# Fast Algorithms for Computing Cactus Representations of Minimum Cuts


*Joint works with **Thatchaphol Saranurak** and **Shang-En Huang**.
(University of Michigan) (Boston College)

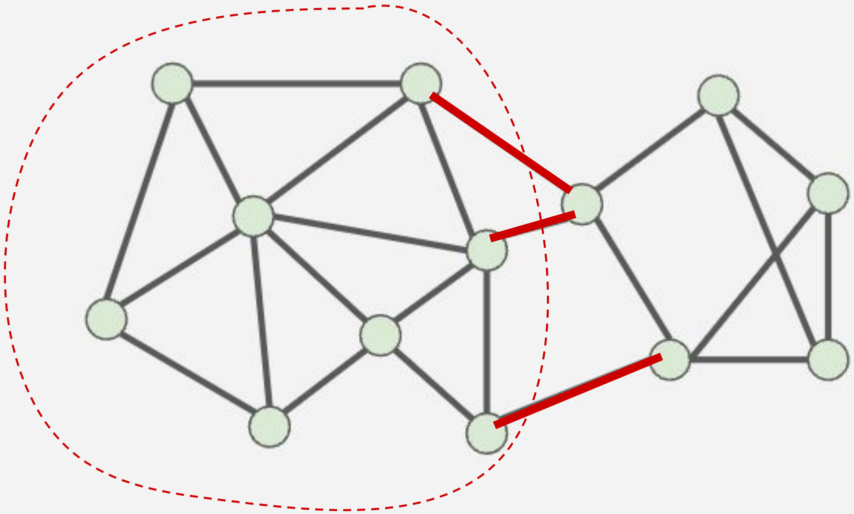
**Zhongtian He**
(Princeton University)


Nov 25, 2024

# Problem: List All Mincuts

Given a **weighted** undirected graph **G = (V, E)**, find all its mincuts.

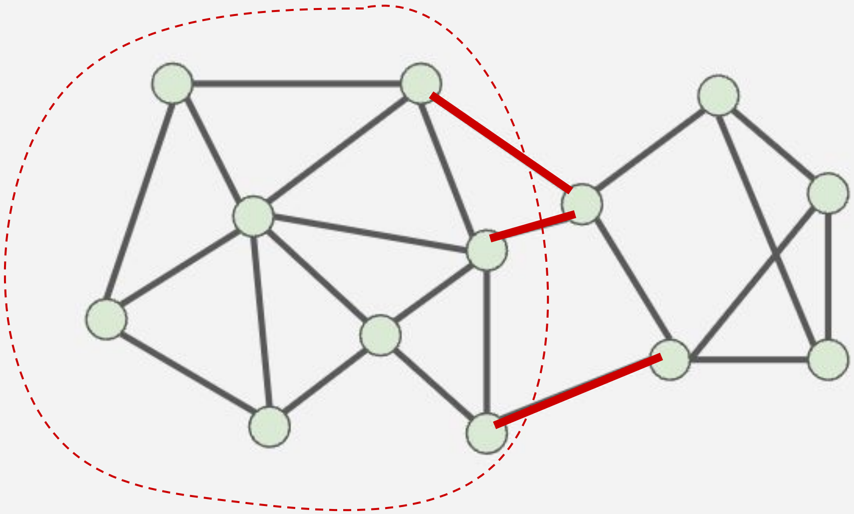
A **mincut** **(A, V\A)** is a partition of **V** such that sum of all edge weights across the cut is minimum.



# Problem: List All Mincuts

Given a **weighted** undirected graph **G = (V, E)**, find all its mincuts.

A **mincut** **(A, V\A)** is a partition of **V** such that sum of all edge weights across the cut is minimum.



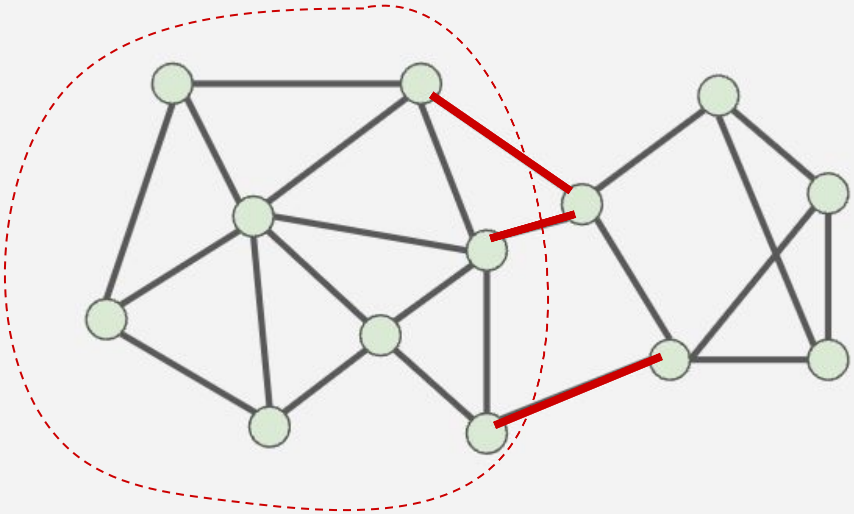
**Theorem** [Dinitz et al. 1976][Karger 1993]

There are $\mathbf{O(n^2)}$ minimum cuts on a graph.

# Problem: List All Mincuts

Given a **weighted** undirected graph **G = (V, E)**, find all its mincuts.

A **mincut** **(A, V\A)** is a partition of **V** such that sum of all edge weights across the cut is minimum.



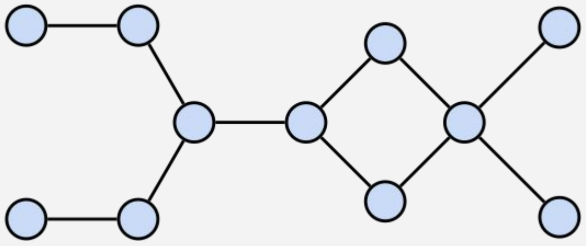
**Theorem** [Dinitz et al. 1976][Karger 1993]

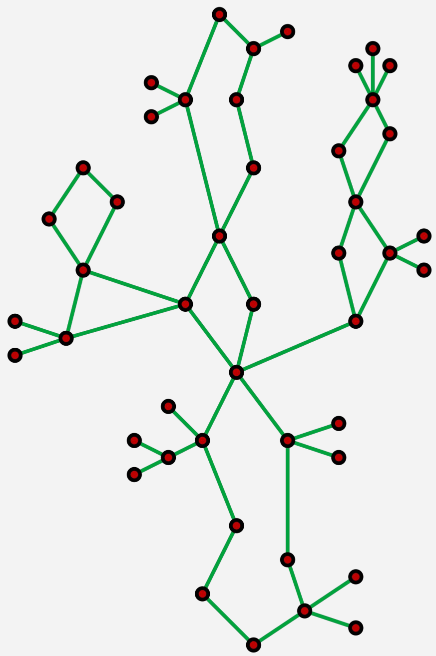
There are $\mathbf{O(n^2)}$ minimum cuts on a graph.

Listing all mincuts requires $\mathbf{\Omega(n^3)}$ time/space.

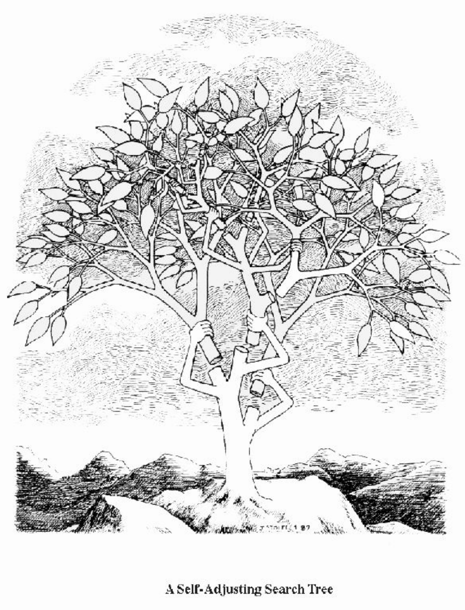
# Cactus Graph

A **cactus** is a graph where every edge belongs to at most one (simple) cycle.





# Cactus Graph




**An ancient tree of hidden power**

*Figure from R.E. Tarjan's Homepage*

**Tree** is fundamental and simple concept in computer science.

How about **Cactus**?

**A cute and innocent cactus**

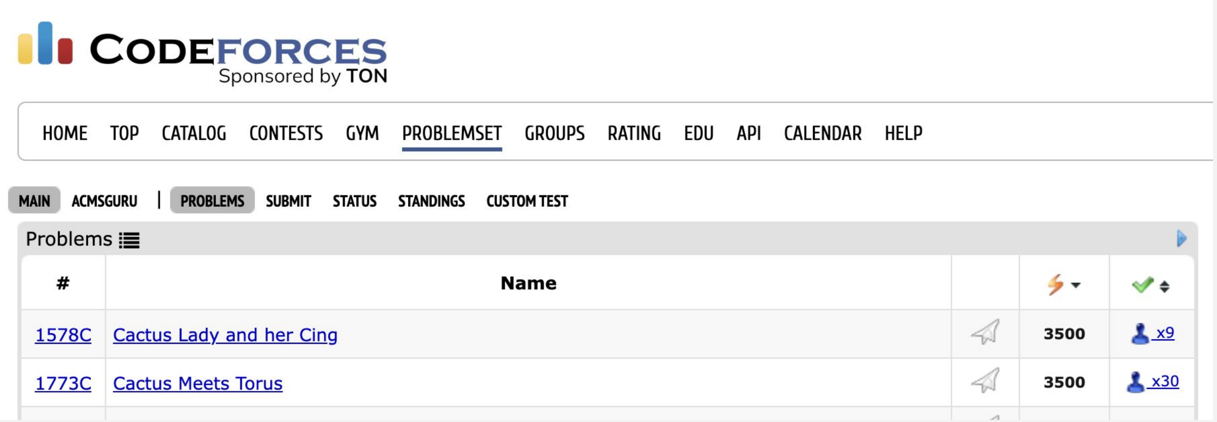
# Cactus in the Theory World

**Algorithms**: Problems NP-hard for general graphs, polynomial time for cacti.

A cute and innocent cactus

# More on Cactus, in the Algorithm World

In competitive programming (ICPC/OI),
cactus problems are famous for its intricacy.



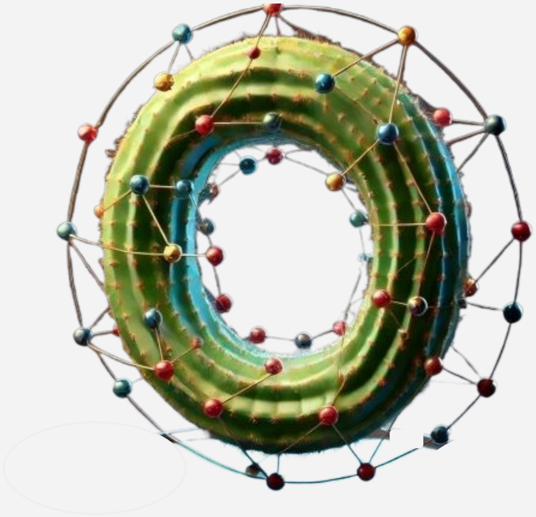

| # | Name | | | |
|---|---|---|---|---|
| 1578C | Cactus Lady and her Cing | | 3500 | x9 |
| 1773C | Cactus Meets Torus | | 3500 | x30 |

A cute and innocent cactus

# Cactus in the Theory World

**Algorithms**: Problems NP-hard for general graphs, polynomial time for cacti.

**Topological Graph Theory**: Graphs with maximum genus 0 are a subfamily of cacti.



**Cactus has no cellular embedding on torus**

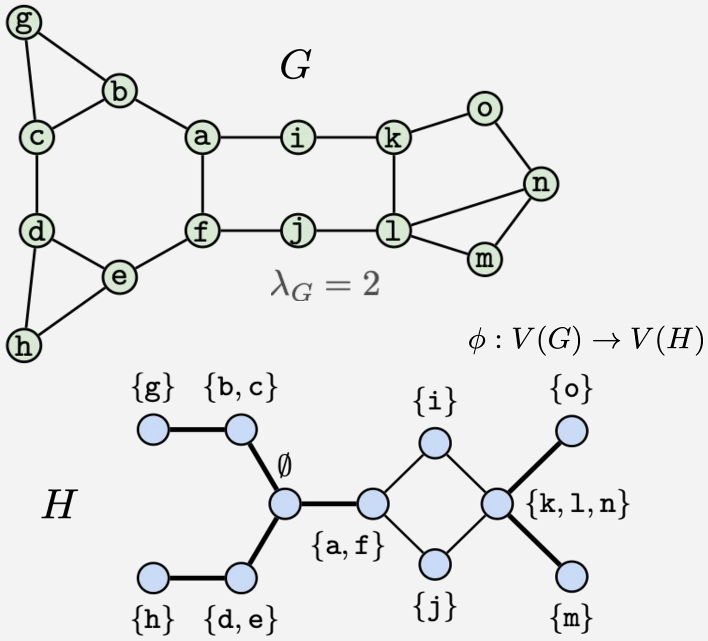
Images generated by ChatGPT 4o

# Cactus in the Theory World

**Algorithms**: Problems NP-hard for general graphs, polynomial time for cacti.

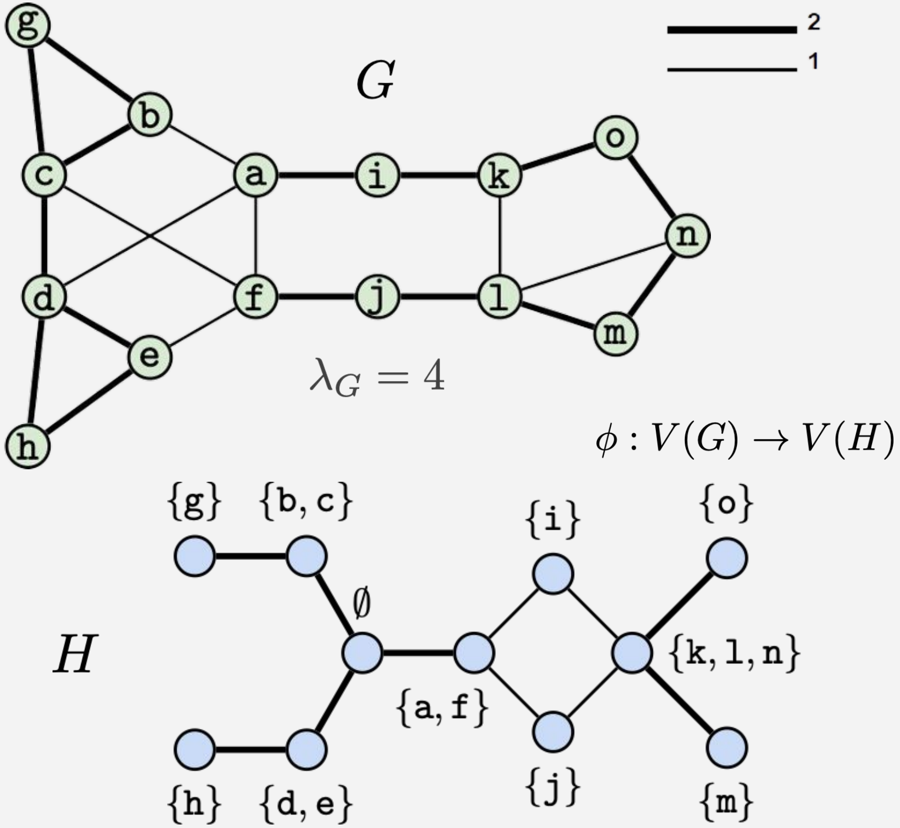
**Topological Graph Theory**: Graphs with maximum genus 0 are a subfamily of cacti.

**Combinatorics (Cactus Representation)**: An edge sparsifier of **O(*n*)** size that exactly captures *all* global mincuts of the graph.




This talk: Compute Cactus Representation efficiently.

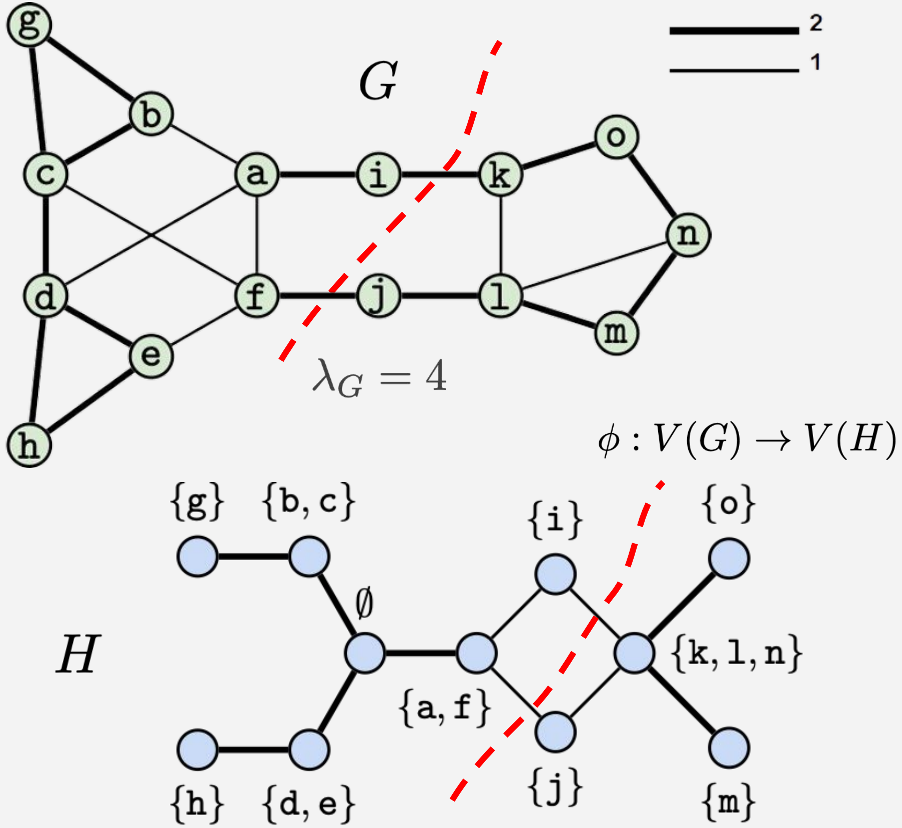
# All Mincuts: Cactus Representations




A **cactus** is a graph where every edge belongs to at most one (simple) cycle.

> **Theorem** [Dinitz et al. 1976]
>
> There exists an $\mathbf{O(n)}$ sized **cactus graph** that preserves *all* mincuts of the given graph.

# All Mincuts: Cactus Representations

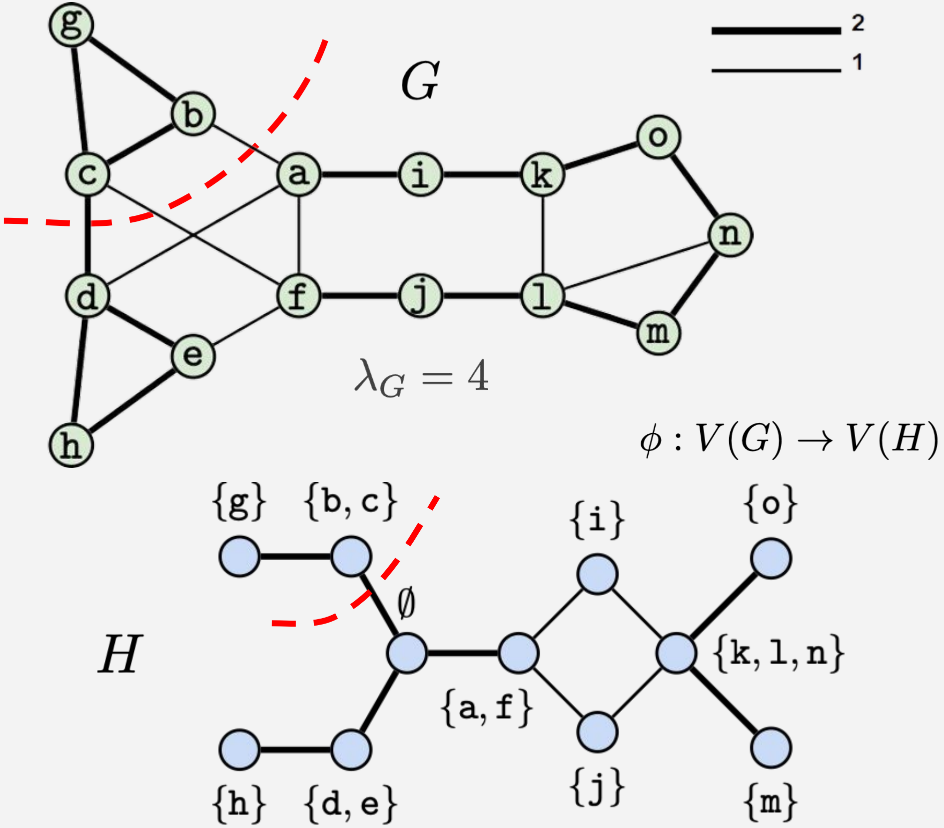



A **cactus** is a graph where every edge belongs to at most one (simple) cycle.

**Theorem** [Dinitz et al. 1976]

There exists an $\mathbf{O}(n)$ sized **cactus graph** that preserves *all* mincuts of the given graph.

# All Mincuts: Cactus Representations

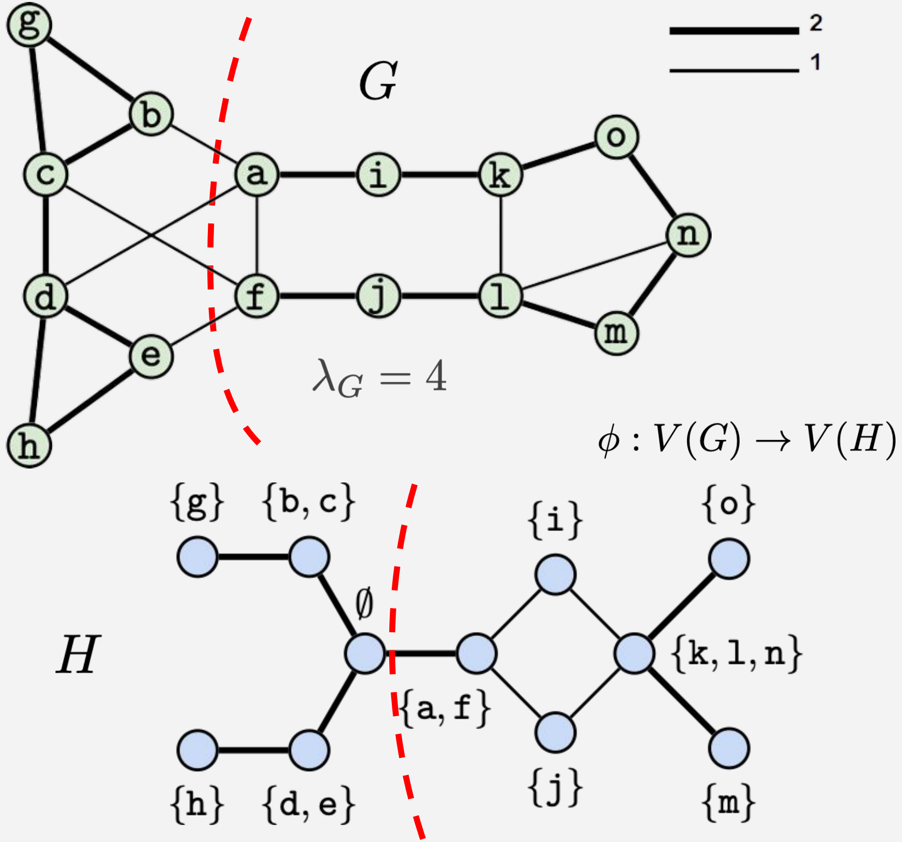



A **cactus** is a graph where every edge belongs to at most one (simple) cycle.

