}}(1)$.

→ The path from n to i has weight $U_e + o_{\mathbb{P}}(1) \leq 1 + \epsilon$.

→ After considering all such paths, we have the (global) MST.

⚠ Where is the problem?



The eating algorithm (easy case)

Consider the following (easier) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via a single edge e.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $U_e + \zeta(3) \leq 1 + \zeta(3)$.

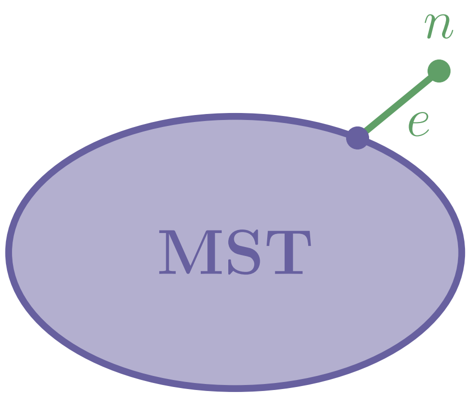
→ We need to do better!

Instead we consider the paths from n to $1, 2, \ldots, n-1$.

→ The weight of a path in the MST is $o_{\mathbb{P}}(1)$.

→ The path from n to i has weight $U_e + o_{\mathbb{P}}(1) \leq 1 + \epsilon$.

→ After considering all such paths, we have the (global) MST.



⚠ Where is the problem? The weight of the paths might change during the process!

The eating algorithm (hard case)

The eating algorithm (hard case)

Consider now the following (harder) scenario.

The eating algorithm (hard case)

Consider now the following (harder) scenario.

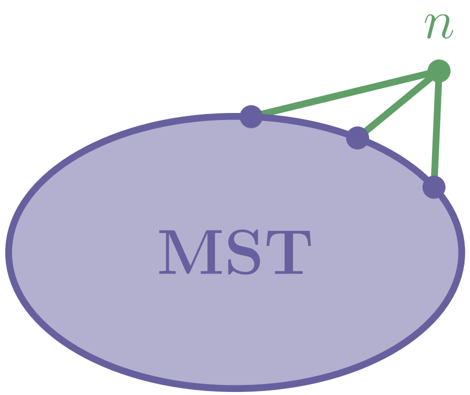
→ We have the MST on $n-1$ nodes.

The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.



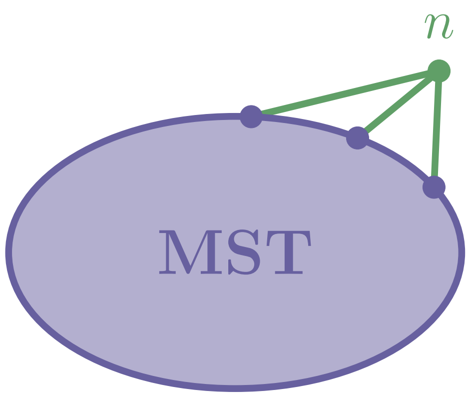
The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.



The eating algorithm (hard case)

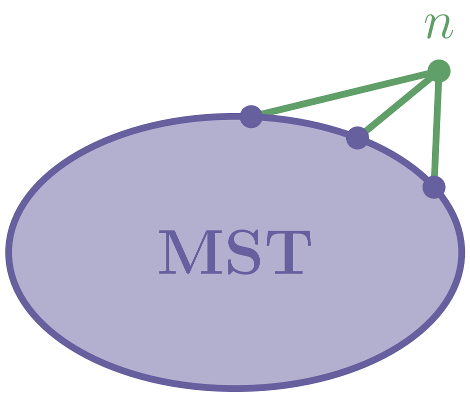
Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $\leq \deg(n) + \zeta(3)$.



The eating algorithm (hard case)

Consider now the following (harder) scenario.