**Theorem** [Dinitz et al. 1976]

There exists an $\mathbf{O(n)}$ sized **cactus graph** that preserves *all* mincuts of the given graph.

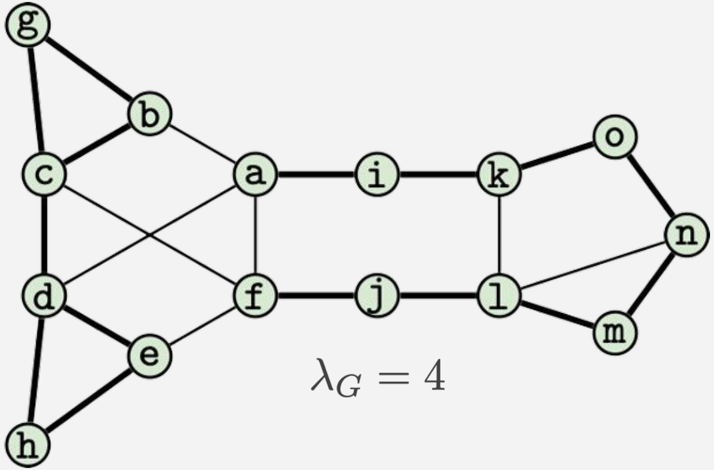
# All Mincuts: Cactus Representations

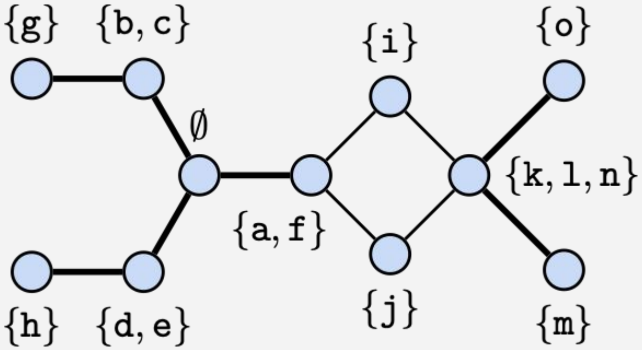



A **cactus** is a graph where every edge belongs to at most one (simple) cycle.

> **Theorem** [Dinitz et al. 1976]
>
> There exists an **O($n$)** sized **cactus graph** that preserves *all* mincuts of the given graph.

# All Mincuts: Cactus Representations




A **cactus** is a graph where every edge belongs to at most one (simple) cycle.

[Dinitz et al. 1976]
There exists an **O(*n*)** sized **cactus graph** that preserves *all* mincuts of the given graph.

**Randomized** Algorithms:

$O(m \log^4 n) \longrightarrow O(m \log^3 n)$

[Karger & Panigrahi 2009] [**He**, Huang, Saranurak 2024]

**Deterministic** Algorithms:

$O(m\, \mathrm{polylog}(n))$

[**He**, Huang, Saranurak 2024] + [Henzinger, Li, Rao, Wang 2024]

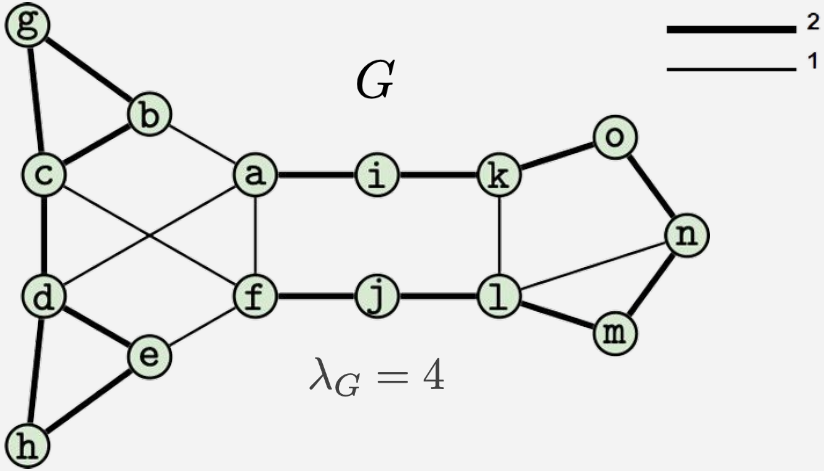
# Outline

1. Tree Packing
2. Minimal Mincuts and Cactus Construction
3. Karger's 2-Respecting Mincuts Algorithm
4. Compute 2-Respecting Minimal Mincuts

# Outline

1. **Tree Packing**
2. Minimal Mincuts and Cactus Construction
3. Karger's 2-Respecting Mincuts Algorithm
4. Compute 2-Respecting Minimal Mincuts

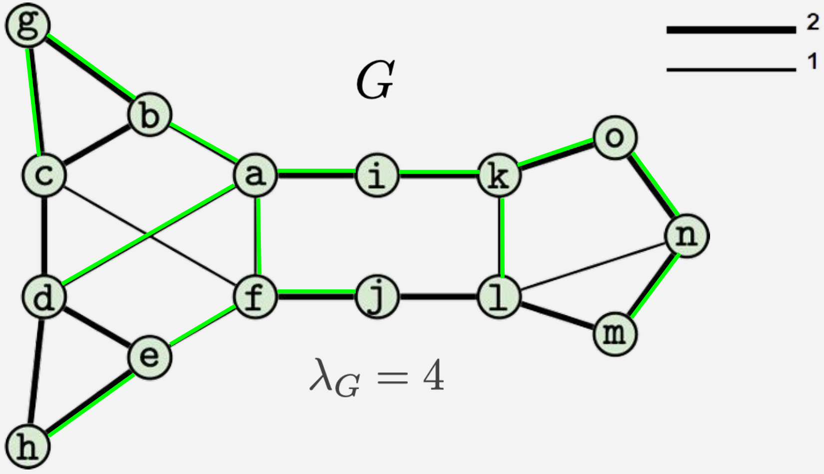
# Tree Packing




**Definition.**

A **tree packing** is a set of (weighted) spanning trees, s.t. the total weight of trees containing edge $e$ is no greater than $w_G(e)$. The **value** of the packing is the total weight of the trees.

# Tree Packing



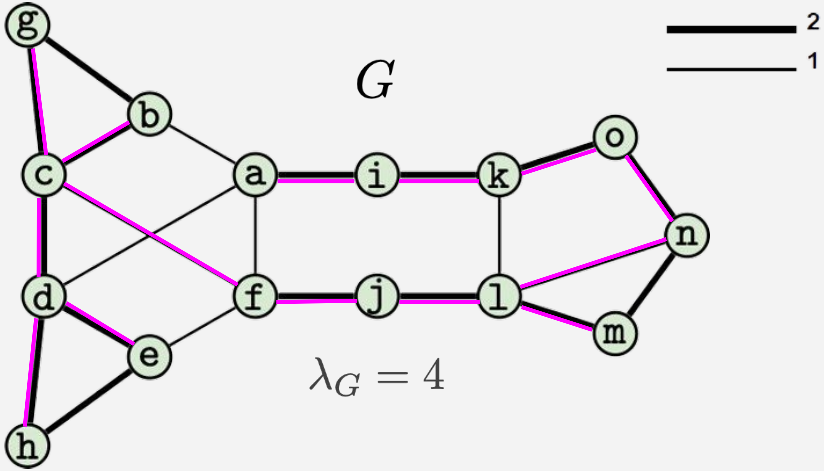

$G$

$\lambda_G = 4$

**Definition.**

A **tree packing** is a set of (weighted) spanning trees, s.t. the total weight of trees containing edge $e$ is no greater than $w_G(e)$. The **value** of the packing is the total weight of the trees.

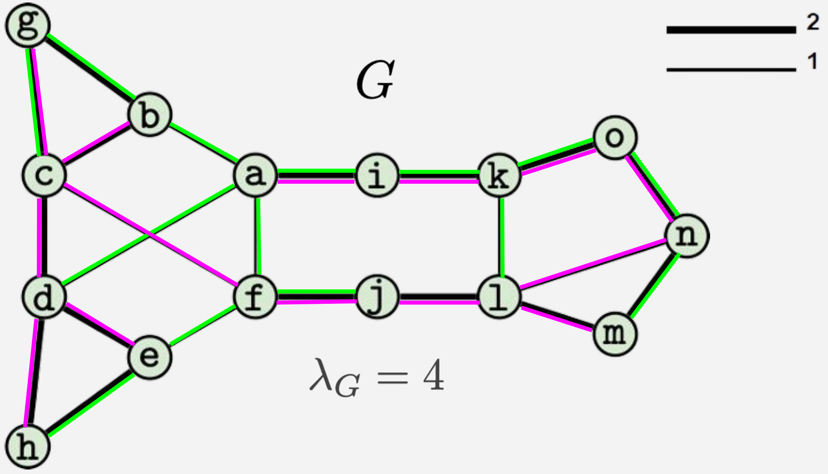
# Tree Packing




**Definition.**

A **tree packing** is a set of (weighted) spanning trees, s.t. the total weight of trees containing edge $e$ is no greater than $w_G(e)$. The **value** of the packing is the total weight of the trees.

# Tree Packing

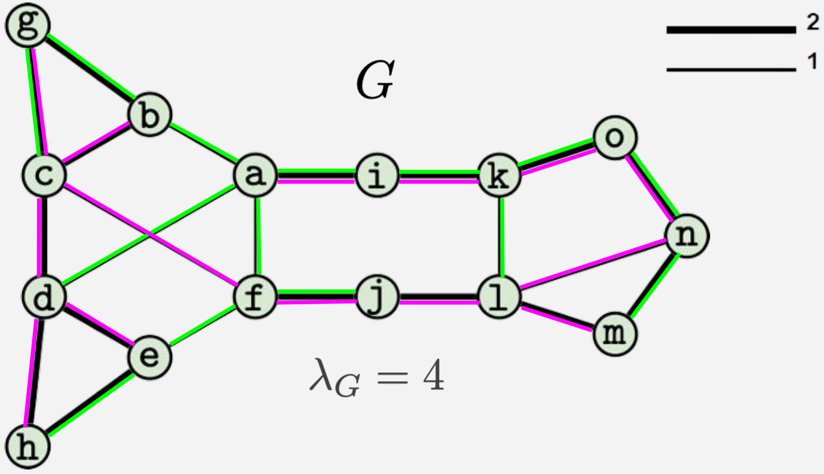



**Definition.**

A **tree packing** is a set of (weighted) spanning trees, s.t. the total weight of trees containing edge $e$ is no greater than $w_G(e)$. The **value** of the packing is the total weight of the trees.

G contains a tree packing of value 2.

# Tree Packing




G contains a tree packing of value 2.

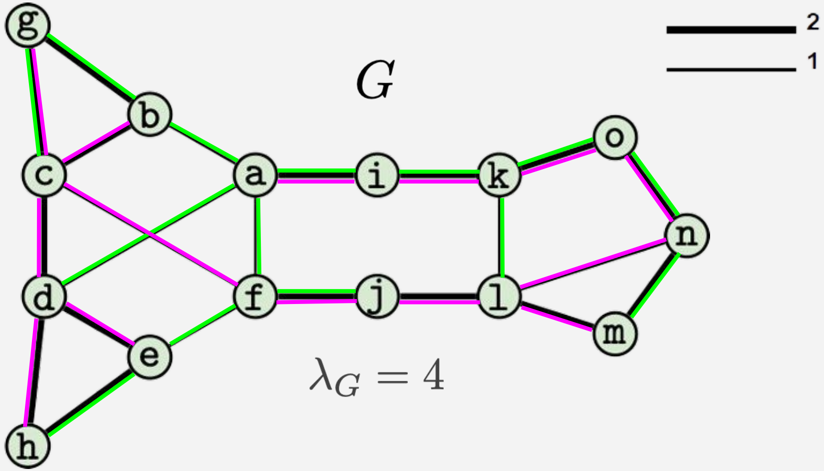
**Definition.**

A **tree packing** is a set of (weighted) spanning trees, s.t. the total weight of trees containing edge $e$ is no greater than $w_G(e)$. The **value** of the packing is the total weight of the trees.

**Theorem** [Nash-Williams 1961]

Any undirected graph with minimum cut $c$ contains a tree packing of value at least $c/2$.

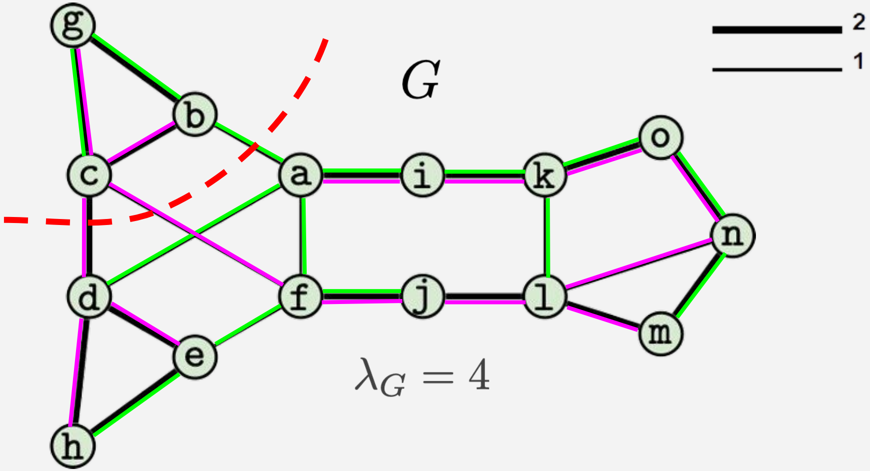
# Tree Packing, 2-Respecting Mincut




**Definition.**

A cut is said to ***k*-respect** a spanning tree if the spanning tree contains at most $k$ edges of the cut.

# Tree Packing, 2-Respecting Mincut

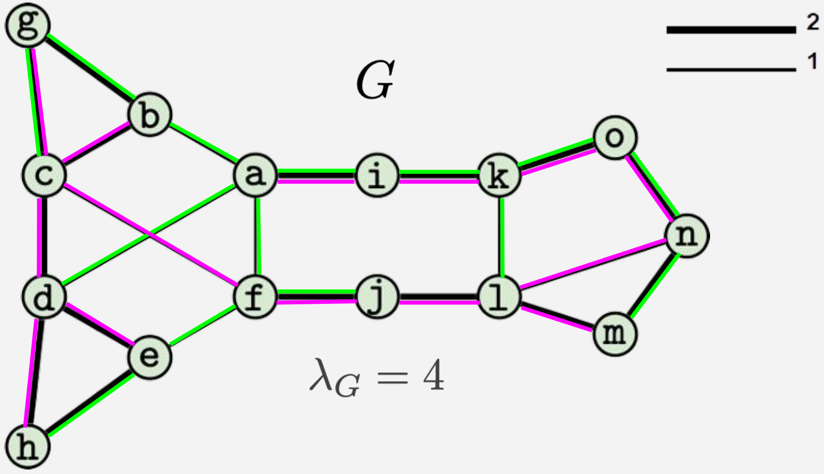



The cut 1-respects the green tree and 2-respects the pink tree.

**Definition.**

A cut is said to ***k*-respect** a spanning tree if the spanning tree contains at most $k$ edges of the cut.

# Tree Packing, 2-Respecting Mincut



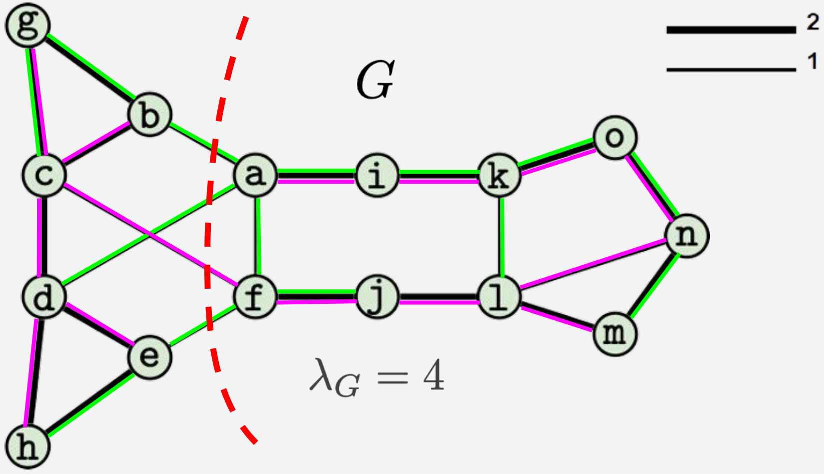

**Theorem** [Nash-Williams 1961]

Any undirected graph with minimum cut $c$ contains a tree packing of value at least $c/2$.

**Lemma.**

In a tree packing of value at least $c/2$, for any mincut, half the trees (by weight) cross the mincut at most twice.

# Tree Packing, 2-Respecting Mincut




The cut 1-respects the pink tree and 3-respects the pink tree.

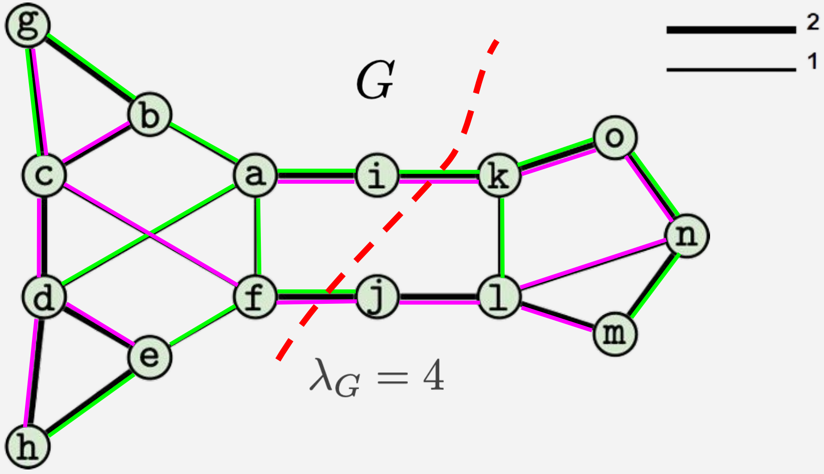
**Theorem** [Nash-Williams 1961]

Any undirected graph with minimum cut $c$ contains a tree packing of value at least $c/2$.

**Lemma.**

In a tree packing of value at least $c/2$, for any mincut, half the trees (by weight) cross the mincut at most twice.

# Tree Packing, 2-Respecting Mincut




The cut 2-respects both the green tree and the pink tree.

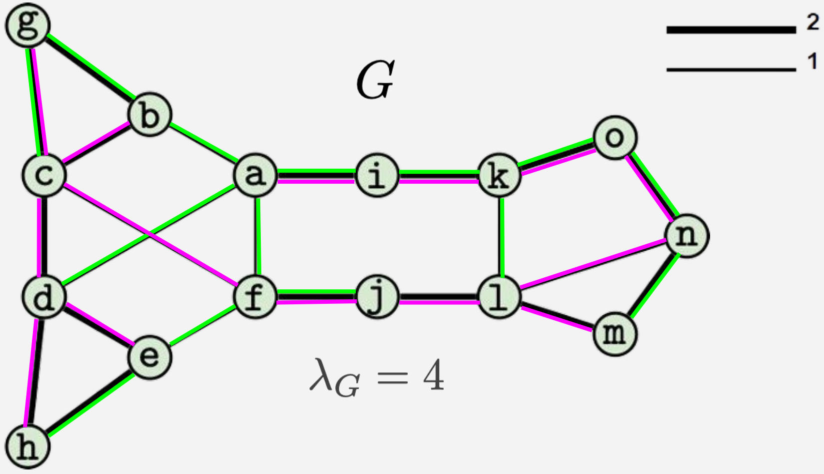
**Theorem** [Nash-Williams 1961]

Any undirected graph with minimum cut $c$ contains a tree packing of value at least $c/2$.

**Lemma.**

In a tree packing of value at least $c/2$, for any mincut, half the trees (by weight) cross the mincut at most twice.

# Tree Packing, 2-Respecting Mincut




**Theorem** [Nash-Williams 1961]

Any undirected graph with minimum cut $c$ contains a tree packing of value at least $c/2$.

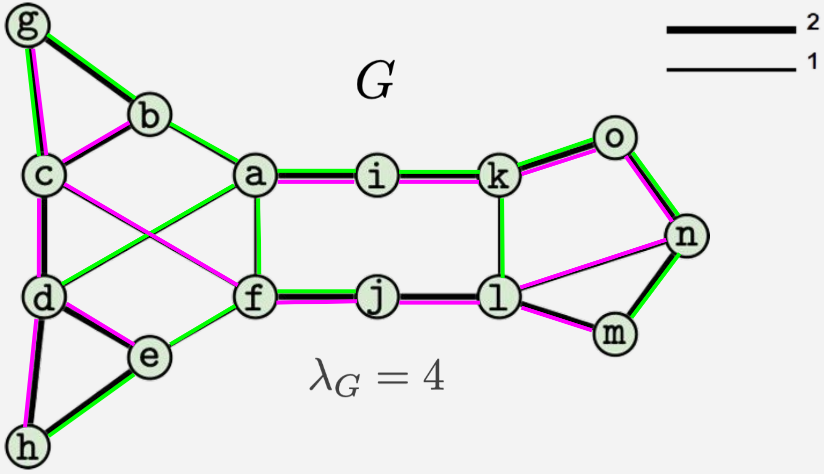
**Lemma.**

In a tree packing of value at least $c/2$, for any mincut, half the trees (by weight) cross the mincut at most twice.

**Theorem** [Karger 1998]

In near-linear time we can construct a set of $O(\log n)$ spanning trees such that each minimum cut 2-respects *1/3* of them w.h.p.

# Tree Packing, 2-Respecting Mincut




**Theorem** [HLRW 2024]

In near-linear time we can construct a set of $(\log n)^{O(1)}$ spanning trees such that each minimum cut 2-respects *1/3* of them.

**Theorem** [Nash-Williams 1961]

Any undirected graph with minimum cut $c$ contains a tree packing of value at least $c/2$.