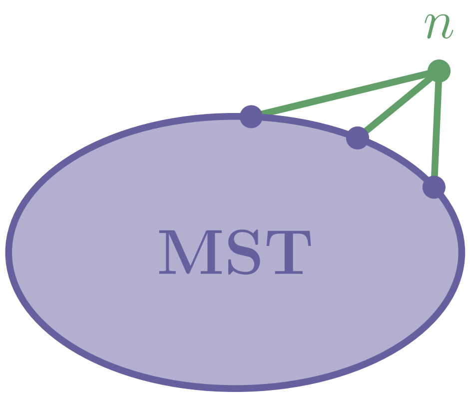
→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $\leq \deg(n) + \zeta(3)$.

→ This is not even bounded anymore!



The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

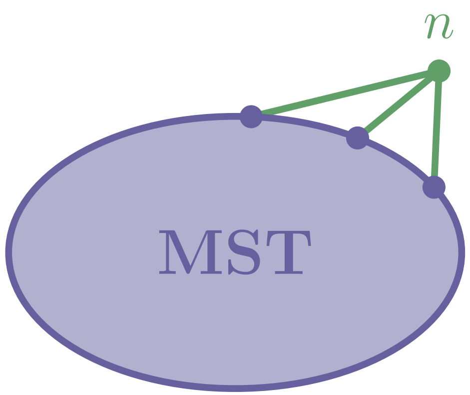
→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $\leq \deg(n) + \zeta(3)$.

→ This is not even bounded anymore!

We consider again the paths from n to $1, 2, \ldots, n-1$.



The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

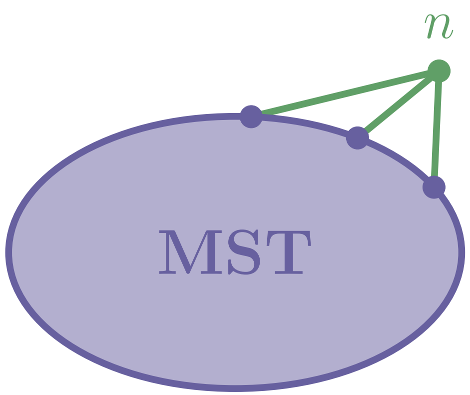
If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $\leq \deg(n) + \zeta(3)$.

→ This is not even bounded anymore!

We consider again the paths from n to $1, 2, \ldots, n-1$.

→ They are not unique, so we need to choose carefully.



The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.

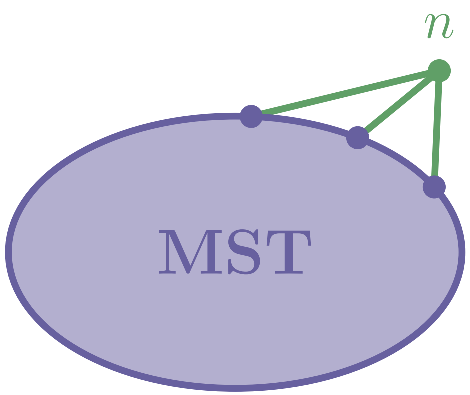
→ The weight of this step is $\leq \deg(n) + \zeta(3)$.

→ This is not even bounded anymore!

We consider again the paths from n to $1, 2, \ldots, n-1$.

→ They are not unique, so we need to choose carefully.

→ After considering the paths, we have a supergraph of the MST.



The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $\leq \deg(n) + \zeta(3)$.

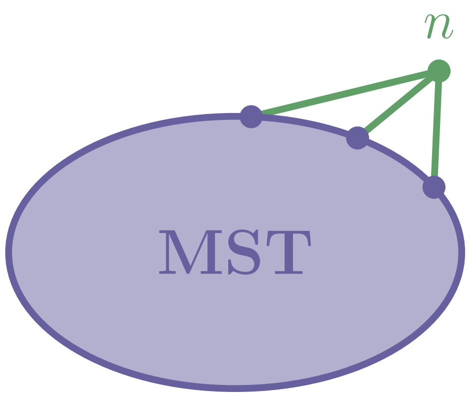
→ This is not even bounded anymore!

We consider again the paths from n to $1, 2, \ldots, n-1$.

→ They are not unique, so we need to choose carefully.

→ After considering the paths, we have a supergraph of the MST.

→ We remove extra edges by considering cycles (carefully again).



The eating algorithm (hard case)

Consider now the following (harder) scenario.

→ We have the MST on $n-1$ nodes.

→ The n-th node is attached to it via multiple edges.

If we consider the set $[n]$, we obtain the (global) MST in one step.

→ The weight of this step is $\leq \deg(n) + \zeta(3)$.

→ This is not even bounded anymore!

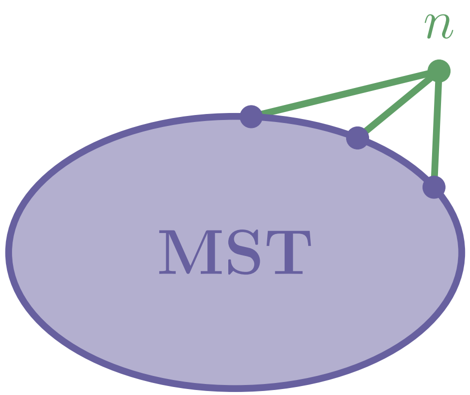
We consider again the paths from n to $1, 2, \ldots, n-1$.

→ They are not unique, so we need to choose carefully.

→ After considering the paths, we have a supergraph of the MST.

→ We remove extra edges by considering cycles (carefully again).

→ Luckily, we obtain steps with weight $1 + o_{\mathbb{P}}(1)$ again.



The three main cases

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities:

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;
- if it has a large diameter, then it contains a long (induced) line.

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;
- if it has a large diameter, then it contains a long (induced) line.
- if it has a large degree, then the neighbours of this high degree either have:

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;
- if it has a large diameter, then it contains a long (induced) line.
- if it has a large degree, then the neighbours of this high degree either have:
    - a large clique, thus creating a large (induced) complete graph, or

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;
- if it has a large diameter, then it contains a long (induced) line.
- if it has a large degree, then the neighbours of this high degree either have:
  - a large clique, thus creating a large (induced) complete graph, or
  - a large independent set, thus creating a large (induced) star.

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;
- if it has a large diameter, then it contains a long (induced) line.
- if it has a large degree, then the neighbours of this high degree either have:
    - a large clique, thus creating a large (induced) complete graph, or
    - a large independent set, thus creating a large (induced) star.

Thus, if we can transform a large complete graph, star, and line into their MST by only changing subgraphs of weight $1 + o_{\mathbb{P}}(1)$, then we can do the same for any graph.

The three main cases

From the eating algorithm, if we manage to replace a large subgraph by its (local) MST, then we can expand it until reaching the (global) MST.

We use this method to reduce H to one of three possibilities: the complete graph, a star, or a line.

- a large graph either has a large degree or a large diameter;
- if it has a large diameter, then it contains a long (induced) line.
- if it has a large degree, then the neighbours of this high degree either have:
  - a large clique, thus creating a large (induced) complete graph, or
  - a large independent set, thus creating a large (induced) star.

Thus, if we can transform a large complete graph, star, and line into their MST by only changing subgraphs of weight $1 + o_{\mathbb{P}}(1)$, then we can do the same for any graph.

⚠ We need to be careful on the dependency with the edge weights $\mathbb{U}$.

Concluding the proof

Concluding the proof

- With the eating algorithm, we can grow MSTs within H (assuming “some” independence).

Concluding the proof

- With the eating algorithm, we can grow MSTs within H (assuming “some” independence).
- This allows us to consider only three different cases: the complete graph, a star, or a line.

Concluding the proof

- With the eating algorithm, we can grow MSTs within H (assuming “some” independence).
- This allows us to consider only three different cases: the complete graph, a star, or a line.

→ To conclude the proof, we construct sequences of sets on those three cases.