**Lemma.**

In a tree packing of value at least $c/2$, for any mincut, half the trees (by weight) cross the mincut at most twice.

# Outline

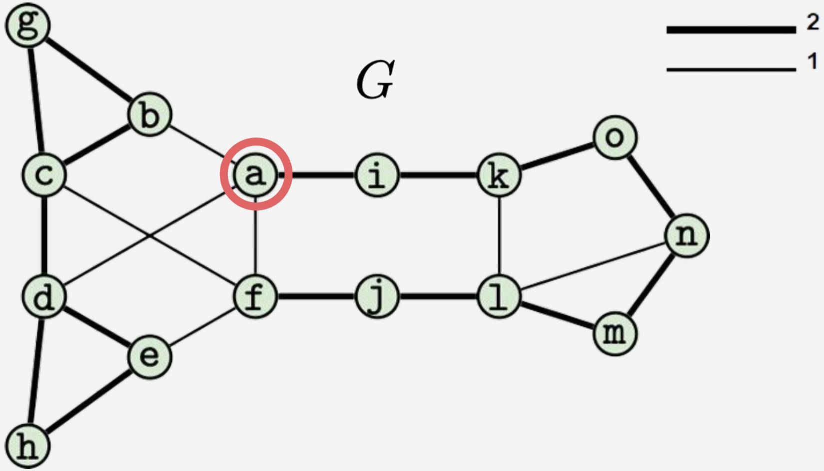
1 Tree Packing

2 **Minimal Mincuts and Cactus Construction**

3 Karger's 2-Respecting Mincuts Algorithm

4 Compute 2-Respecting Minimal Mincuts

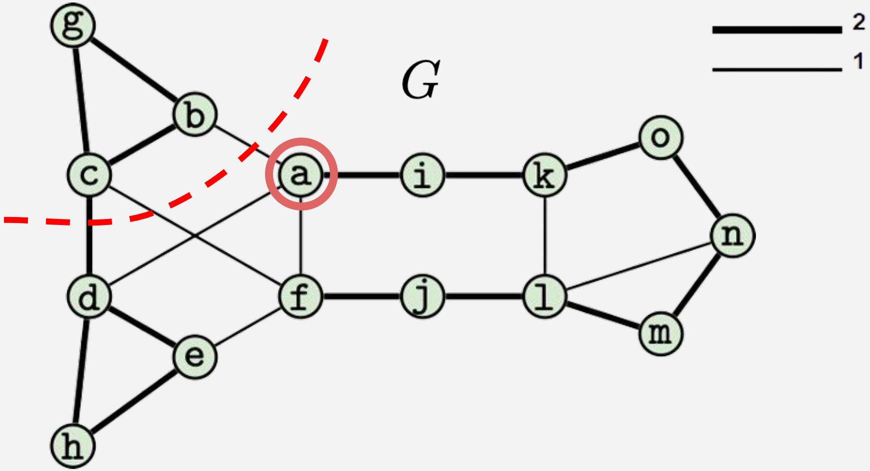
# Minimal Mincuts




We designate an arbitrary but fixed root vertex $r$.

Let $r$ be the vertex $a$.

# Minimal Mincuts



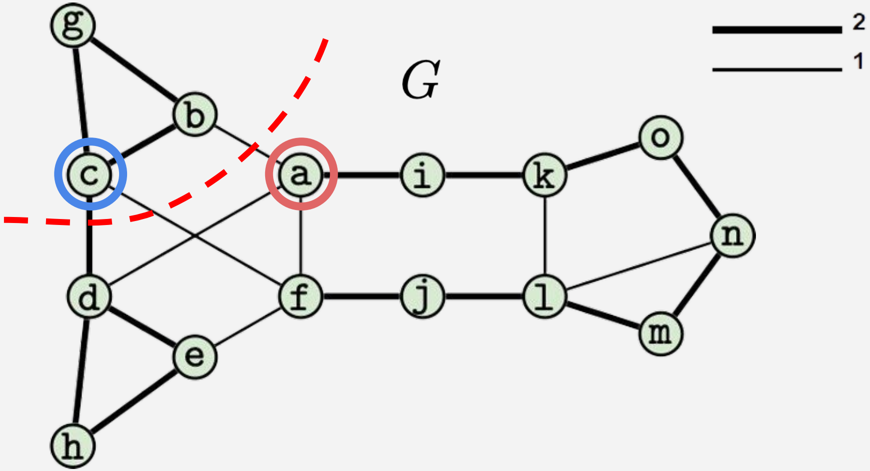

The size of this cut is 3.

We designate an arbitrary but fixed root vertex $r$.

**Definition.**

The **size** of a cut $(X, V \setminus X)$ where $r \notin X$ is then defined to be the number of vertices in $X$.

# Minimal Mincuts




The minimal mincut for vertex $c$.

We designate an arbitrary but fixed root vertex $r$.

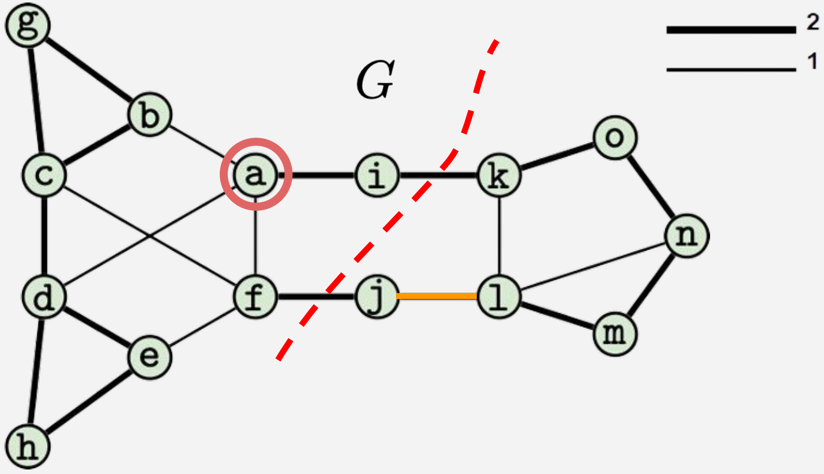
**Definition.**

The **size** of a cut $(X, V \setminus X)$ where $r \notin X$ is then defined to be the number of vertices in $X$.

**Definition**

The **minimal mincut** for a vertex $v$ is the mincut of the least size separating $v$ from $r$.

# Minimal Mincuts




The minimal mincut for edge $(j, l)$.

We designate an arbitrary but fixed root vertex $r$.

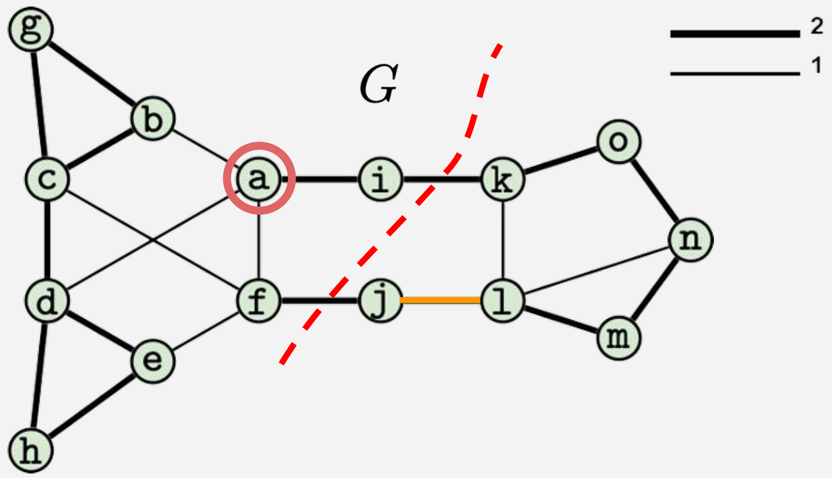
**Definition.**

The **size** of a cut $(X, V \setminus X)$ where $r \notin X$ is then defined to be the number of vertices in $X$.

**Definition**

The **minimal mincut** for an edge $e$ is the mincut of the least size separating $e$ from $r$.

# Uniqueness of Minimal Mincuts




We designate an arbitrary but fixed root vertex $r$.

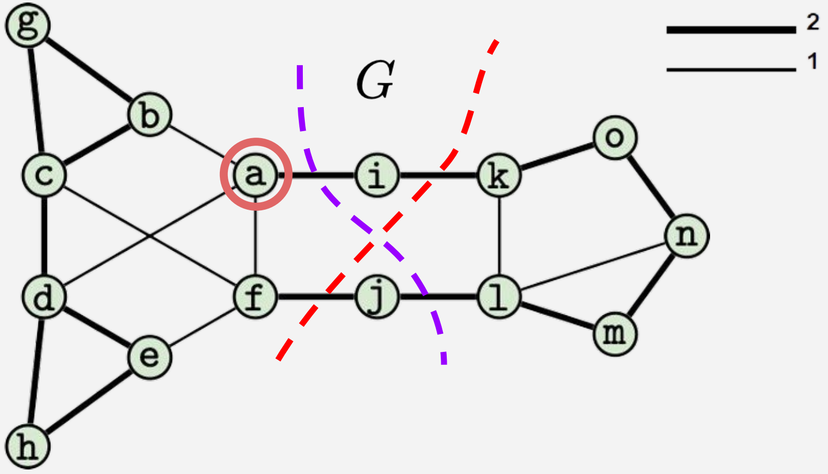
**Lemma.**

If a minimal mincut for a vertex or edge exists, then it is unique.

**Definition**

The **minimal mincut** for a vertex $v$ (resp. edge $e$) is the mincut of the least size separating $v$ (resp. $e$) from $r$.

# Uniqueness of Minimal Mincuts




We designate an arbitrary but fixed root vertex $r$.

**Lemma.**

If a minimal mincut for a vertex or edge exists, then it is unique.

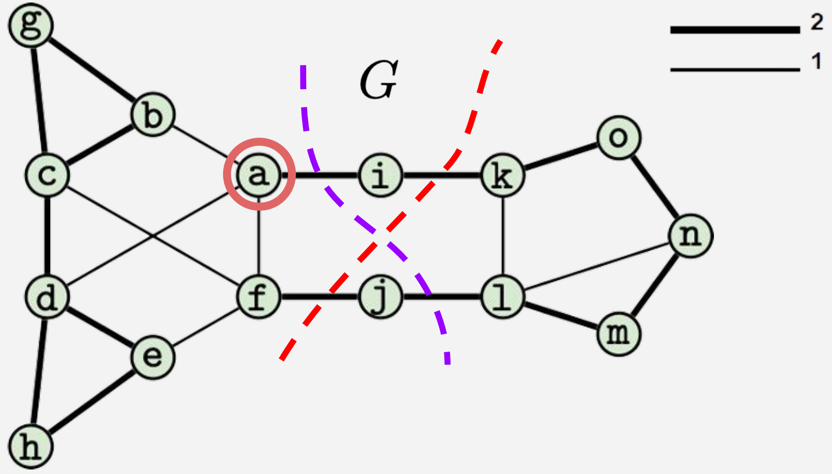
**Definition**

Two cuts $X$ and $Y$ are **crossing** if each of $X \cap Y, X \setminus Y, Y \setminus X, \overline{X} \cap \overline{Y}$ is non-empty.

**Definition**

The **minimal mincut** for a vertex $v$ (resp. edge $e$) is the mincut of the least size separating $v$ (resp. $e$) from $r$.

# Uniqueness of Minimal Mincuts




We designate an arbitrary but fixed root vertex $r$.

**Lemma.**

If a minimal mincut for a vertex or edge exists, then it is unique.

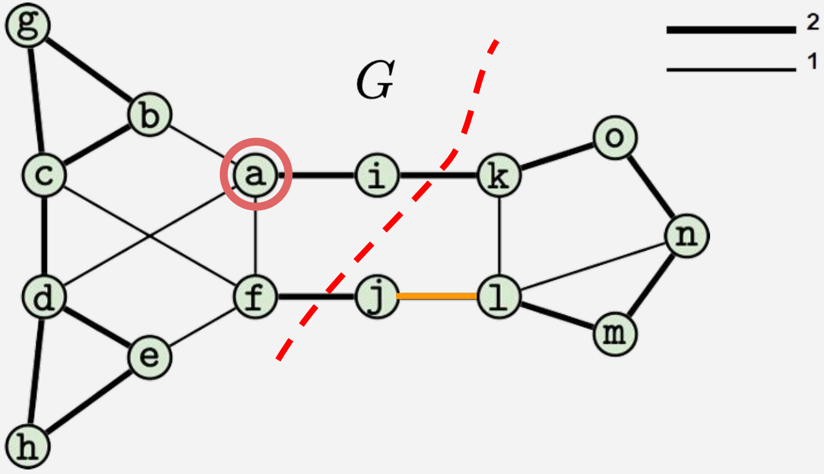
**Lemma**

If $X$ and $Y$ are crossing mincuts, then each of $X \cap Y, X \setminus Y, Y \setminus X, X \cup Y$ is also mincut.

**Definition**

The **minimal mincut** for a vertex $v$ (resp. edge $e$) is the mincut of the least size separating $v$ (resp. $e$) from $r$.

# Uniqueness of Minimal Mincuts




The minimal mincut for edge $(j, l)$.

We designate an arbitrary but fixed root vertex $r$.

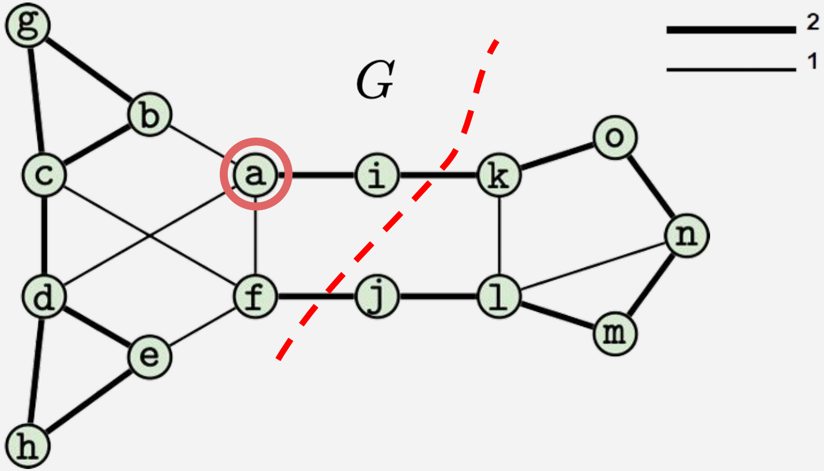
**Lemma.**

If a minimal mincut for a vertex or edge exists, then it is unique.

**Definition**

The **minimal mincut** for a vertex $v$ (resp. edge $e$) is the mincut of the least size separating $v$ (resp. $e$) from $r$.

# Minimal Mincuts and Certificates



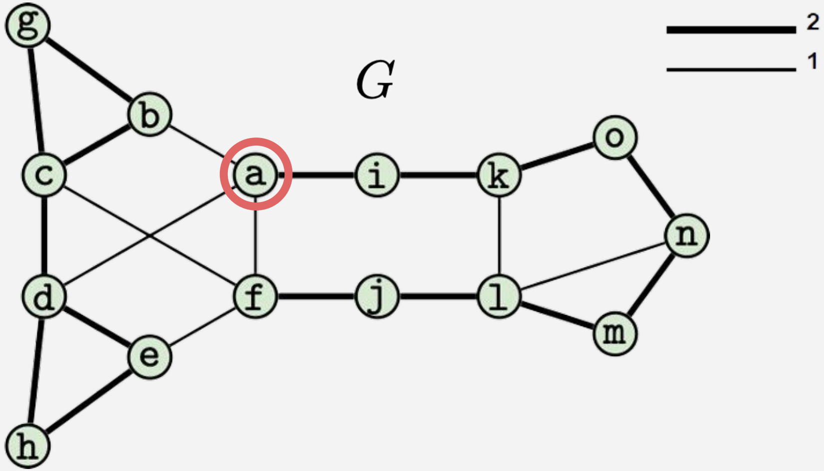

**Definition.**

We say a cut $(X, V \setminus X)$ has a **vertex certificate** (resp. **edge certificate**) if it is a minimal mincut for some vertex $v$ (resp. edge $e$).

**Definition**

The **minimal mincut** for a vertex $v$ (resp. edge $e$) is the mincut of the least size separating $v$ (resp. $e$) from $r$.

# Minimal Mincuts and Certificates




**Definition.**

We say a cut $X$ has a **vertex certificate** (resp. **edge certificate**) if it is a minimal mincut for some vertex $v$ (resp. edge $e$).

A cut $(X, V \setminus X)$ will be simply denoted by $X$.

**Definition**

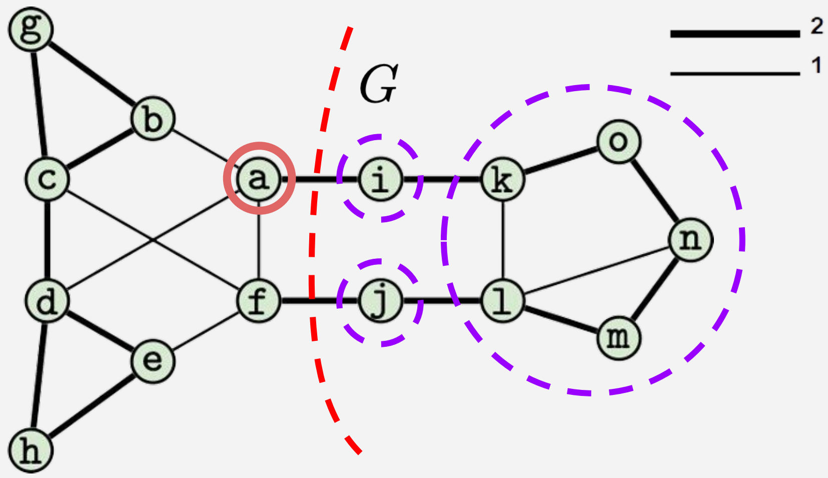
A **chain certificate** is a sequence of disjoint non-empty vertex subsets $(C_0, C_1, \ldots, C_\ell)$ where $\ell \geq 1$, and recursively:

1. For each $i, C_i$ has either a vertex/edge certificate, or a chain certificate.
2. For each $0 \leq i < \ell, C_i \cup C_{i+1}$ has an edge certificate.

A set $X$ has **chain certificate** $(C_0, C_1, \ldots, C_\ell)$ if

$$X = \bigcup_{i=0}^{\ell} C_i.$$

# Minimal Mincuts and Certificates




The cut (red) has a chain certificate (purple).

A cut $(X, V \setminus X)$ will be simply denoted by $X$.

**Definition**

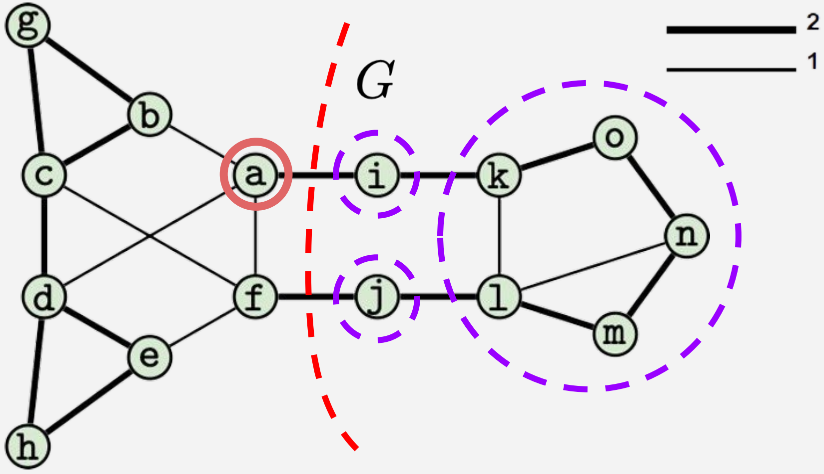
A **chain certificate** is a sequence of disjoint non-empty vertex subsets $(C_0, C_1, \ldots, C_\ell)$ where $\ell \geq 1$, and recursively:

1. For each $i, C_i$ has either a vertex/edge certificate, or a chain certificate.
2. For each $0 \leq i < \ell, C_i \cup C_{i+1}$ has an edge certificate.

A set $X$ has **chain certificate** $(C_0, C_1, \ldots, C_\ell)$ if

$$X = \bigcup_{i=0}^{\ell} C_i.$$

# Minimal Mincuts and Certificates




The cut (red) has a chain certificate (purple).

**Lemma**

$X$, which has a chain certificate, is either a mincut or the vertex set $V$.

**Definition**

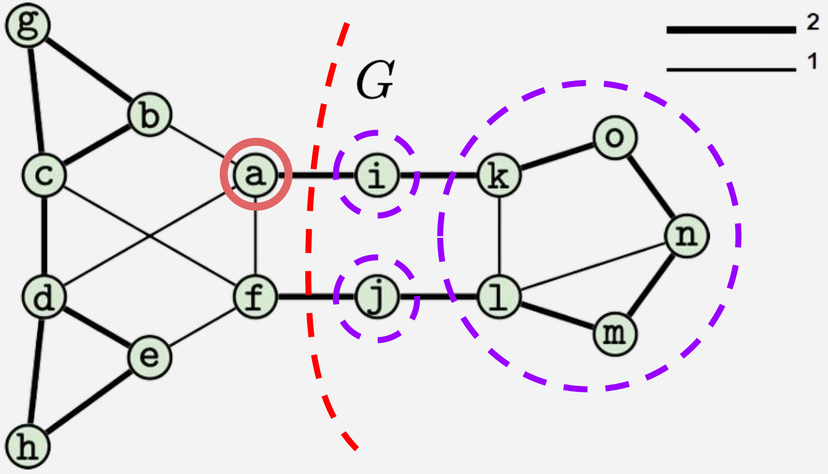
A **chain certificate** is a sequence of disjoint non-empty vertex subsets $(C_0, C_1, \dots, C_\ell)$ where $\ell \geq 1$, and recursively:

1. For each $i, C_i$ has either a vertex/edge certificate, or a chain certificate.
2. For each $0 \leq i < \ell, C_i \cup C_{i+1}$ has an edge certificate.

A set $X$ has **chain certificate** $(C_0, C_1, \dots, C_\ell)$ if

$$X = \bigcup_{i=0}^{\ell} C_i.$$

# Minimal Mincuts and Certificates




**Lemma**

Every mincut on $G$ has either a vertex certificate, an edge certificate, or a chain certificate.

**Lemma**

$X$, which has a chain certificate, is either a mincut or the vertex set $V$.