Concluding the proof

- With the eating algorithm, we can grow MSTs within H (assuming "some" independence).
- This allows us to consider only three different cases: the complete graph, a star, or a line.

$\to$ To conclude the proof, we construct sequences of sets on those three cases.

  - For the complete graph and the star, it is quite easy, since all nodes are close to each other.

Concluding the proof

- With the eating algorithm, we can grow MSTs within H (assuming "some" independence).
- This allows us to consider only three different cases: the complete graph, a star, or a line.

→ To conclude the proof, we construct sequences of sets on those three cases.

- For the complete graph and the star, it is quite easy, since all nodes are close to each other.
- The line is more complicated, since we have to start with a large subline of small weights, but then the corresponding MST is not independent of the weights.

Concluding the proof

- With the eating algorithm, we can grow MSTs within H (assuming “some” independence).
- This allows us to consider only three different cases: the complete graph, a star, or a line.

→ To conclude the proof, we construct sequences of sets on those three cases.

- For the complete graph and the star, it is quite easy, since all nodes are close to each other.
- The line is more complicated, since we have to start with a large subline of small weights, but then the corresponding MST is not independent of the weights.

→ In that case, the eating algorithm still works, but the proof is more tedious.

Table of contents

Future work

Future work

Some future directions:

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value?

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? ...

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? ...
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? ...
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
  - The p-norm of the one-step changes, instead of the ∞-norm?

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? ...
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
  - The p-norm of the one-step changes, instead of the ∞-norm? ....

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \dots, S_k)$. Now, what can we say about k? What is its minimal possible value? . . .
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
  - The p-norm of the one-step changes, instead of the ∞-norm? . . . .
  - The size of a one-step change, instead of the weight?

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? . . .
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
  - The p-norm of the one-step changes, instead of the ∞-norm? . . . .
  - The size of a one-step change, instead of the weight? . . . .

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? . . .
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
  - The p-norm of the one-step changes, instead of the ∞-norm? . . . .
  - The size of a one-step change, instead of the weight? . . . .
- Our theorem proves the existence of **a specific** optimization with respect to (H_n, λ). What happens now if we consider a **random** sequence $\mathbb{S} = (S_1, \ldots, S_k)$? If we keep generating new subsets for as long as we want, do we eventually reach the minimum spanning tree?

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \dots, S_k)$. Now, what can we say about k? What is its minimal possible value? . . .
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
  - The p-norm of the one-step changes, instead of the ∞-norm? . . . .
  - The size of a one-step change, instead of the weight? . . . .
- Our theorem proves the existence of **a specific** optimization with respect to (H_n, λ). What happens now if we consider a **random** sequence $\mathbb{S} = (S_1, \dots, S_k)$? If we keep generating new subsets for as long as we want, do we eventually reach the minimum spanning tree? . . . . .

Future work

Some future directions:

- Our theorem states that, for H_n and $\lambda > 1$, asymptotically there exists an optimization $\mathbb{S} = (S_1, \ldots, S_k)$. Now, what can we say about k? What is its minimal possible value? . . .
- Our theorem proves the existence of a threshold for a certain cost function (the maximal weight of a one-step change $H_{n,i-1}[S_i]$). Could we consider other weight functions such as
    - The p-norm of the one-step changes, instead of the ∞-norm? . . . .
    - The size of a one-step change, instead of the weight? . . . .
- Our theorem proves the existence of **a specific** optimization with respect to (H_n, λ). What happens now if we consider a **random** sequence $\mathbb{S} = (S_1, \ldots, S_k)$? If we keep generating new subsets for as long as we want, do we eventually reach the minimum spanning tree? . . . . .

$\rightarrow$ Let me focus on the first question, in particular the reason why it is and not .

Open problem

Open problem

From now on, $U_1, \ldots, U_n$ are independent uniforms and $\mathcal{P}_n$ is the set of partitions of $[n]$.