**Definition**

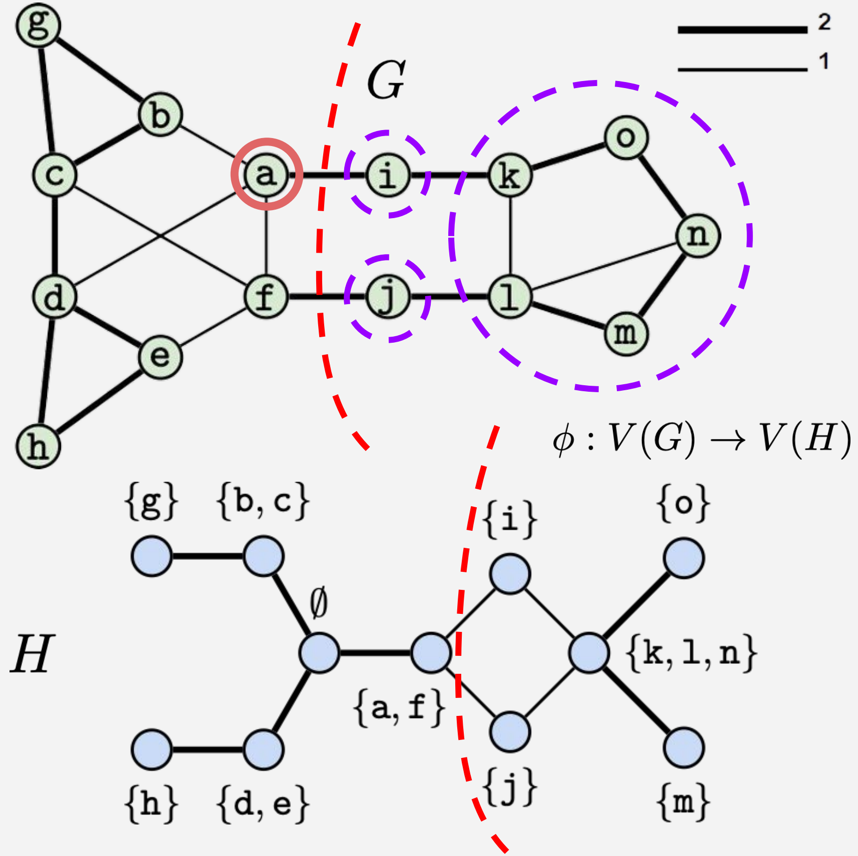
A **chain certificate** is a sequence of disjoint non-empty vertex subsets $(C_0, C_1, \ldots, C_\ell)$ where $\ell \geq 1$, and recursively:

1. For each $i, C_i$ has either a vertex/edge certificate, or a chain certificate.
2. For each $0 \leq i < \ell, C_i \cup C_{i+1}$ has an edge certificate.

A set $X$ has **chain certificate** $(C_0, C_1, \ldots, C_\ell)$ if

$$X = \bigcup_{i=0}^{\ell} C_i.$$

# Minimal Mincuts and Certificates




**Lemma**

$X$, which has a chain certificate, is either a mincut or the vertex set $V$.

**Definition**

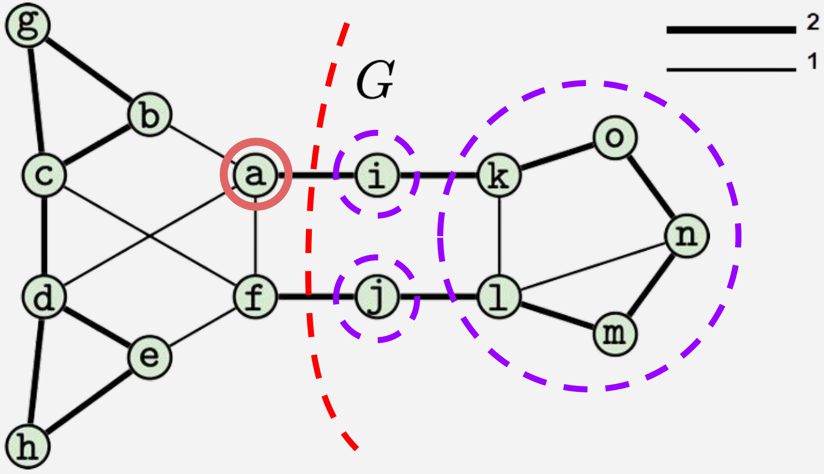
A **chain certificate** is a sequence of disjoint non-empty vertex subsets $(C_0, C_1, \ldots, C_\ell)$ where $\ell \geq 1$, and recursively:

1. For each $i$, $C_i$ has either a vertex/edge certificate, or a chain certificate.
2. For each $0 \leq i < \ell$, $C_i \cup C_{i+1}$ has an edge certificate.

A set $X$ has **chain certificate** $(C_0, C_1, \ldots, C_\ell)$ if

$$X = \bigcup_{i=0}^{\ell} C_i.$$

# Minimal Mincuts and Certificates




**Lemma**

If $X$ and $Y$ are crossing mincuts, then each of $X \cap Y, X \setminus Y, Y \setminus X, X \cup Y$ is also mincut.

**Lemma**

$X$, which has a chain certificate, is either a mincut or the vertex set $V$.

**Definition**

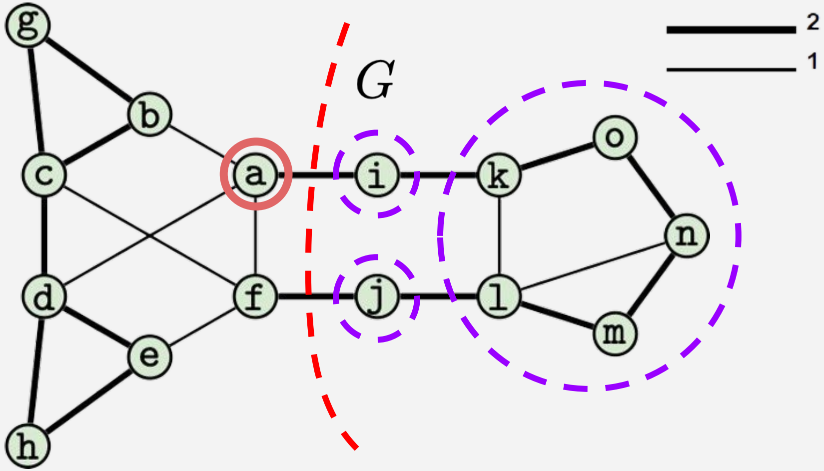
A **chain certificate** is a sequence of disjoint non-empty vertex subsets $(C_0, C_1, \ldots, C_\ell)$ where $\ell \geq 1$, and recursively:

1. For each $i, C_i$ has either a vertex/edge certificate, or a chain certificate.
2. For each $0 \leq i < \ell, C_i \cup C_{i+1}$ has an edge certificate.

A set $X$ has **chain certificate** $(C_0, C_1, \ldots, C_\ell)$ if

$$X = \bigcup_{i=0}^{\ell} C_i.$$

# Minimal Mincuts and Certificates



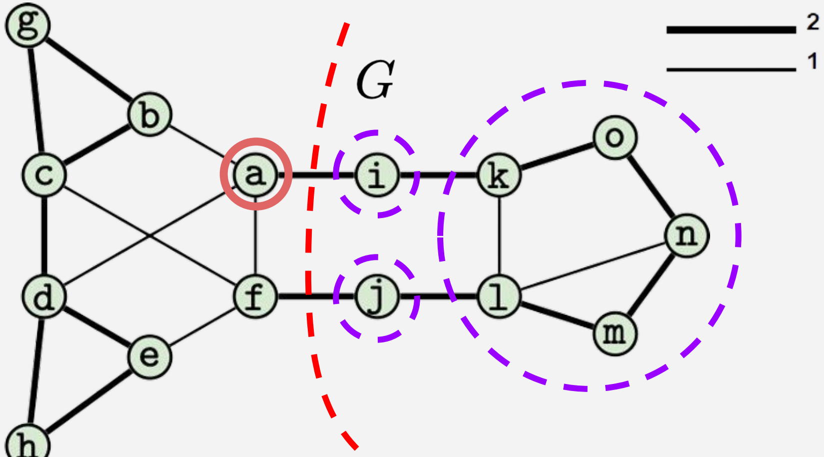

**Lemma**

If $X$ and $Y$ are crossing mincuts, then each of $X \cap Y, X \setminus Y, Y \setminus X, X \cup Y$ is also mincut.

**Lemma**

Every mincut on $G$ has either a vertex certificate, an edge certificate, or a chain certificate.

# Minimal Mincuts and Certificates




We build the cactus by scanning the minimal mincuts from size small to large, and reduce to **containment query**.

**Theorem**

Given a graph $G$, a tree packing $T$ and the set of (2-respecting) cuts representing minimal mincuts of each vertex $v$ and each edge $e$, we can deterministic computes a cactus representation of $G$ in $O(m\alpha(m,n) + n|T|)$ time.

**Lemma**

Every mincut on $G$ has either a vertex certificate, an edge certificate, or a chain certificate.

# Karger's Near-Linear Time Algorithm

1. Compute a tree packing of size $O(\log n)$ such that each mincut 2-respects ⅓ of them w.h.p.
2. For each tree in the packing, compute a minimum 2-respecting cut on the tree.
3. Take the minimum over all the trees in step 2.

# Cactus Constuction Algorithm

1. Compute a tree packing of size $O(\log n)$ such that each mincut 2-respects ⅓ of them w.h.p.
2. For each tree in the packing, compute a minimal 2-respecting mincut for every vertex and edge on the tree.
3. Building the cactus representation using the minimal mincuts from step 2.

# Cactus Constuction Algorithm (Deterministic)

1. Compute a tree packing of size $(\log n)^{O(1)}$ such that each mincut 2-respects ⅓ of them.
2. For each tree in the packing, compute a minimal 2-respecting mincut for every vertex and edge on the tree.
3. Building the cactus representation using the minimal mincuts from step 2.

We make step 2&3 deterministic, faster, and more modular.

# Outline

1 Tree Packing

2 Minimal Mincuts and Cactus Construction

3 **Karger's 2-Respecting Mincuts Algorithm**

4 Compute 2-Respecting Minimal Mincuts

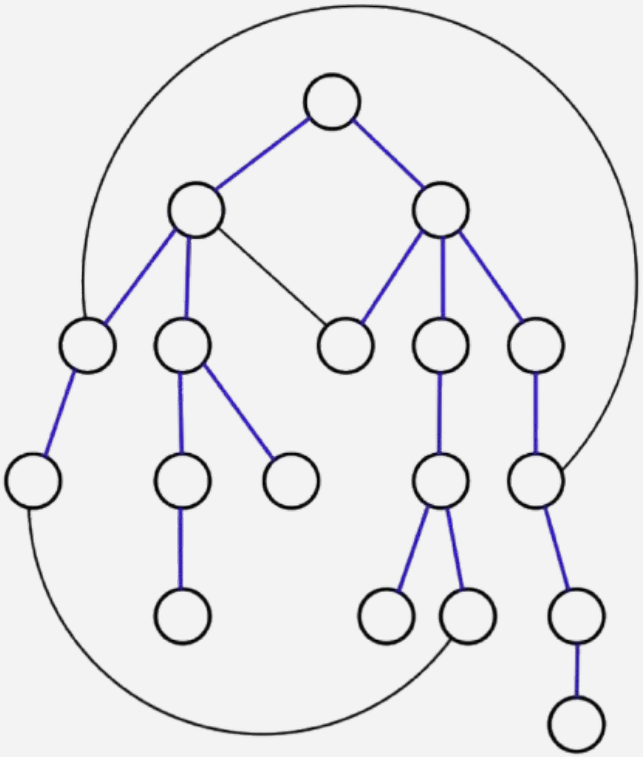
# Karger’s Near-Linear Time Algorithm

1. Compute a tree packing of size $O(\log n)$ such that each mincut 2-respects ⅓ of them w.h.p.
2. **For each tree in the packing, compute a minimum 2-respecting cut on the tree.**
3. Take the minimum over all the trees in step 2.

*Reference: slides by Bryce Sandlund [BLS 2020].*

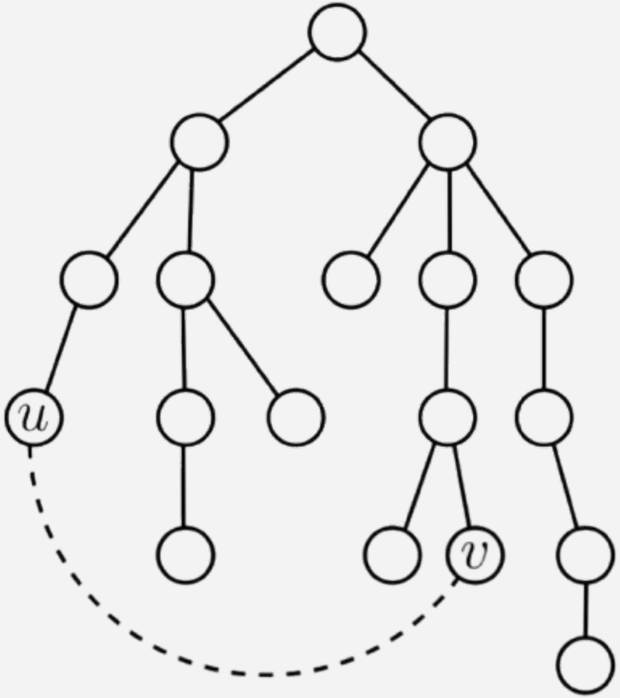
# 1-Respect Algorithm

Given a spanning tree $T$ of a graph $G$, find a smallest cut of $G$ that cuts one edge of $T$.



# 1-Respect Algorithm

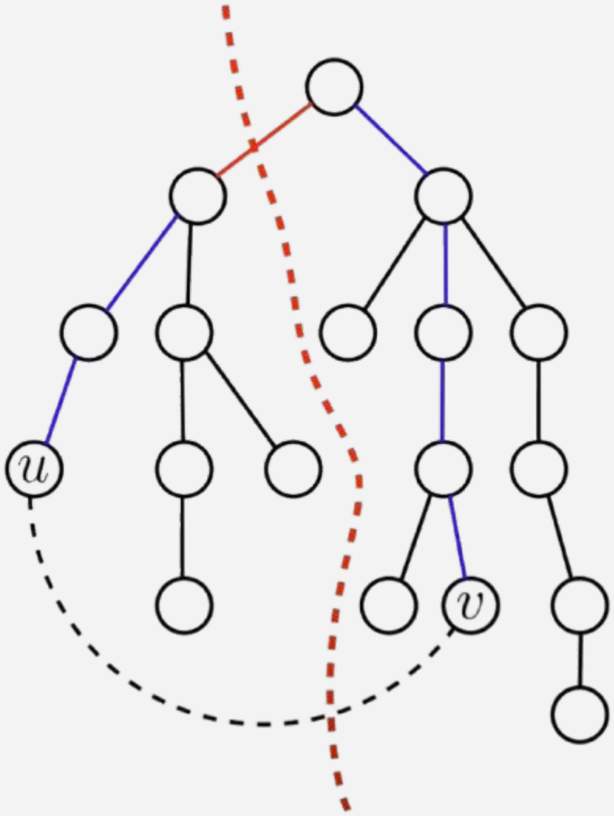
When is a non-tree edge $uv$ cut?



# 1-Respect Algorithm

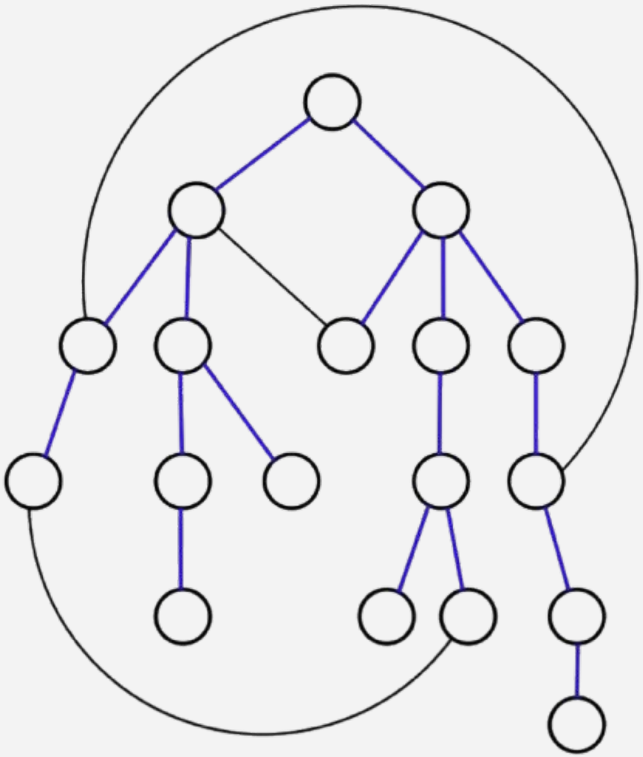
When is a non-tree edge $uv$ cut?

Non tree-edge $uv$ is cut iff the cut in $G$ cuts an edge on the $uv$-path on $T$.



# 1-Respect Algorithm

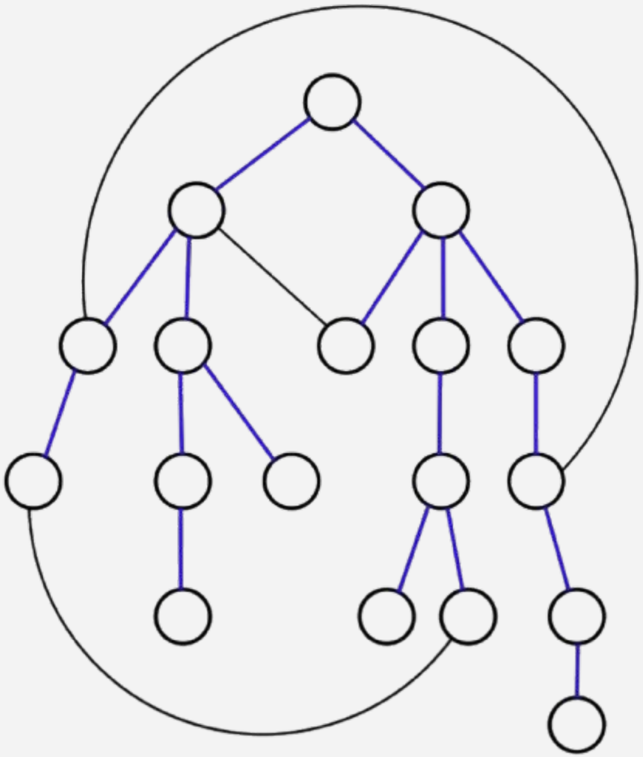
How to compute the set of all $n-1$ cuts that 1-respects $T$?



# 1-Respect Algorithm

How to compute the set of all $n-1$ cuts that 1-respects $T$?

Idea: Iterate an edge $e$ through $T$, keeping track of non-tree edges that cross a cut at $e$.

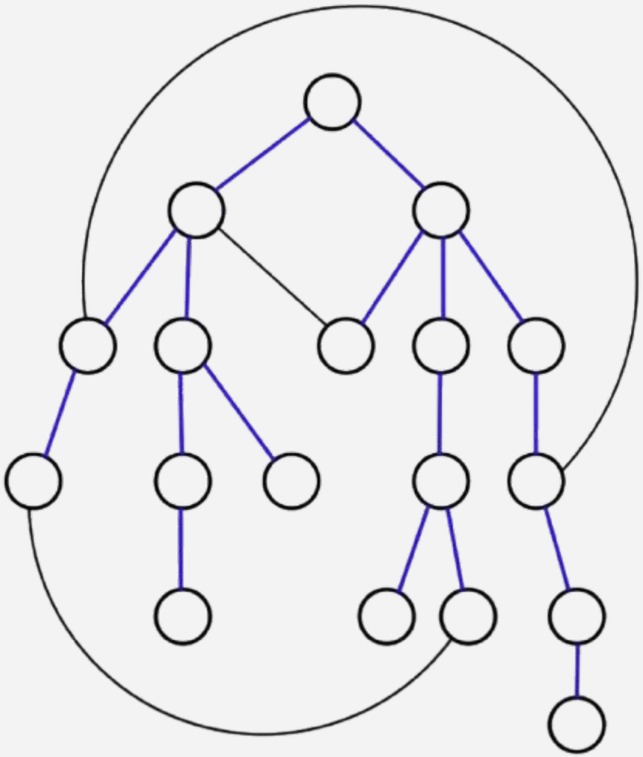


# 1-Respect Algorithm

How to compute the set of all $n-1$ cuts that 1-respects $T$?

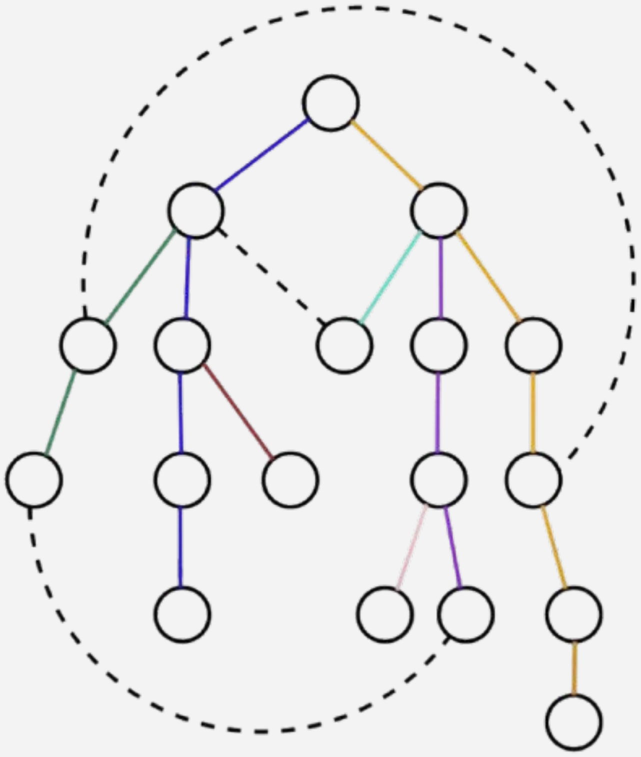
Idea: Iterate an edge $e$ through $T$, keeping track of non-tree edges that cross a cut at $e$.

Is there an order of edges $e$ that results in non-tree edges transitioning on and off the current cut a small number of times?



# Heavy-Light Decomposition

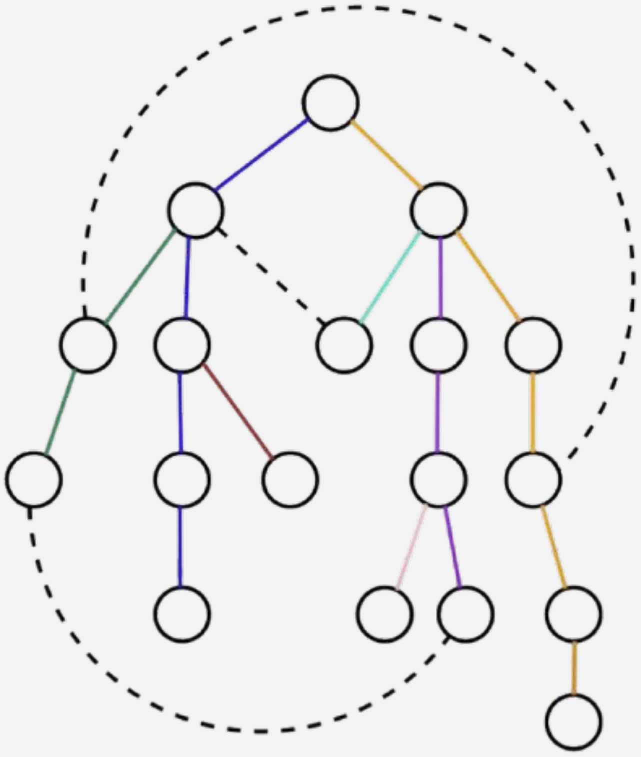
1. Split $T$ into root-to-leaf paths.
2. Continue the path to the child with the most descendants.



# Heavy-Light Decomposition

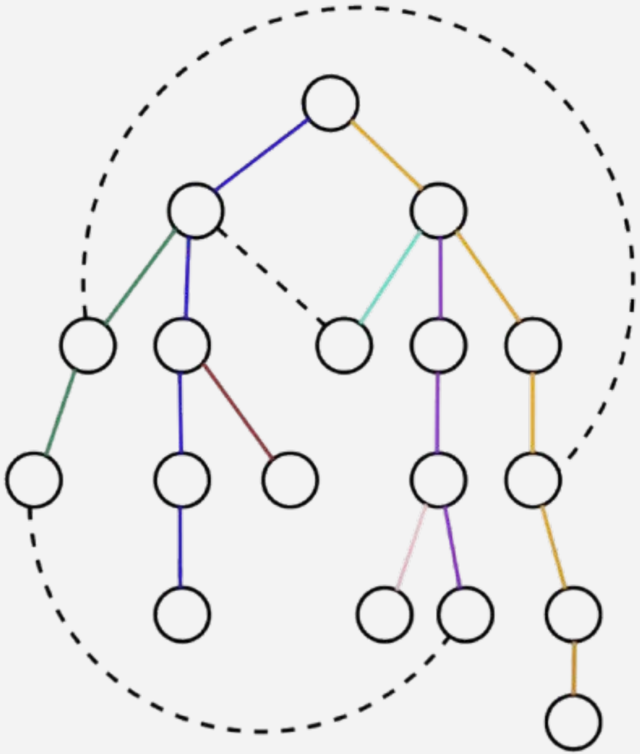
1. Split $T$ into root-to-leaf paths.
2. Continue the path to the child with the most descendants.

Any root-to-leaf paths requires at most $O(\log n)$ color changes.



# 1-Respect Algorithm

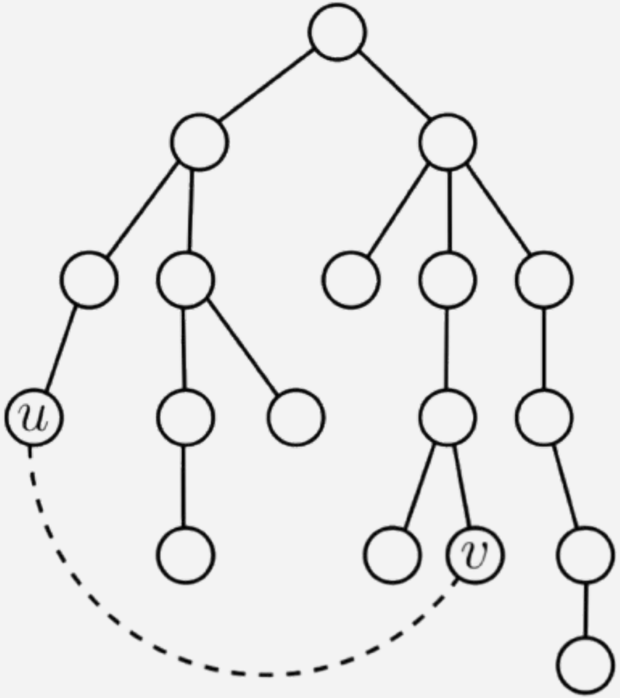
1. Iterate edge $e$ in heavy-light decomposition order
2. Keep track of non-tree edges that cross a cut at $e$.



Non tree edge $uv$ will transition on or off the current cut $O(\log n)$ times.

# 2-Respect Algorithm

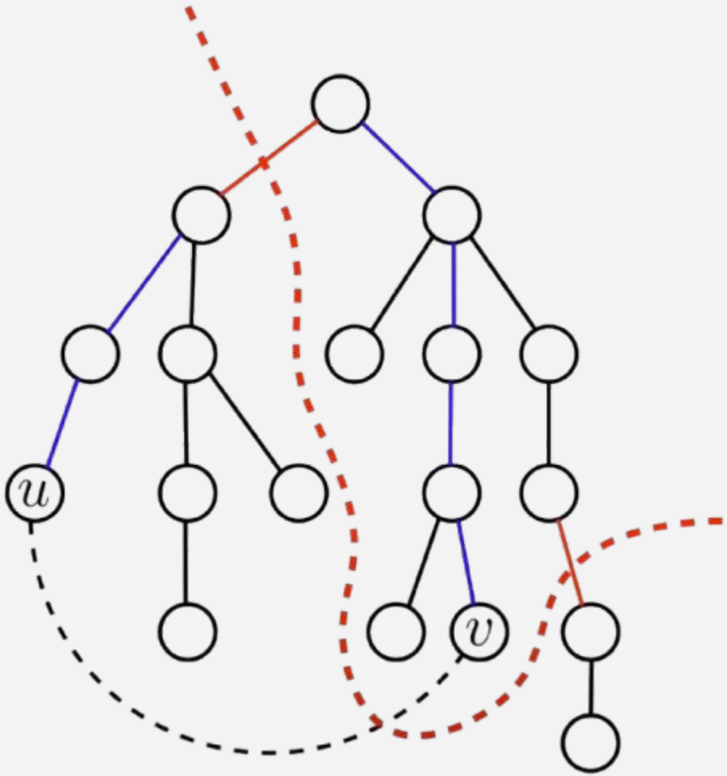
When two edges of $T$ are cut, when does a non-tree edge $uv$ cross the cut?



# 2-Respect Algorithm

When two edges of $T$ are cut, when does a non-tree edge $uv$ cross the cut?

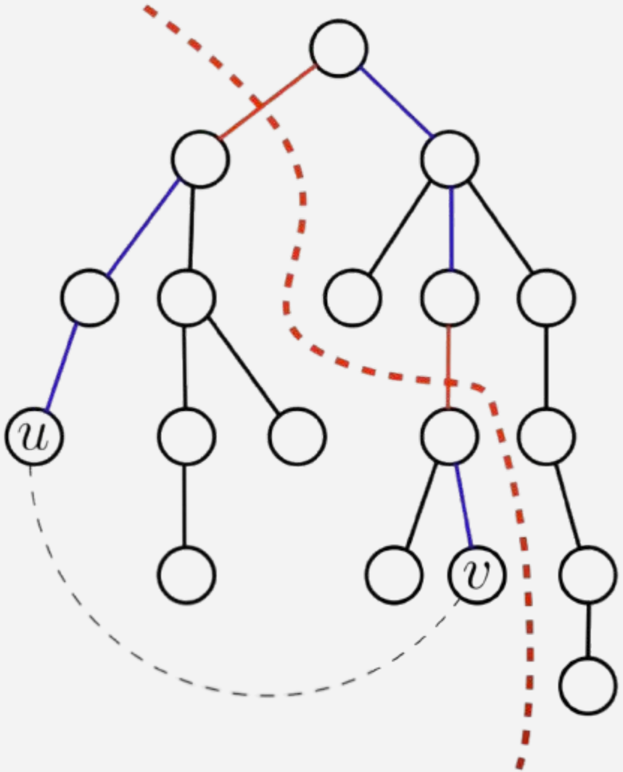
Non tree-edge $uv$ is cut iff the cut in $G$ cuts exact one edge on the $uv$-path on $T$.



# 2-Respect Algorithm

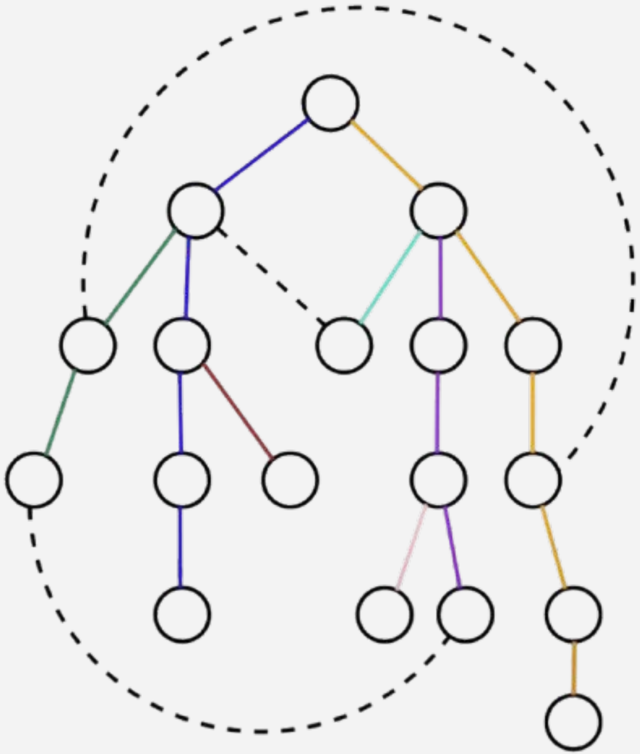
When two edges of $T$ are cut, when does a non-tree edge $uv$ cross the cut?

Non tree-edge $uv$ is cut iff the cut in $G$ cuts exact one edge on the $uv$-path on $T$.



# 2-Respect Algorithm

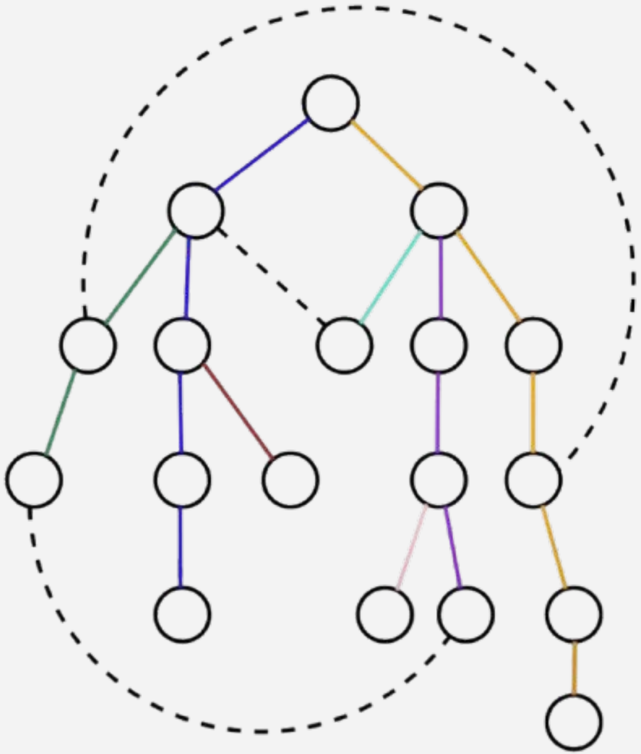
How can we leverage our 1-respect strategy for cuts that cut two edges of $T$?



# 2-Respect Algorithm

How can we leverage our 1-respect strategy for cuts that cut two edges of $T$?

We cannot spend $\Omega(n^2)$ time checking all the cuts.



# Top Tree Data Structure

Operations over a weighted tree $T$.

- *PathAdd(u,v,w)* : Add weight $w$ to all edges on the $uv$-path in $T$.
- *NonPathAdd(u,v,w)* : Add weight $w$ to all edges not on the uv-path in $T$.
- *QueryMinimum()* : Return the minimum weight edge in $T$.

# Top Tree Data Structure

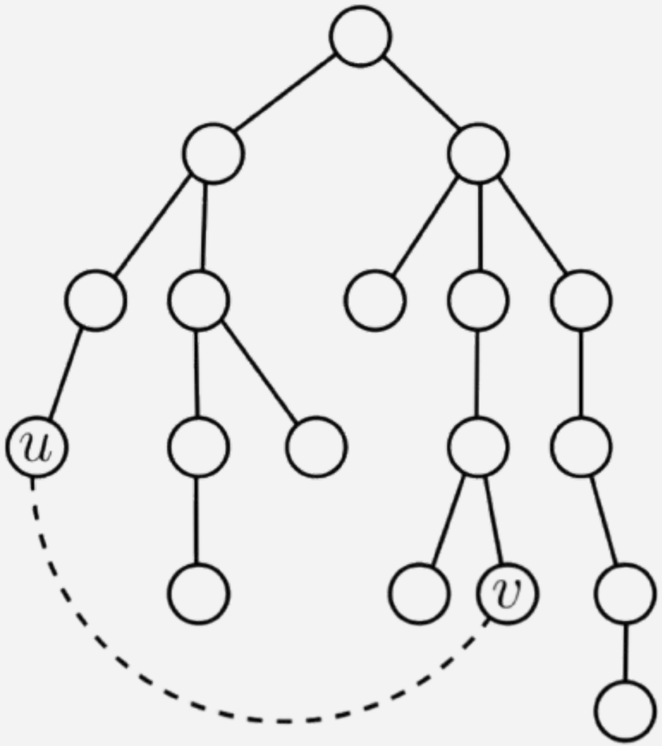
Operations over a weighted tree $T$.

- *PathAdd(u,v,w)* : Add weight $w$ to all edges on the $uv$-path in $T$.
- *NonPathAdd(u,v,w)* : Add weight $w$ to all edges not on the uv-path in $T$.
- *QueryMinimum()* : Return the minimum weight edge in $T$.

All operations take $O(\log n)$ time.

# 2-Respect Algorithm

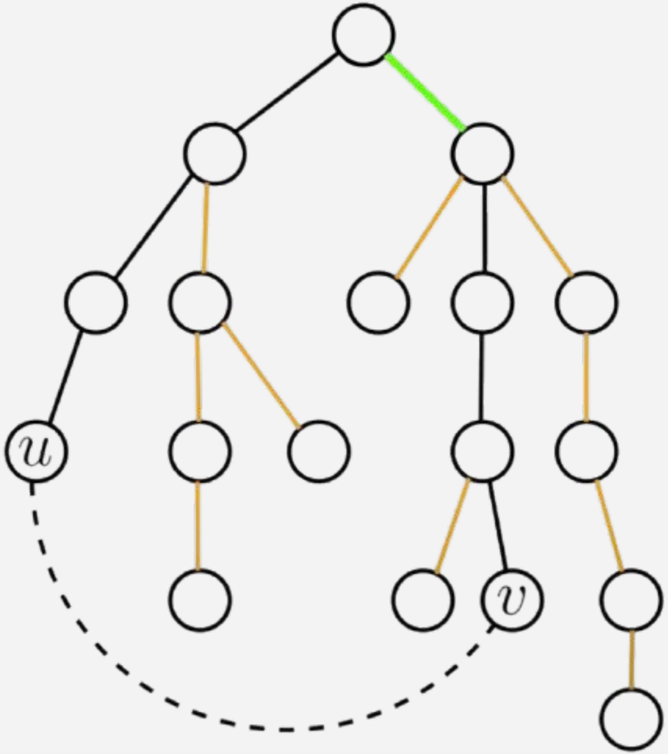
Call the two tree edges that we cut $e$ and $f$. If we fix $e$, we can determine which $f$ result in non-tree edge $uv$ cross the cut.



# 2-Respect Algorithm

Call the two tree edges that we cut $e$ and $f$. If we fix $e$, we can determine which $f$ result in non-tree edge $uv$ cross the cut.

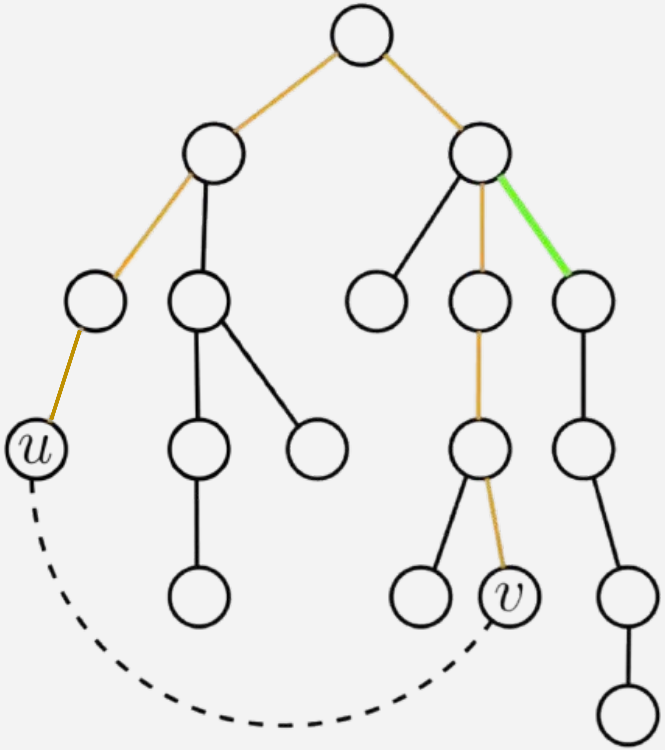
- If $e$ is on $uv$-path, any $f$ off the $uv$-path cut $uv$.



# 2-Respect Algorithm

Call the two tree edges that we cut $e$ and $f$. If we fix $e$, we can determine which $f$ result in non-tree edge $uv$ cross the cut.

- If $e$ is on $uv$-path, any $f$ off the $uv$-path cut $uv$.
- If $e$ is off $uv$-path, any $f$ on the $uv$-path cut $uv$.



# Top Tree Data Structure

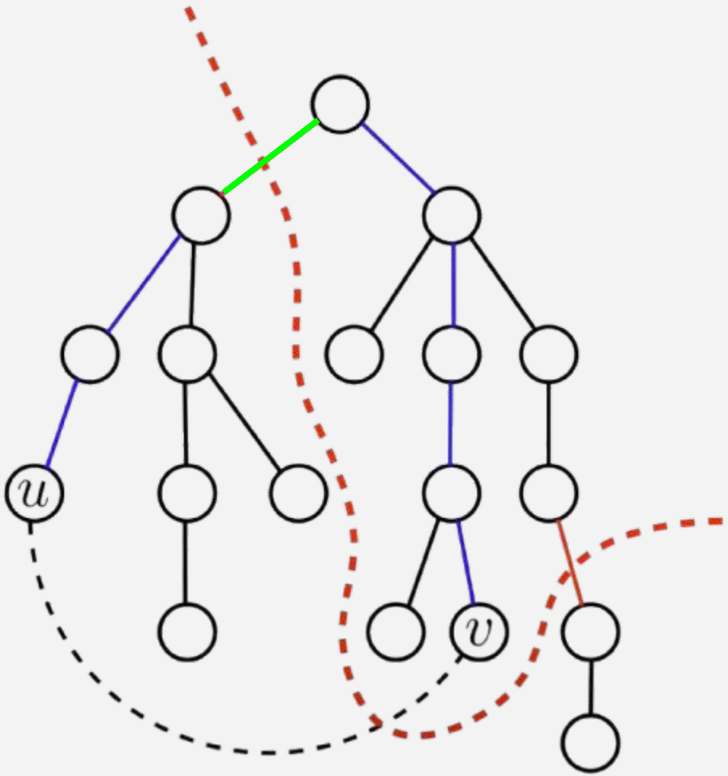
Operations over a weighted tree $T$.

- *PathAdd(u,v,w)* : Add weight $w$ to all edges on the $uv$-path in $T$.
- *NonPathAdd(u,v,w)* : Add weight $w$ to all edges not on the uv-path in $T$.
- *QueryMinimum()* : Return the minimum weight edge in $T$.

All operations take $O(\log n)$ time.

# 2-Respect Algorithm

Call the two tree edges that we cut $e$ and $f$. If we fix $e$, we can determine which $f$ result in non-tree edge $uv$ cross the cut.




- If $e$ is on $uv$-path, any $f$ off the $uv$-path cut $uv$.
- If $e$ is off $uv$-path, any $f$ on the $uv$-path cut $uv$.

Use top tree to find best $f$!

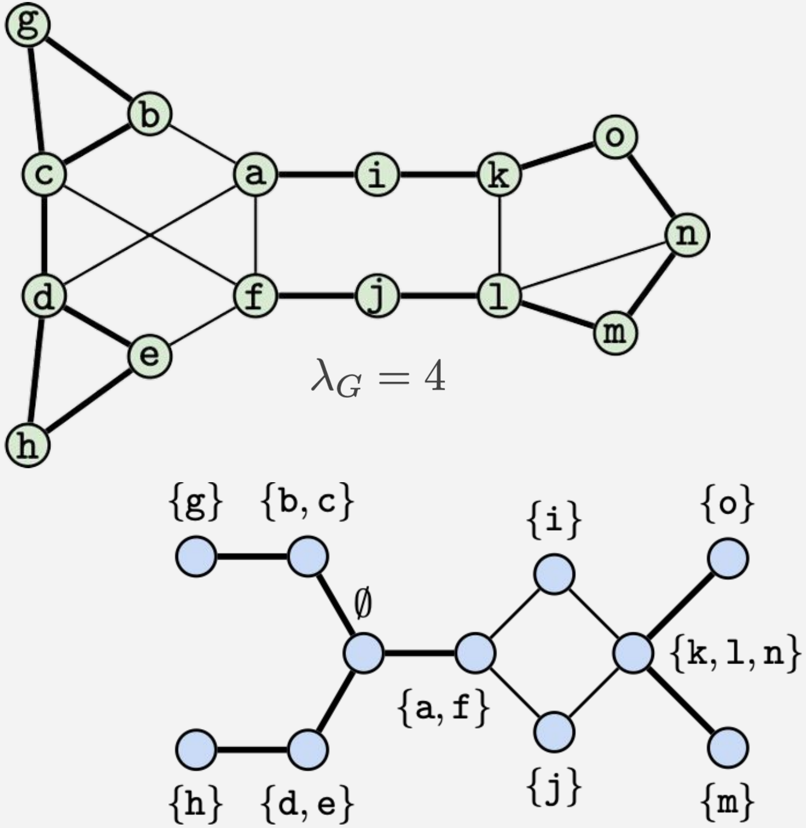
# Outline

1. Tree Packing
2. Minimal Mincuts and Cactus Construction
3. Karger's 2-Respecting Mincuts Algorithm
4. **Compute 2-Respecting Minimal Mincuts**

# Cactus Constuction Algorithm

1. Compute a tree packing of size $O(\log n)$ such that each mincut 2-respects ⅓ of them w.h.p.
2. For each tree in the packing, compute a minimal 2-respecting mincut for every vertex and edge on the tree. (very technical)
3. Building the cactus representation using the minimal mincuts from step 2.

# Summary of Part I




A **cactus** is a graph where every edge belongs to at most one (simple) cycle.