Open problem

From now on, $U_1, \ldots, U_n$ are independent uniforms and $\mathcal{P}_n$ is the set of partitions of $[n]$.
For a partition $(S_1, \ldots, S_k) \in \mathcal{P}_n$, we refer to k as its size and $\max_j \{\sum_{i \in S_j} U_i\}$ as its weight.

Open problem

From now on, $U_1, \ldots, U_n$ are independent uniforms and $\mathcal{P}_n$ is the set of partitions of $[n]$.
For a partition $(S_1, \ldots, S_k) \in \mathcal{P}_n$, we refer to k as its size and $\max_j \{\sum_{i \in S_j} U_i\}$ as its weight.

Pre-question

Open problem

From now on, $U_1, \dots, U_n$ are independent uniforms and $\mathcal{P}_n$ is the set of partitions of $[n]$.
For a partition $(S_1, \dots, S_k) \in \mathcal{P}_n$, we refer to k as its size and $\max_j\{\sum_{i \in S_j} U_i\}$ as its weight.

Pre-question

What is (asymptotically) the minimal size of a partition of weight at most 1?

Open problem

From now on, $U_1, \ldots, U_n$ are independent uniforms and $\mathcal{P}_n$ is the set of partitions of $[n]$.
For a partition $(S_1, \ldots, S_k) \in \mathcal{P}_n$, we refer to k as its size and $\max_j\{\sum_{i \in S_j} U_i\}$ as its weight.

Pre-question

What is (asymptotically) the minimal size of a partition of weight at most 1?

It is actually not too hard to prove that this should be of order $n/2$: we can almost exactly pair the uniforms $U_1, \ldots, U_n$ so that the sum of each pair is less than 1.

Open problem

From now on, $U_1, \ldots, U_n$ are independent uniforms and $\mathcal{P}_n$ is the set of partitions of $[n]$.
For a partition $(S_1, \ldots, S_k) \in \mathcal{P}_n$, we refer to k as its size and $\max_j\{\sum_{i \in S_j} U_i\}$ as its weight.

Pre-question

What is (asymptotically) the minimal size of a partition of weight at most 1?

It is actually not too hard to prove that this should be of order $n/2$: we can almost exactly pair the uniforms $U_1, \ldots, U_n$ so that the sum of each pair is less than 1.

$\rightarrow$ We are now interested in the behaviour of the size when we put more constraints on the partition.

Open problem

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \backslash (S_1 \cup \ldots \cup S_{i-1})$.

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

Open problem

Call *line-connected partition* a partition $(S_1, \dots, S_k)$ where S_i is an interval of $[n] \backslash (S_1 \cup \dots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \dots, U_n$ as aligned on a line.

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

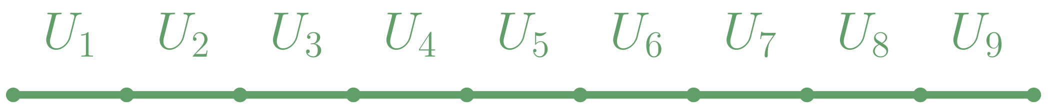
- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

Open problem

Call *line-connected partition* a partition $(S_1, \dots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \dots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \dots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

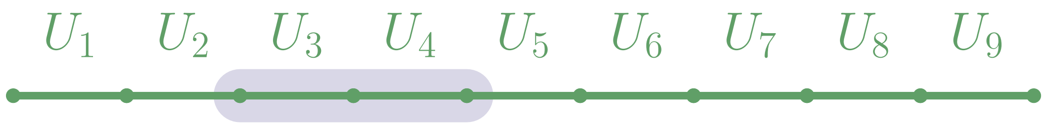


Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



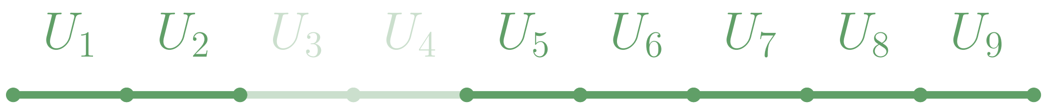
$S_1 = \{3, 4\}$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \backslash (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