[Dinitz et al. 1976]
There exists an **O(*n*)** sized **cactus graph** that preserves *all* mincuts of the given graph.

**Randomized** Algorithms:

$O(m \log^4 n) \longrightarrow O(m \log^3 n)$

[Karger & Panigrahi 2009] [**He**, Huang, Saranurak 2024]

**Deterministic** Algorithms:

$O(m\,\mathrm{polylog}(n))$

[**He**, Huang, Saranurak 2024] + [Henzinger, Li, Rao, Wang 2024]

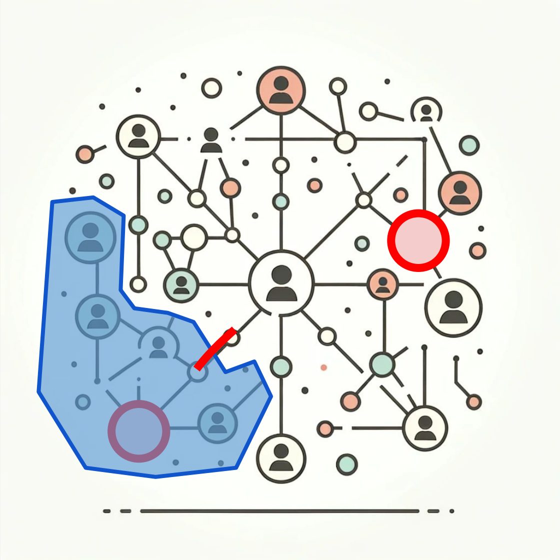
# Part II

# Cactus Representations in Polylogarithmic Max-flow via Maximal Isolating Mincuts

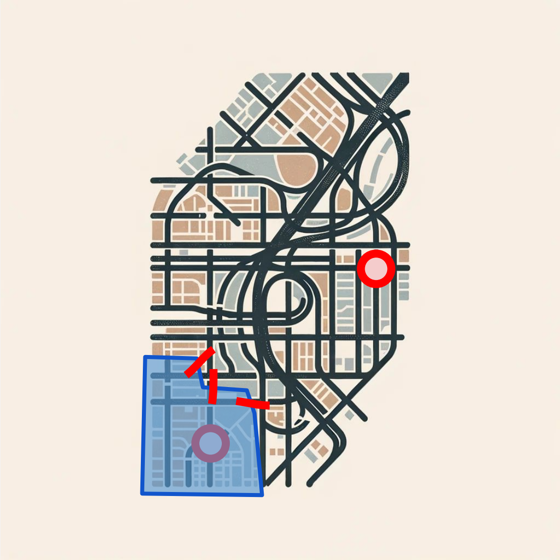
*Images generated by ChatGPT 4.0

# Steiner Mincuts

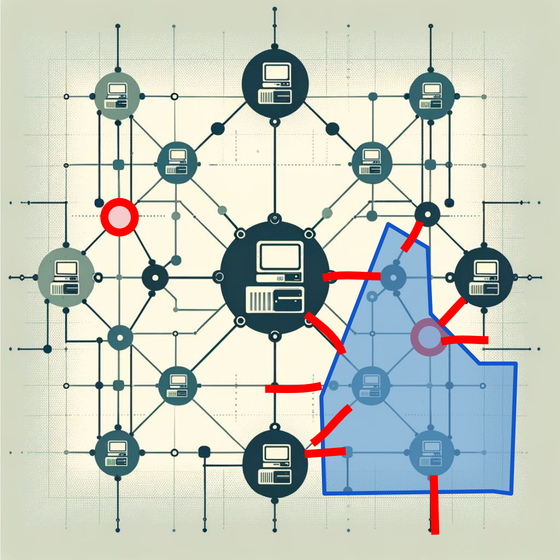
"Find a proper subset $\mathbf{X}$ of $\boldsymbol{T}$ such that the cost to disconnecting $\mathbf{X}$ from $\boldsymbol{T}\setminus\mathbf{X}$ is minimized."



Social Network

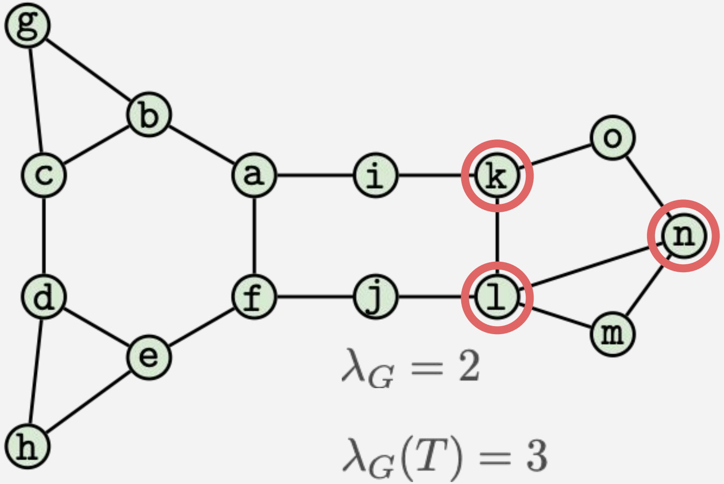


Road Network



Computer Network

# Steiner Mincuts

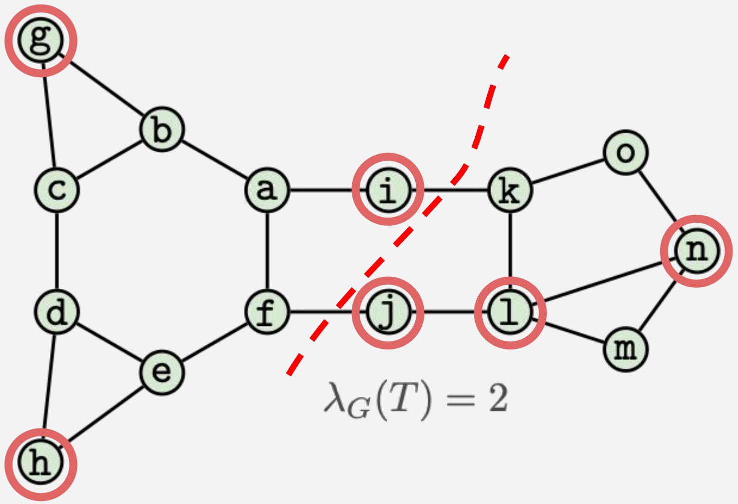


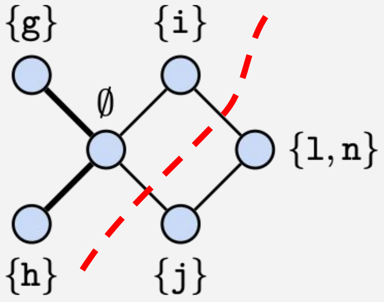

$\lambda_G = 2$

$\lambda_G(T) = 3$

A **T-mincut** is a “partition” of ***T*** that has a minimum possible valued cut among all non-trivial partition of ***T***.

# Steiner Mincuts



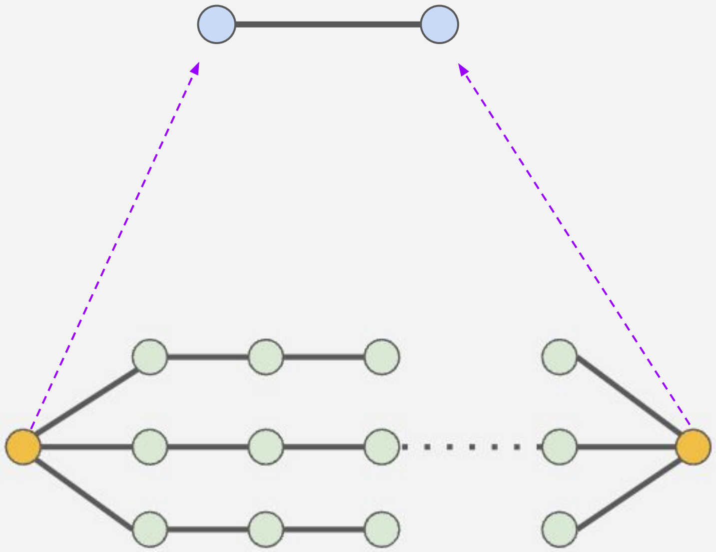




A **T-mincut** is a "partition" of ***T*** that has a minimum possible valued cut among all non-trivial partition of ***T***.

**Theorem** [Karger 1993][Dinitz et al. 1994]

There exists an **O(|*T*|)** sized **cactus graph** that preserves *all* ***T***-mincuts of the given graph.

# Steiner Mincuts

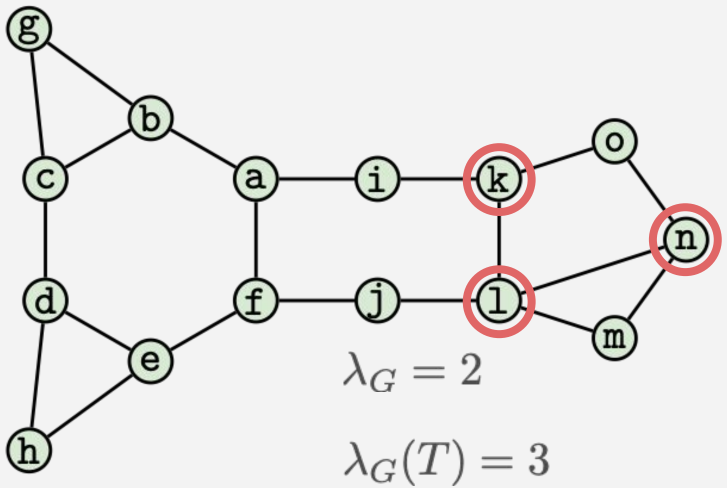


A **T-mincut** is a "partition" of ***T*** that has a minimum possible valued cut among all non-trivial partition of ***T***.

**Theorem** [Karger 1993][Dinitz et al. 1994]

There exists an **O(|*T*|)** sized **cactus graph** that preserves *all* ***T***-mincuts of the given graph.

# Steiner Mincuts: Our Result




$\lambda_G = 2$

$\lambda_G(T) = 3$

A **T-mincut** is a "partition" of ***T*** that has a minimum possible valued cut among all non-trivial partition of ***T***.

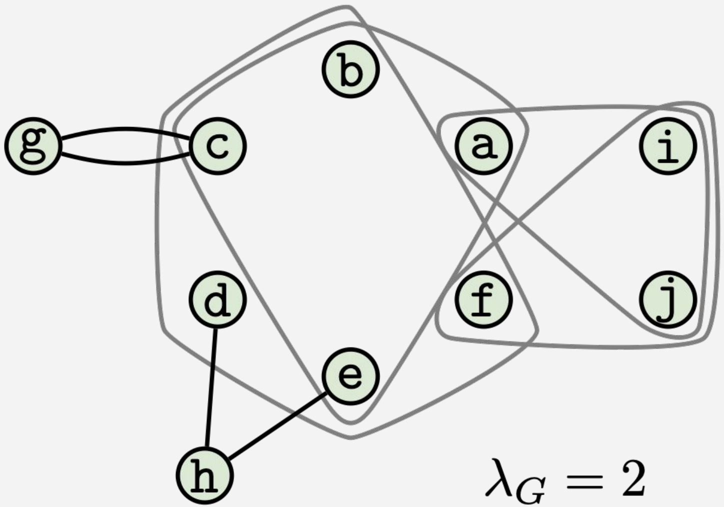
**Theorem** [Karger 1993][Dinitz et al. 1994]

There exists an **O(|*T*|)** sized **cactus graph** that preserves *all* **T**-mincuts of the given graph.

Computing cactus representation of Steiner mincuts:

| | |
|---|---|
| Dinitz and Vainshtein [1994] | $O(\|T\| \cdot \mathrm{MaxFlow}(m))$ |
| Cole and Hariharan [2003] | $\tilde{O}(m + \lambda_G(T) \cdot n)$ |
| **He**, Huang, and Saranurak [2024] | $O(\log^4 n \cdot \mathrm{MaxFlow}(O(m)))$ |

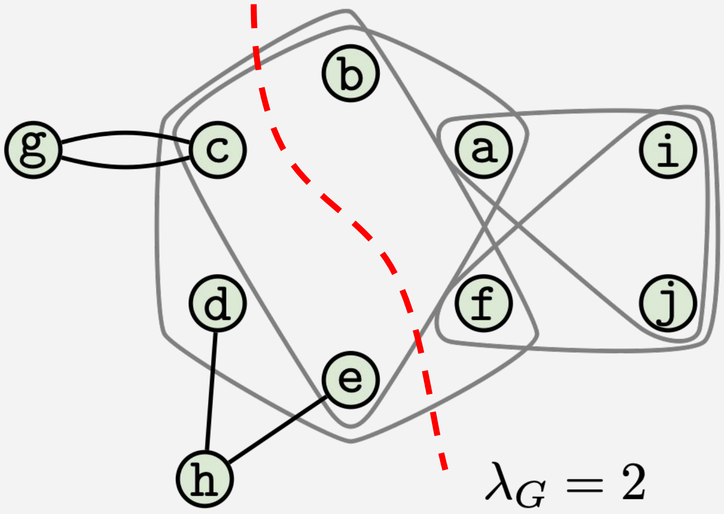
# Hypergraph Mincuts: Hypercactus Representation



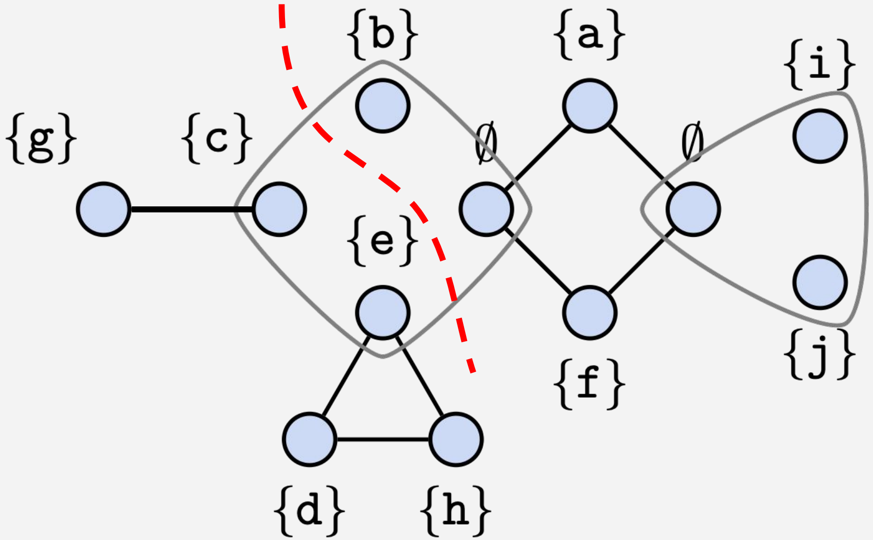

$\lambda_G = 2$

A Hypergraph

# Hypergraph Mincuts: Hypercactus Representation




A Hypergraph

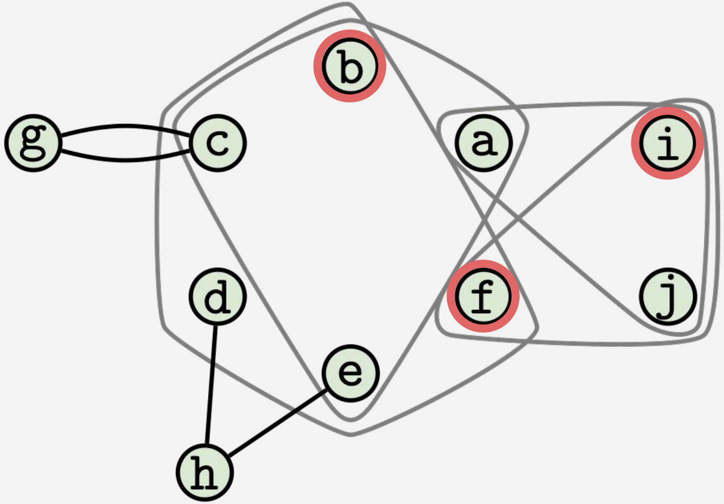



Hypercactus Representation

**Theorem** [Fleiner & Jordán 1999]

There exists an $\mathbf{O(n)}$ sized **hypercactus graph** that preserves *all* mincuts of the given hypergraph.

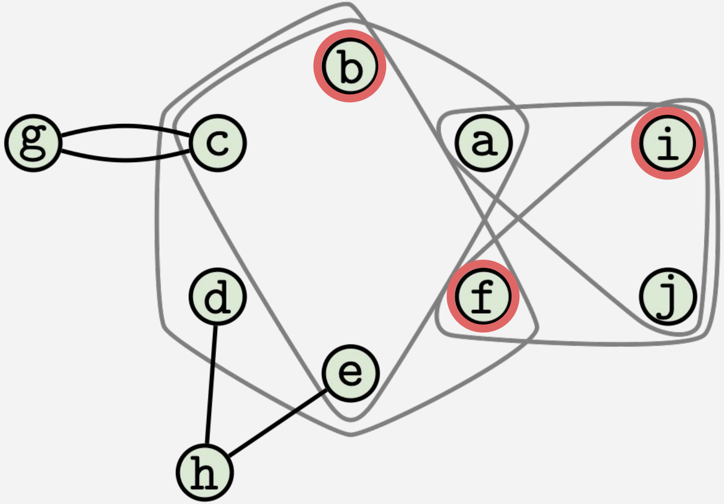
# Steiner Hypergraph Mincuts: Our Result




**Theorem** [Fleiner & Jordán 1999]

There exists an **O(|*T*|)** sized **hypercactus graph** that preserves *all* ***T***-mincuts of the given hypergraph.

# Steiner Hypergraph Mincuts: Our Result




**Theorem** [Fleiner & Jordán 1999]

There exists an **O(|*T*|)** sized **hypercactus graph** that preserves *all* ***T***-mincuts of the given hypergraph.

## Algorithms:

$$O(|T| \cdot \mathrm{MaxFlow}(O(p))) \longrightarrow O(\log^4 n \cdot \mathrm{MaxFlow}(O(p)))$$

[Chekuri & Xu 2017] + [Fleiner & Jordán 1999]

[**He**, Huang, Saranurak 2024]

# Our results: Summary

| | Time | Steiner | Note |
|---|---|---|---|
| Karger and Panigrahi [2009] | $O(m \log^4 n)$ | No | Normal Graph |
| **He**, Huang, Saranurak [2024] | $O(m \log^3 n)$ | No | Normal Graph |
| Dinitz and Vainshtein [1994] | $O(\|T\| \cdot m^{1+o(1)})$ | Yes | Normal Graph |
| Chekuri and Xu [2017] | $O(\|T\| \cdot p^{1+o(1)})$ | Yes | Hypergraph |
| **He**, Huang, Saranurak [2024] | $O(m^{1+o(1)})$ | Yes | Normal Graph |
| **He**, Huang, Saranurak [2024] | $O(p^{1+o(1)})$ | Yes | Hypergraph |

This talk

# Outline

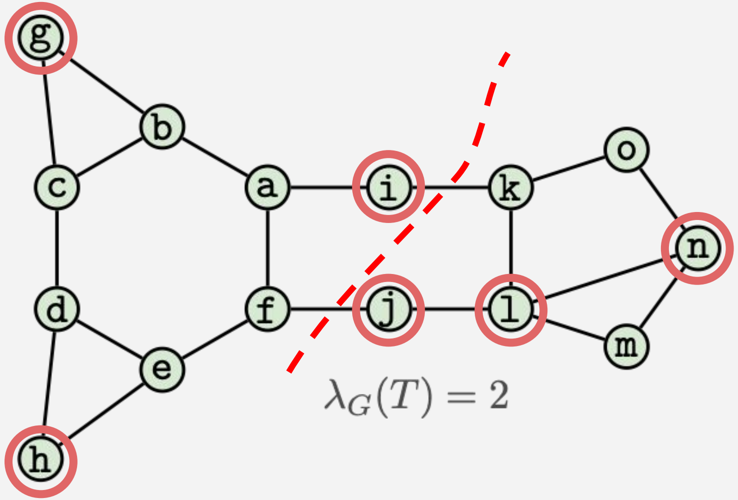
1. Divide and Conquer Framework to Compute the Cactus Representation
2. Minimal Isolating Mincuts
3. Why Maximal Isolating Mincuts (Novel Variant)
4. Compute Maximal Isolating Mincuts

# Outline

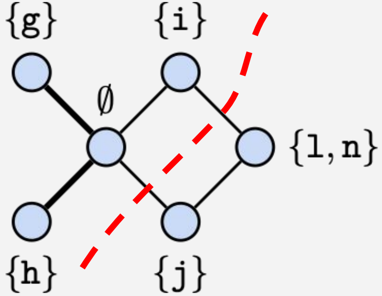
1. **Divide and Conquer Framework to Compute the Cactus Representation**
2. Minimal Isolating Mincuts
3. Why Maximal Isolating Mincuts (Novel Variant)
4. Compute Maximal Isolating Mincuts

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




1. Find any non-trivial ***T***-mincut (***T***-*split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

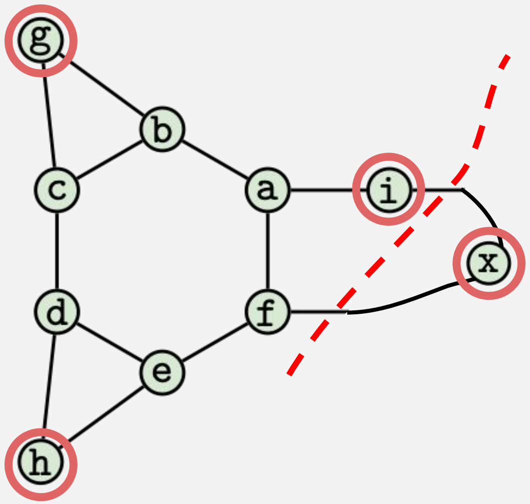



**Definition. T**-split [Chekuri and Xu 2017]

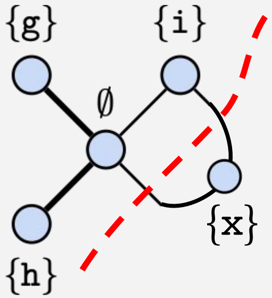
A T-split partitions the graph such that both side contain at least 2 terminal vertices.

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




1. Find any non-trivial ***T***-mincut (***T***-*split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

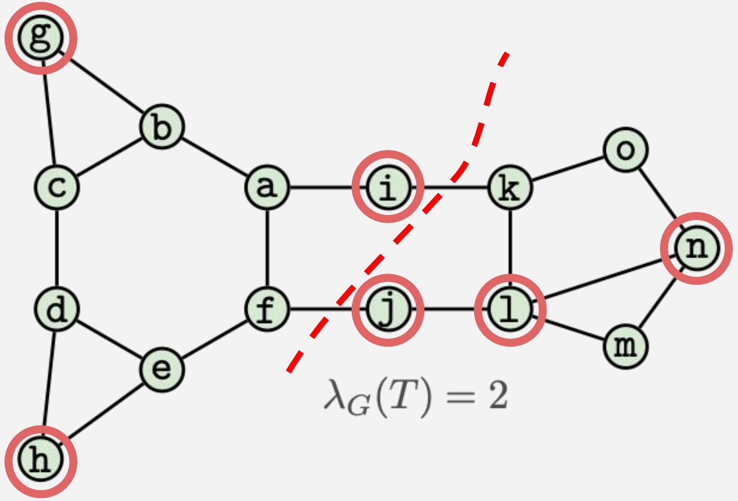



**Definition. T**-split [Chekuri and Xu 2017]

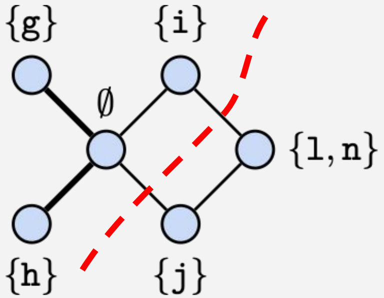
A T-split partitions the graph such that both side contain at least 2 terminal vertices.

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




1. Find any non-trivial ***T***-mincut (***T***-*split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

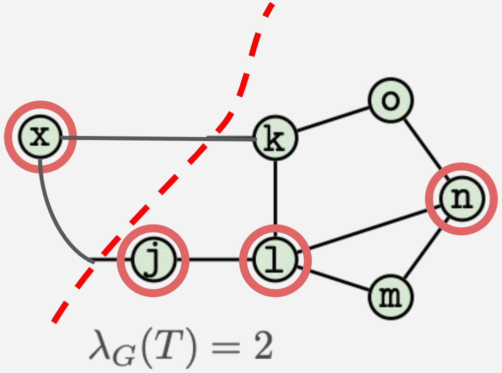



**Definition. T**-split [Chekuri and Xu 2017]

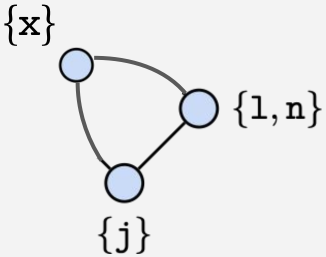
A T-split partitions the graph such that both side contain at least 2 terminal vertices.

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




1. Find any non-trivial ***T***-mincut (***T****-split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

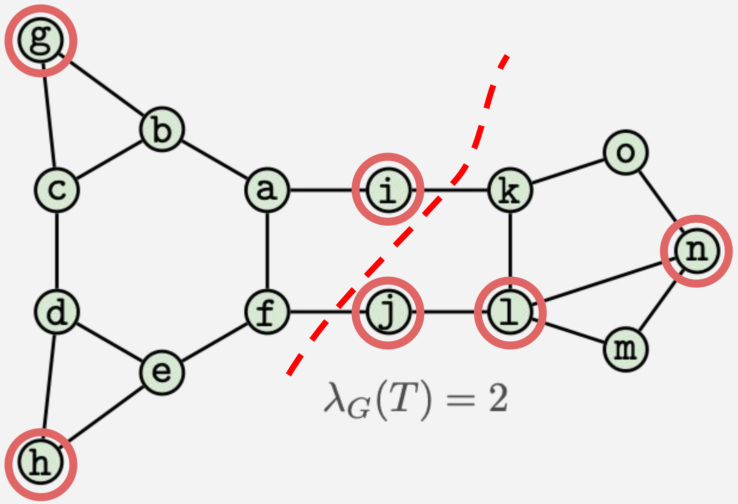



**Definition. T**-split [Chekuri and Xu 2017]

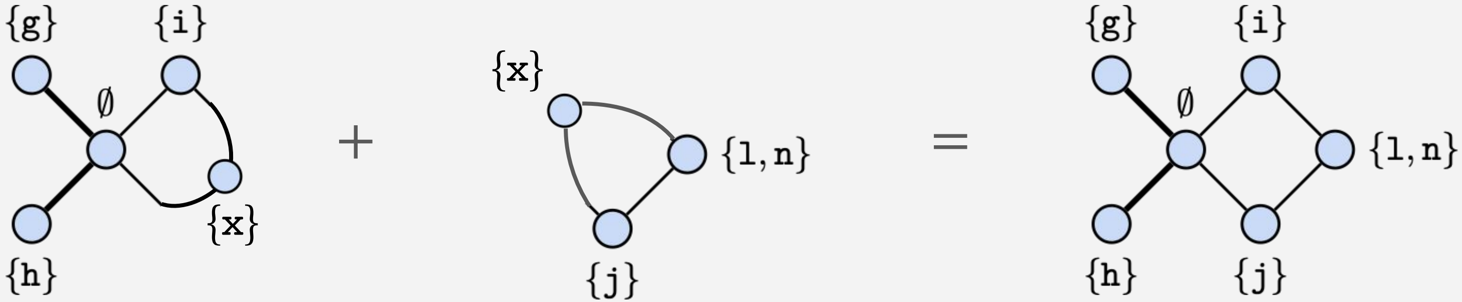
A T-split partitions the graph such that both side contain at least 2 terminal vertices.

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework

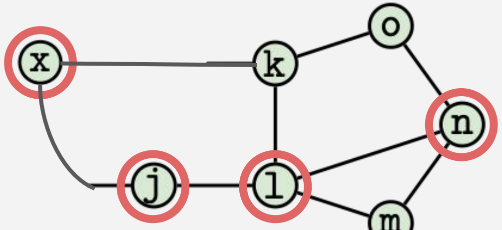



1. Find any non-trivial ***T***-mincut (***T****-split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the “sub-cacti”.



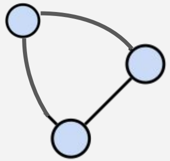
[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




$\lambda_G(T) = 2$

**Base case** (No T-split found): The cactus is either a triangle or a star.



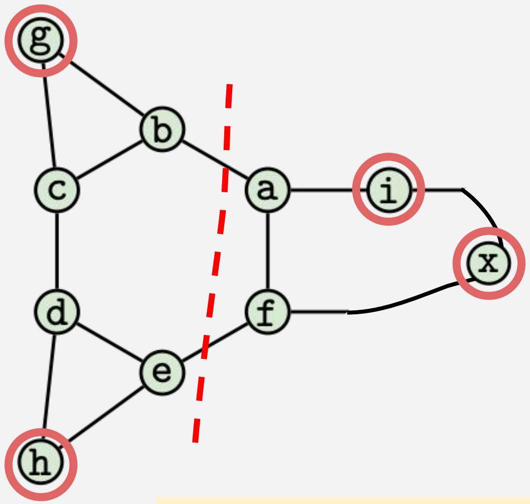

1. Find any non-trivial $\boldsymbol{T}$-mincut ($\boldsymbol{T}$-*split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

**Definition. $\boldsymbol{T}$-split** [Chekuri and Xu 2017]

A T-split partitions the graph such that both side contain at least 2 terminal vertices.

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




1. Find any non-trivial **T**-mincut (**T**-*split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

**Q1**: How to find **T**-splits ***efficiently***?

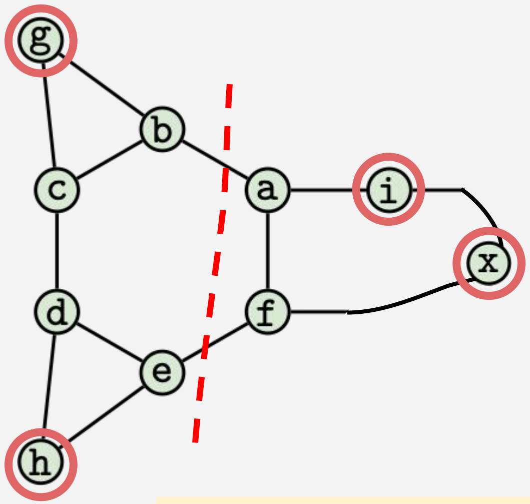
**A**: [Chekuri and Xu 2017] find some arbitrary **T**-splits using max-flow.

**Q2**: Can we bound the **depth** of divide and conquer?

**A**: We find some "balanced" **T**-splits, implies $O(\log n)$ depth!

[Chekuri and Xu 2017]

# A Contraction-Based Divide and Conquer Framework




1. Find any non-trivial **$T$**-mincut (**$T$**-*split*).
2. Contract all vertices on each side and recurse.
3. Recover the cactus representation from the "sub-cacti".

**Q1**: How to find **T**-splits ***efficiently***?

**A**: [Chekuri and Xu 2017] find some arbitrary **T**-splits using max-flow.

**Q2**: Can we bound the **depth** of divide and conquer?

**A**: We find some "balanced" **T**-splits, implies $O(\log n)$ depth!

Inspired by
**Isolating Mincuts**

# Outline

1 Divide and Conquer Framework to Compute the Cactus Representation

2 **Minimal Isolating Mincuts**

3 Why Maximal Isolating Mincuts (Novel Variant)

4 Compute Maximal Isolating Mincuts

# Minimal Isolating Mincuts

- This tool is simple and very powerful: 10+ papers in a few years
- It will not be useful for us though.
- We will introduce a new variant of this, but the basic definition of this part is useful.

# Outline


2 **Minimal Isolating Mincuts**

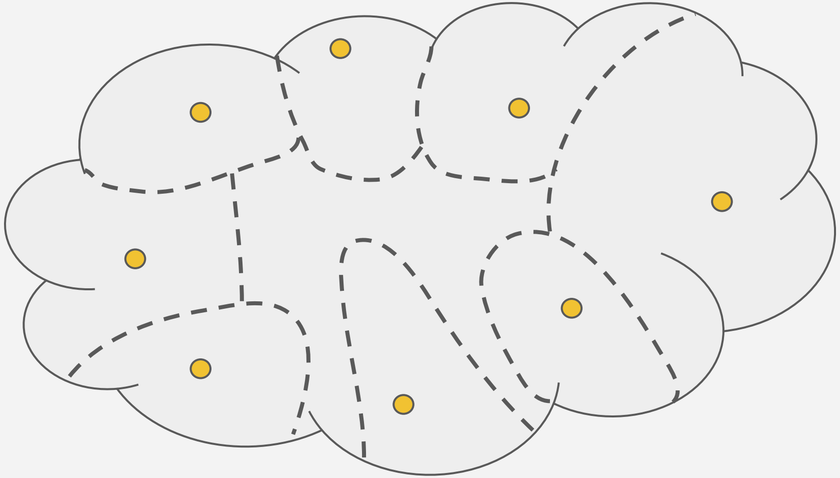
2.1 **Definition**

2.2 Simple Application: Steiner Mincut

# Isolating Mincuts [Li & Panigrahi 2020]

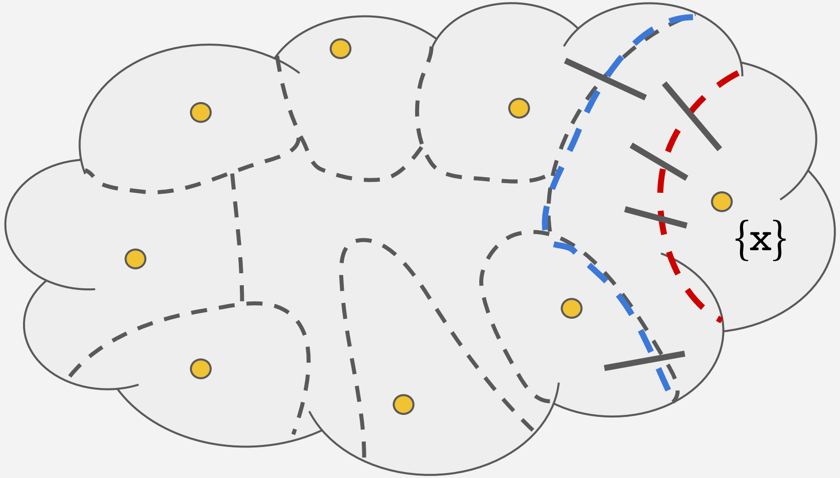
Given a set **T** of terminals, for each terminal $x \in T$ find a minimum valued cut that separate $x$ from $T \setminus \{x\}$.

***x*-mincut of *T***



# Minimal Isolating Mincuts [Li & Panigrahi 2020]

An $\boldsymbol{x}$-mincut of $\boldsymbol{T}$ is "minimal" if all proper subset containing $\boldsymbol{x}$ has a strictly larger cut boundary.




**Isolating Cuts Lemma** [Li & Panigrahi 2020]

Given a terminal set $\boldsymbol{T}$, there is an algorithm that computes the minimal isolating mincuts for every $\boldsymbol{x}$ in $\boldsymbol{T}$ using $O(\log |T|)$ max-flows.

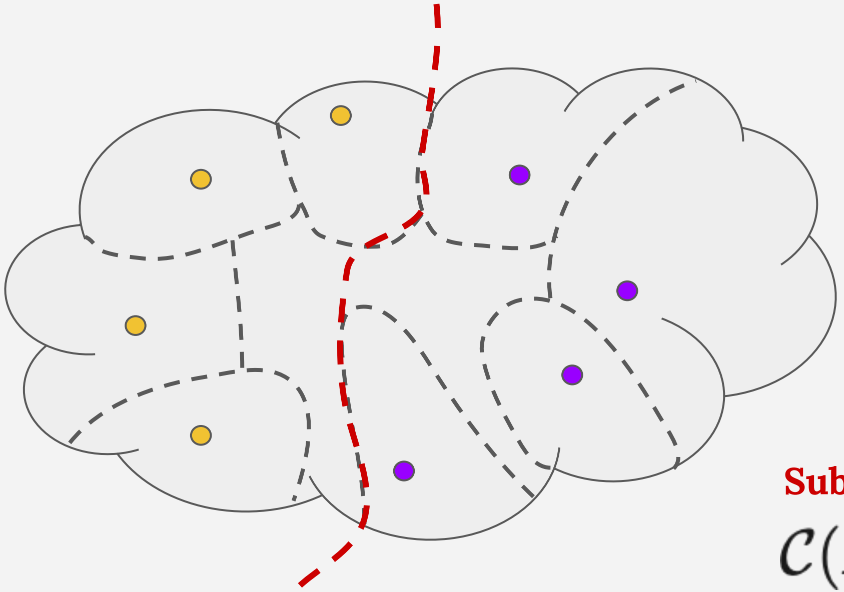
Uniqueness comes from submodularity.

**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Minimal Isolating Mincuts: Definitions Cont'd

An **A**-mincut of ***T*** is "minimal" if all proper subset containing **A** has a strictly larger cut boundary.



The minimal **A**-mincut **X** satisfies:

1. **X** contains **A** and **X** is disjoint from ***T*****\A**.
2. **C(X)** is minimum under 1.
3. **|X|** is minimum under 1. and 2.

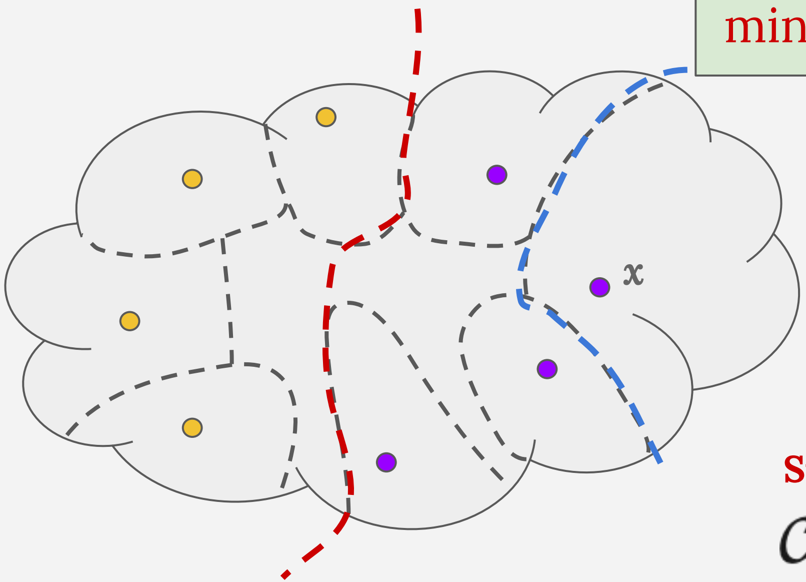
**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Computing Minimal Isolating Mincuts

Suppose we arbitrarily partition $\boldsymbol{T}$ into two halves $(\mathbf{A}, \mathbf{B})$, we can compute minimal $\mathbf{A}$-mincut of $\boldsymbol{T}$ and minimal $\mathbf{B}$-mincut of $\boldsymbol{T}$.
Again, using submodularity, we can show that

minimal $\boldsymbol{x}$-mincut $\subseteq$ minimal $\mathbf{A}$-mincut for all $\boldsymbol{x}$ in $\mathbf{A}$.



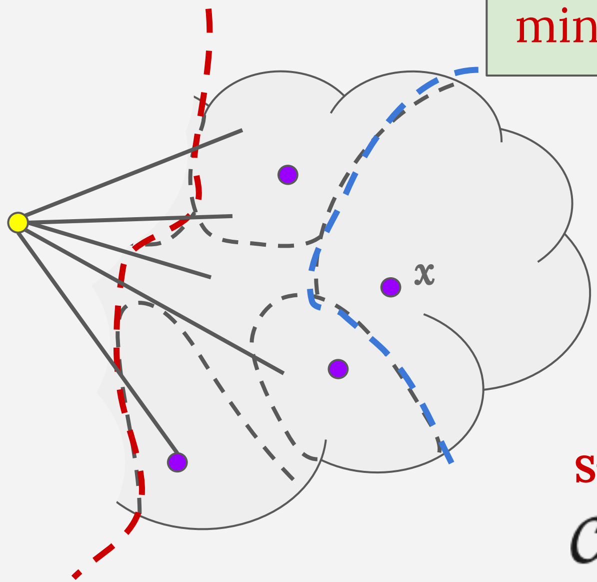

**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Computing Minimal Isolating Mincuts

Suppose we arbitrarily partition $T$ into two halves $(\mathbf{A}, \mathbf{B})$, we can compute minimal $\mathbf{A}$-mincut of $T$ and minimal $\mathbf{B}$-mincut of $T$.
Again, using submodularity, we can show that

minimal $x$-mincut $\subseteq$ minimal $\mathbf{A}$-mincut for all $x$ in $\mathbf{A}$.




**Contract & Recurse!**

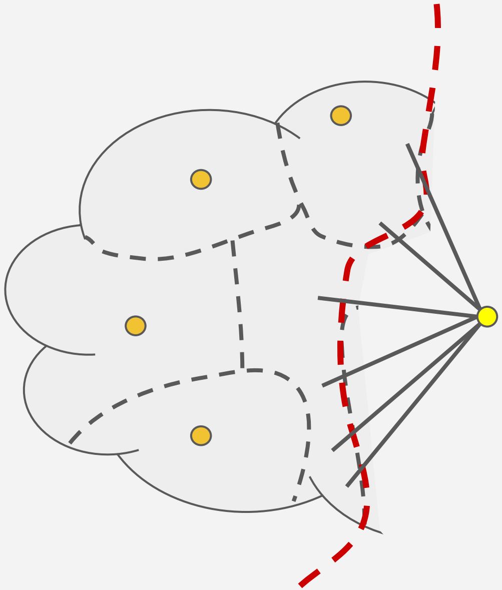
**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Computing Minimal Isolating Mincuts

Suppose we arbitrarily partition $\boldsymbol{T}$ into two halves $(\mathbf{A}, \mathbf{B})$, we can compute minimal $\mathbf{A}$-mincut of $\boldsymbol{T}$ and minimal $\mathbf{B}$-mincut of $\boldsymbol{T}$.

Again, using submodularity, we can show that



minimal $\boldsymbol{x}$-mincut $\subseteq$ minimal $\mathbf{A}$-mincut for all $\boldsymbol{x}$ in $\mathbf{A}$.

**Contract & Recurse!**

**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Outline


2 **Minimal Isolating Mincuts**

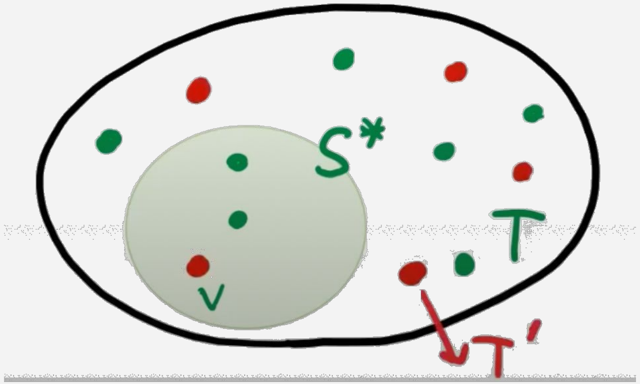
2.1 Definition

2.2 **Simple Application: Steiner Mincut**

# Steiner Mincuts

A **T-mincut** is a “partition” of $T$ that has a minimum possible valued cut among all non-trivial partition of $T$.

**Problem:** Find a $T$-mincut **S**.



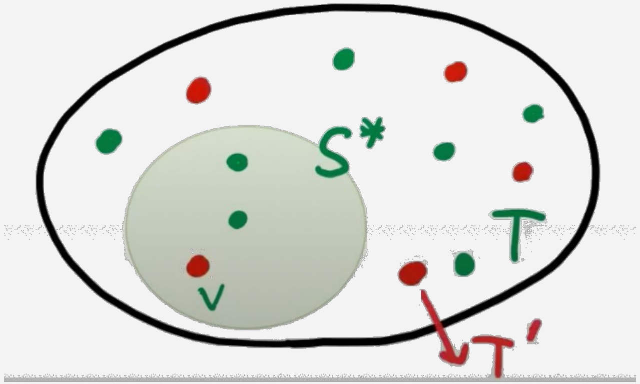
# Steiner Mincuts

A **T-mincut** is a "partition" of ***T*** that has a minimum possible valued cut among all non-trivial partition of ***T***.

**Problem:** Find a ***T***-mincut **S**.

**Observe:** For any fixed $a \in T$, **S** must separate $a$ from some $b \in T$.

We can use **|T|** max-flows to solve the problem.



# Steiner Mincuts

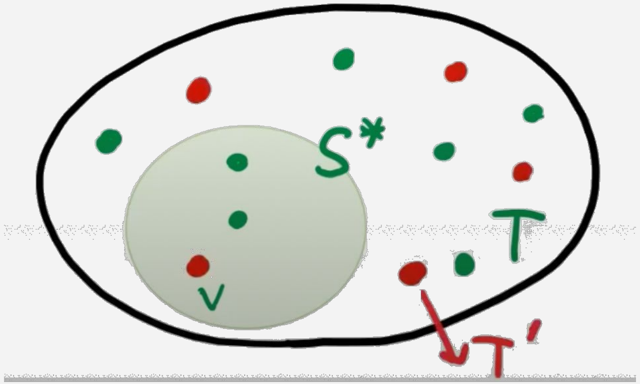
A **T-mincut** is a "partition" of ***T*** that has a minimum possible valued cut among all non-trivial partition of ***T***.