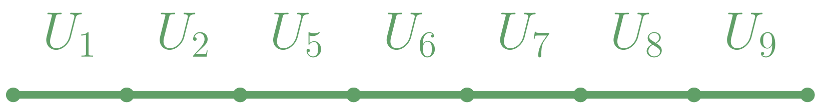
$S_1 = \{3, 4\}$

Open problem

Call *line-connected partition* a partition $(S_1, \dots, S_k)$ where S_i is an interval of $[n] \backslash (S_1 \cup \dots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \dots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



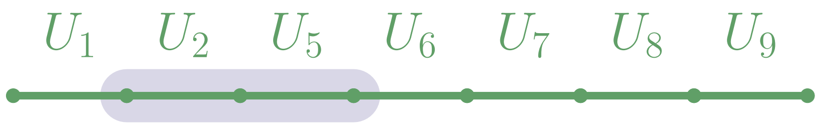
$S_1 = \{3, 4\}$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



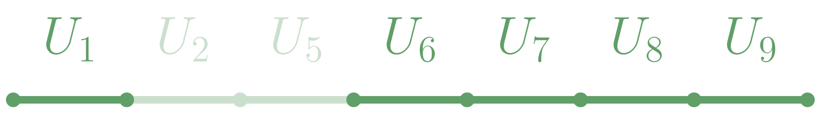
$S_1 = \{3, 4\}$
$S_2 = \{2, 5\}$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



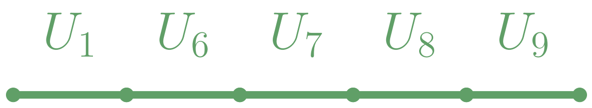
$S_1 = \{3, 4\}$
$S_2 = \{2, 5\}$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



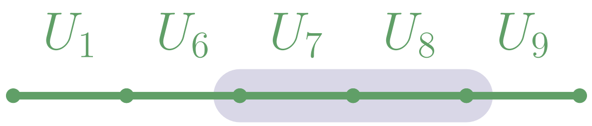
$S_1 = \{3, 4\}$
$S_2 = \{2, 5\}$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



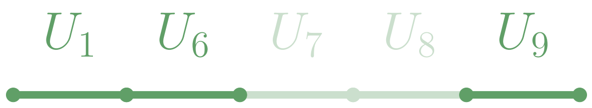
$$S_1 = \{3, 4\}$$
$$S_2 = \{2, 5\}$$
$$S_3 = \{7, 8\}$$

Open problem

Call *line-connected partition* a partition $(S_1, \dots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \dots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \dots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

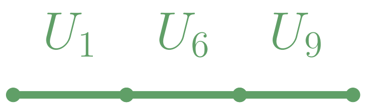


$S_1 = \{3, 4\}$
$S_2 = \{2, 5\}$
$S_3 = \{7, 8\}$

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



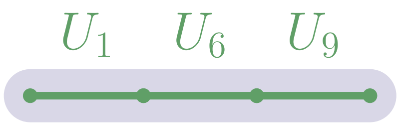
$$S_1 = \{3, 4\}$$
$$S_2 = \{2, 5\}$$
$$S_3 = \{7, 8\}$$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.



$S_1 = \{3, 4\}$
$S_2 = \{2, 5\}$
$S_3 = \{7, 8\}$
$S_4 = \{1, 6, 9\}$

Open problem

Call *line-connected partition* a partition $(S_1, \dots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \dots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \dots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

$$S_1 = \{3, 4\}$$
$$S_2 = \{2, 5\}$$
$$S_3 = \{7, 8\}$$
$$S_4 = \{1, 6, 9\}$$

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \backslash (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

Question

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

Question

What is (asymptotically) the minimal size of a line-connected partition of weight at most 1?

Open problem

Call *line-connected partition* a partition $(S_1, \ldots, S_k)$ where S_i is an interval of $[n] \setminus (S_1 \cup \ldots \cup S_{i-1})$.

A line partition can be constructed as follows.

- See $U_1, \ldots, U_n$ as aligned on a line.
- Remove a segment from this line.
- Reconnect the two ends of the removed segment and repeat the first step.

Question

What is (asymptotically) the minimal size of a line-connected partition of weight at most 1?
In particular, is it approximately $n/2$ as it was the case for general partitions?

Open problem: some progress (or not)

Open problem: some progress (or not)

It is actually rather easy to prove the lower bound. Indeed, for a partition $(S_1, \ldots, S_k)$ of weight at most 1, we have

$$k = \sum_{j=1}^{k} 1 \geq \sum_{j=1}^{k} \sum_{i \in S_j} U_i = \sum_{i \in [n]} U_i \simeq \frac{n}{2}\,.$$

Open problem: some progress (or not)

It is actually rather easy to prove the lower bound. Indeed, for a partition $(S_1, \dots, S_k)$ of weight at most 1, we have

$$k = \sum_{j=1}^{k} 1 \geq \sum_{j=1}^{k} \sum_{i \in S_j} U_i = \sum_{i \in [n]} U_i \simeq \frac{n}{2} \, .$$

Sadly, the upper bound proves to be more difficult to obtain. For example, if considering the special case of interval partitions (which are also themselves line-connected partitions, but easier to study), computations seems to show that the asymptotic size is of order $n/(2-\alpha)$ for some $\alpha > 0$.

Open problem: some progress (or not)

It is actually rather easy to prove the lower bound. Indeed, for a partition $(S_1, \ldots, S_k)$ of weight at most 1, we have

$$k = \sum_{j=1}^{k} 1 \geq \sum_{j=1}^{k} \sum_{i \in S_j} U_i = \sum_{i \in [n]} U_i \simeq \frac{n}{2}\,.$$

Sadly, the upper bound proves to be more difficult to obtain. For example, if considering the special case of interval partitions (which are also themselves line-connected partitions, but easier to study), computations seems to show that the asymptotic size is of order $n/(2-\alpha)$ for some $\alpha > 0$.

$\rightarrow$ I personally tend to believe that the correct behaviour is $n/2$ for line-connected partitions.

Open problem: some progress (or not)

It is actually rather easy to prove the lower bound. Indeed, for a partition $(S_1, \ldots, S_k)$ of weight at most 1, we have

$$k = \sum_{j=1}^{k} 1 \geq \sum_{j=1}^{k} \sum_{i \in S_j} U_i = \sum_{i \in [n]} U_i \simeq \frac{n}{2}\,.$$

Sadly, the upper bound proves to be more difficult to obtain. For example, if considering the special case of interval partitions (which are also themselves line-connected partitions, but easier to study), computations seems to show that the asymptotic size is of order $n/(2-\alpha)$ for some $\alpha > 0$.

→ I personally tend to believe that the correct behaviour is $n/2$ for line-connected partitions.

→ I am biased because this would simplify the general results I want to study.

Open problem: some progress (or not)

It is actually rather easy to prove the lower bound. Indeed, for a partition $(S_1, \ldots, S_k)$ of weight at most 1, we have

$$k = \sum_{j=1}^{k} 1 \geq \sum_{j=1}^{k} \sum_{i \in S_j} U_i = \sum_{i \in [n]} U_i \simeq \frac{n}{2}\,.$$

Sadly, the upper bound proves to be more difficult to obtain. For example, if considering the special case of interval partitions (which are also themselves line-connected partitions, but easier to study), computations seems to show that the asymptotic size is of order $n/(2-\alpha)$ for some $\alpha > 0$.

→ I personally tend to believe that the correct behaviour is $n/2$ for line-connected partitions.

→ I am biased because this would simplify the general results I want to study.

→ A proof that it is not $n/2$ but rather $n/(2-\beta)$ for some $\beta > 0$ is also welcome.

Open problem: some progress (or not)

It is actually rather easy to prove the lower bound. Indeed, for a partition $(S_1, \ldots, S_k)$ of weight at most 1, we have

$$k = \sum_{j=1}^{k} 1 \geq \sum_{j=1}^{k} \sum_{i \in S_j} U_i = \sum_{i \in [n]} U_i \simeq \frac{n}{2} .$$

Sadly, the upper bound proves to be more difficult to obtain. For example, if considering the special case of interval partitions (which are also themselves line-connected partitions, but easier to study), computations seems to show that the asymptotic size is of order $n/(2-\alpha)$ for some $\alpha > 0$.

→ I personally tend to believe that the correct behaviour is $n/2$ for line-connected partitions.

→ I am biased because this would simplify the general results I want to study.

→ A proof that it is not $n/2$ but rather $n/(2-\beta)$ for some $\beta > 0$ is also welcome.

→ It would however lead to further questions...

Open problem: motivation

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

We conjecture that the speed with respect to (H_n, λ) should be of order $w(H_n)/\lambda \simeq |E(H_n)|/2\lambda$ and believe to have the proof when:

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

We conjecture that the speed with respect to (H_n, λ) should be of order $w(H_n)/\lambda \simeq |E(H_n)|/2\lambda$ and believe to have the proof when:

- λ diverges to ∞ with n.

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

We conjecture that the speed with respect to (H_n, λ) should be of order $w(H_n)/\lambda \simeq |E(H_n)|/2\lambda$ and believe to have the proof when:

- λ diverges to ∞ with n.
- H_n has a diverging density: $|E(H_n)|/|V(H_n)| \to \infty$.

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

We conjecture that the speed with respect to (H_n, λ) should be of order $w(H_n)/\lambda \simeq |E(H_n)|/2\lambda$ and believe to have the proof when:

- λ diverges to ∞ with n.
- H_n has a diverging density: $|E(H_n)|/|V(H_n)| \to \infty$.
- H_n is a star.

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

We conjecture that the speed with respect to (H_n, λ) should be of order $w(H_n)/\lambda \simeq |E(H_n)|/2\lambda$ and believe to have the proof when:

- λ diverges to ∞ with n.
- H_n has a diverging density: $|E(H_n)|/|V(H_n)| \to \infty$.
- H_n is a star.

In general, the speed with respect to (H_n, λ) is closely related to the size of a special type of partition built from H_n of weight at most λ, and the case of the line once again proves to be the most difficult one to study...

Open problem: motivation

The previous problem arises when considering the *speed* of an optimization: what is the minimal value of k such that there exists an optimization of k sets with respect to (H_n, λ)?

We conjecture that the speed with respect to (H_n, λ) should be of order $w(H_n)/\lambda \simeq |E(H_n)|/2\lambda$ and believe to have the proof when:

- λ diverges to ∞ with n.
- H_n has a diverging density: $|E(H_n)|/|V(H_n)| \to \infty$.
- H_n is a star.

In general, the speed with respect to (H_n, λ) is closely related to the size of a special type of partition built from H_n of weight at most λ, and the case of the line once again proves to be the most difficult one to study...

To fully solve the "speed problem", we would further need to understand the size of a line-connected partition of weight at most λ, which should not be substantially harder than the case $\lambda = 1$.

Thank you!

Thank you!
Thank you!
Thank you!

References

- Addario-Berry, L., Barrett, J., & Corsini, B. (2022). Finding minimum spanning trees via local improvements. *arXiv preprint arXiv:2205.05075*. (ABBC'22)
- Addario-Berry, L., Broutin, N., & Reed, B. (2006). The diameter of the minimum spanning tree of a complete graph. *Discrete Mathematics & Theoretical Computer Science, (Proceedings)*. (ABBR'06)
- Frieze, A. M. (1985). On the value of a random minimum spanning tree problem. *Discrete Applied Mathematics, 10(1), 47-56*. (F'85)

This work was partially supported by the *Institut des Sciences Mathématiques* (ISM).