**Problem:** Find a ***T***-mincut **S**.

**Observe:** For any fixed $a \in T$, **S** must separate $a$ from some $b \in T$.

We can use **|T|** max-flows to solve the problem.

**Will show:** $\log^3 n$ max-flows via isolating cuts.

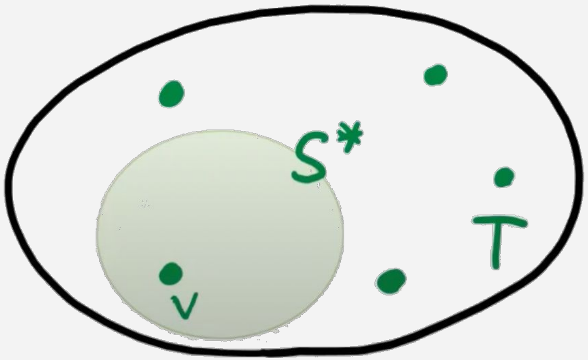



For more applications, see TCS+ Talk by Thatchaphol Saranurak.

# Steiner Mincuts: Easy Case

Let $S^*$ be some Steiner mincut. (Assume $|S^* \cap T| \leq |T \setminus S^*|$)

**Easy Case:** Suppose $|S^* \cap T| = 1$.

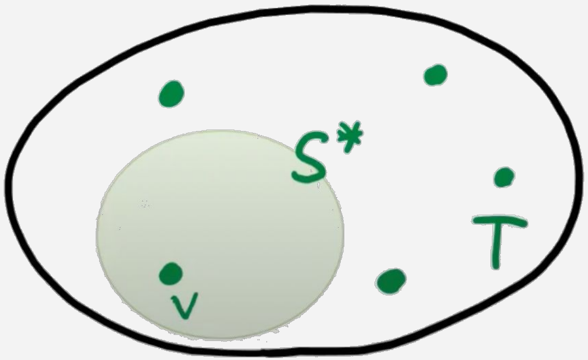


# Steiner Mincuts: Easy Case

Let $S^*$ be some Steiner mincut. (Assume $|S^* \cap T| \leq |T \setminus S^*|$)

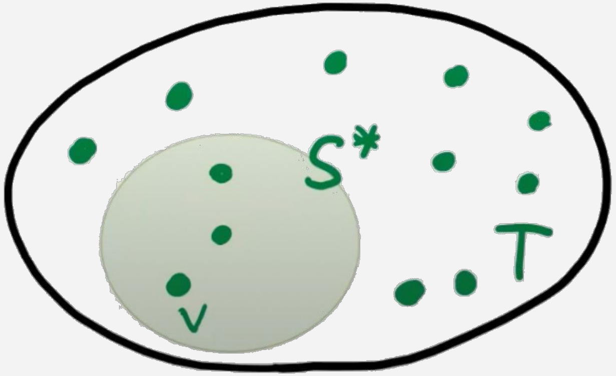
**Easy Case:** Suppose $|S^* \cap T| = 1$.

**Algo:** Just call IsoCut($T$).



# Steiner Mincuts: General Case

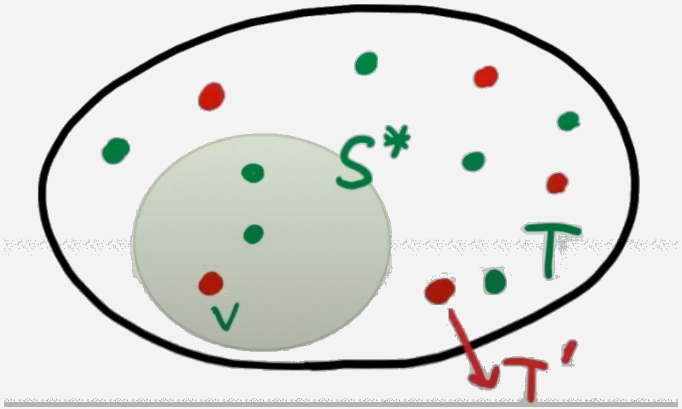
**General Case:** Suppose $|S^* \cap T| \in [2^i, 2^{i+1}]$.



# Steiner Mincuts: General Case

**General Case:** Suppose $|S^* \cap T| \in [2^i, 2^{i+1}]$.

$T' = \text{Sample}(T, 1/2^i)$ (i.e. sample each $t$ independently with prob $1/2^i$)



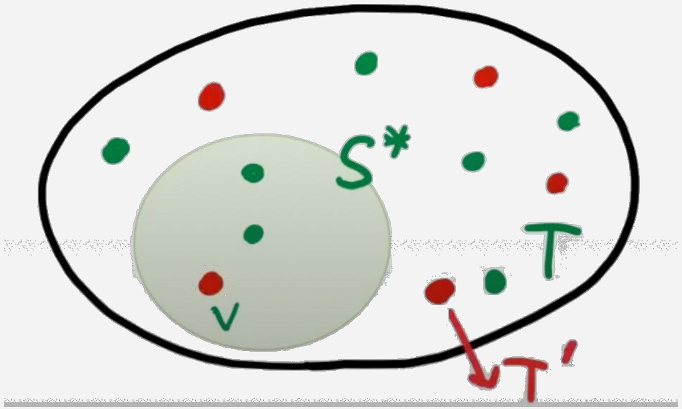
# Steiner Mincuts: General Case

**General Case:** Suppose $|S^* \cap T| \in [2^i, 2^{i+1}]$.

$T' = \mathrm{Sample}(T, 1/2^i)$ (i.e. sample each $t$ independently with prob $1/2^i$)

$|S^* \cap T'| = 1$ with prob $\Omega(1)$.

If happens, this is the **easy case**! Just call IsoCut($T'$).



# Steiner Mincuts: General Case

**General Case:** Suppose $|S^* \cap T| \in [2^i, 2^{i+1}]$.

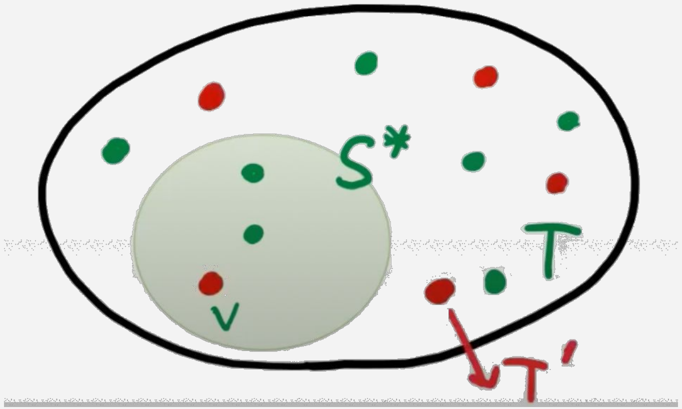
$T' = \text{Sample}(T, 1/2^i)$ (i.e. sample each $t$ independently with prob $1/2^i$)

$|S^* \cap T'| = 1$ with prob $\Omega(1)$.

If happens, this is the **easy case**! Just call IsoCut($T'$).

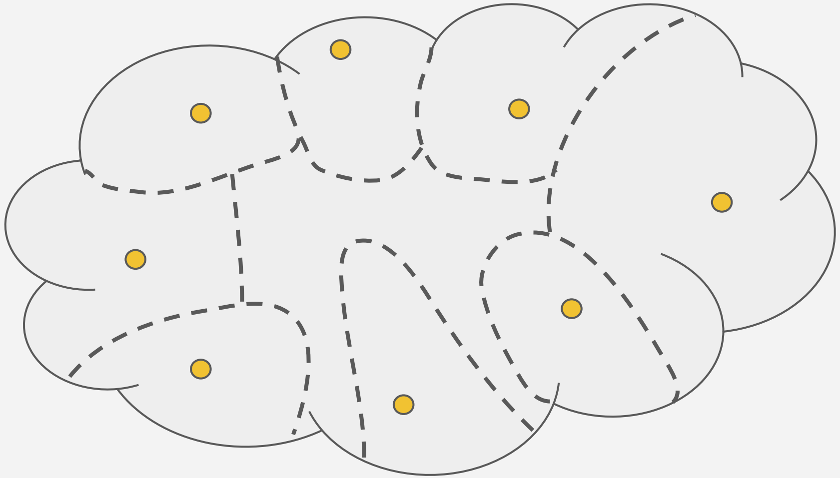
**Algorithm:**

- For $i = 0, \ldots, \log|T|$: (guess the size of $|S^* \cap T|$)
  - Repeat $O(\log n)$ times
    - $T' = \text{Sample}(T, 1/2^i)$
    - Call IsoCut($T'$)
- Return min-weight cut among all calls to IsoCut($\cdot$)



# Maximal Isolating Mincuts [**He**, Huang, Saranurak 2024]

An $\boldsymbol{x}$-mincut of $\boldsymbol{T}$ is "**maximal**" if all proper superset containing $\boldsymbol{x}$ has a strictly larger cut value.



**Maximal Isolating Cuts** [**He**, Huang, Saranurak 2024]

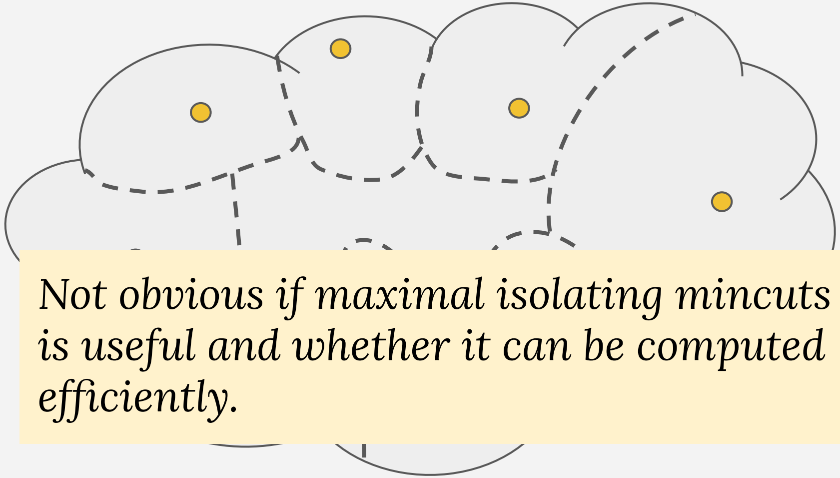
Given a terminal set $\boldsymbol{T}$, there is an algorithm that computes the **maximal** isolating mincuts for $\boldsymbol{T}$ using $O(\log |T|)$ max-flows.

**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Maximal Isolating Mincuts [**He**, Huang, Saranurak 2024]

An $\boldsymbol{x}$-mincut of $\boldsymbol{T}$ is "**maximal**" if all proper superset containing $\boldsymbol{x}$ has a strictly larger cut value.



*Not obvious if maximal isolating mincuts is useful and whether it can be computed efficiently.*

**Maximal Isolating Cuts** [**He**, Huang, Saranurak 2024]

Given a terminal set $\boldsymbol{T}$, there is an algorithm that computes the **maximal** isolating mincuts for $\boldsymbol{T}$ using $O(\log |T|)$ max-flows.

**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Outline

1. Divide and Conquer Framework to Compute the Cactus Representation

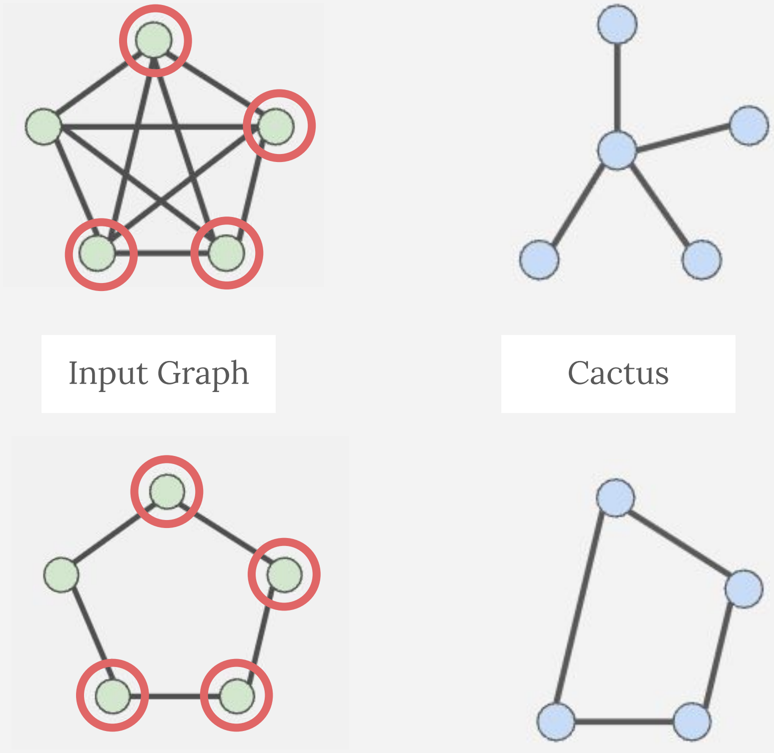
2. Minimal Isolating Mincuts

3. **Why Maximal Isolating Mincuts (Novel Variant)**

4. Compute Maximal Isolating Mincuts

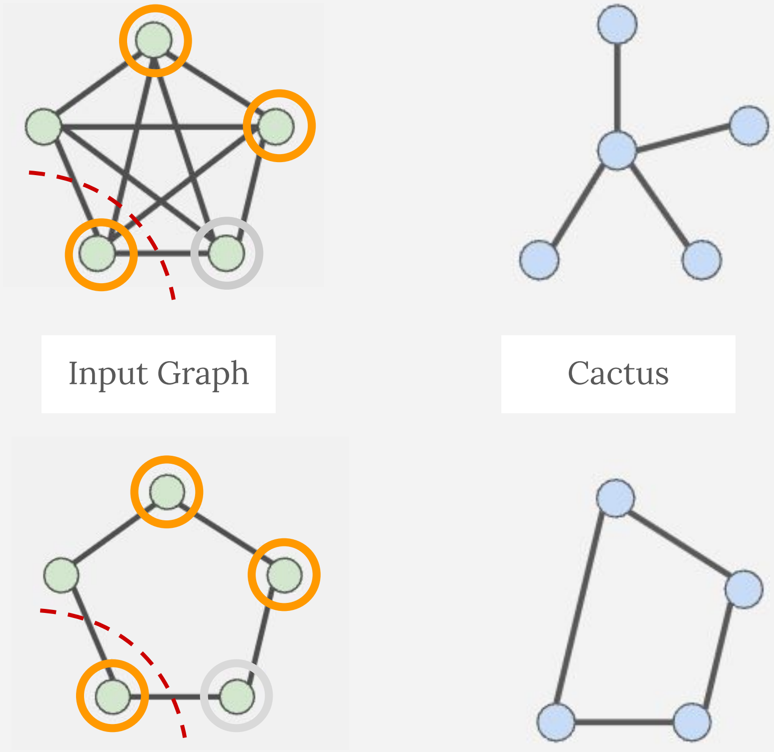
# Indistinguishability

If we use the tools that computes only **minimal isolating mincuts** (with sub-sampling), we cannot distinguish between a complete graph and a cycle.



# Indistinguishability

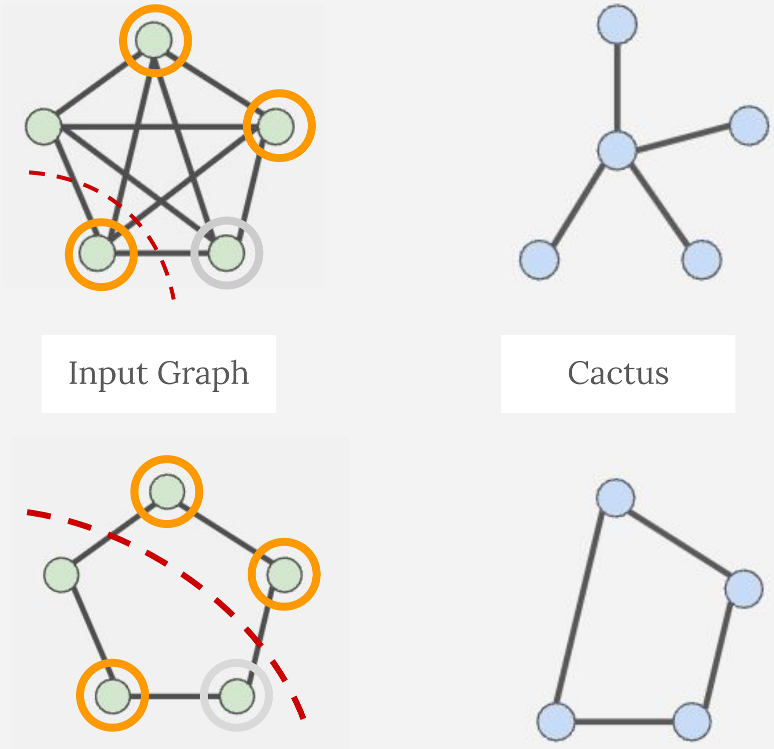
If we use the tools that computes only **minimal isolating mincuts** (with **sub-sampling**), we cannot distinguish between a complete graph and a cycle.



# Indistinguishability

If we use the tools that computes only **minimal isolating mincuts**, (with sub-sampling), we cannot distinguish between a complete graph and a cycle.

We can distinguish these two cases with **maximal isolating mincuts**!

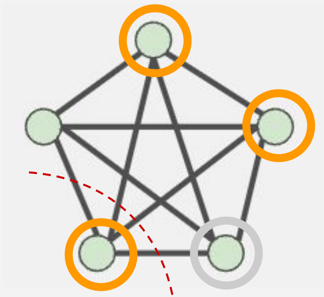


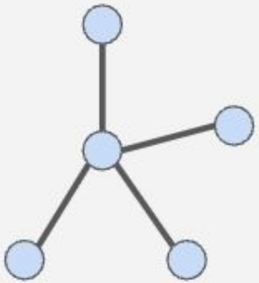
# Indistinguishability

If we use the tools that computes only **minimal isolating mincuts**, (with sub-sampling), we cannot distinguish between a complete graph and a cycle.

We can distinguish these two cases with **maximal isolating mincuts**!

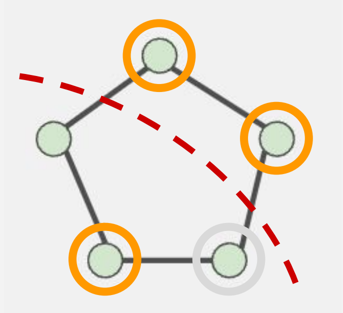
Moreover, we can find “balanced” cuts which makes the divide and conquer more efficient!

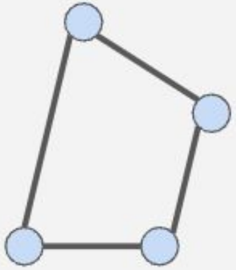




Input Graph

Cactus



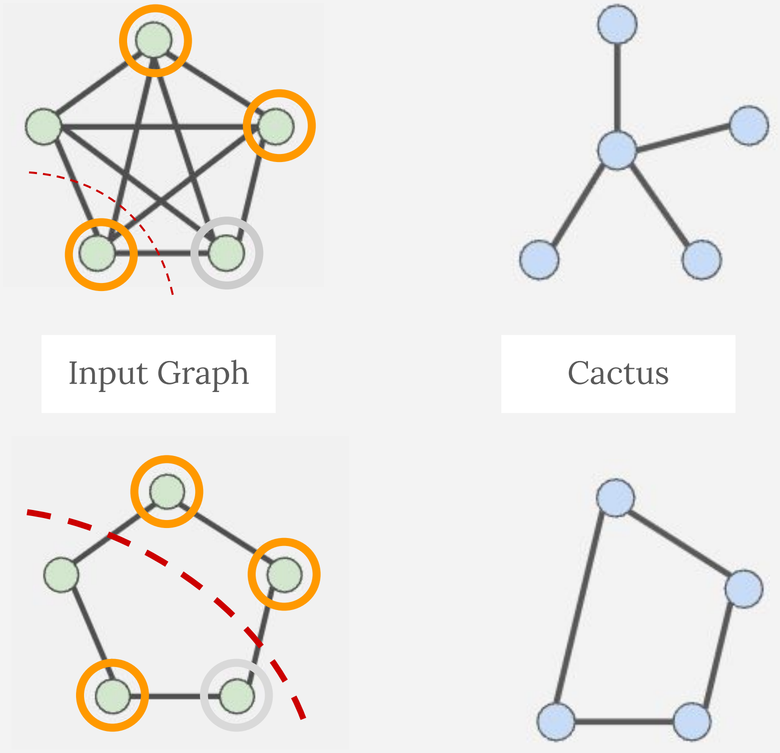


# Indistinguishability

If we use the tools that computes only **minimal isolating mincuts**, (with sub-sampling), we cannot distinguish between a complete graph and a cycle.

We can distinguish these two cases with **maximal isolating mincuts**!

Moreover, we can find “balanced” cuts which makes the divide and conquer more efficient!



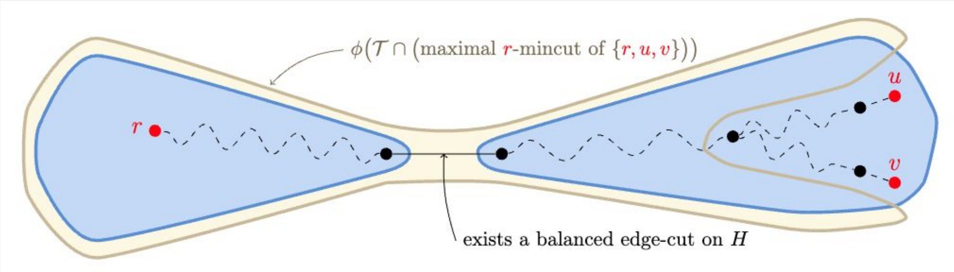
# Maximal Isolating Mincuts + Sample Terminals ⇒ D&C

Consider the underlying cactus representation.

Is there a "balanced" edge-cut in the current cactus?

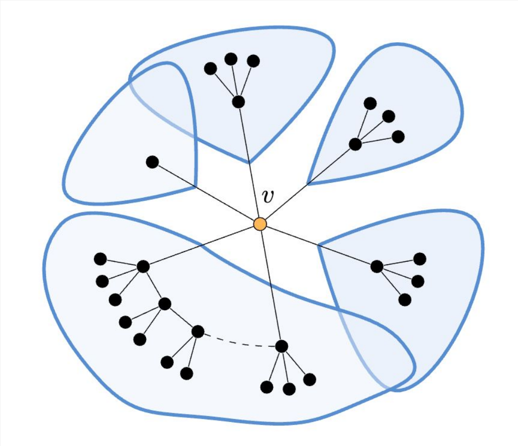
Yes

Both sides contain at least $|T|/4$ terminals.




No

There exists a centroid $v$.

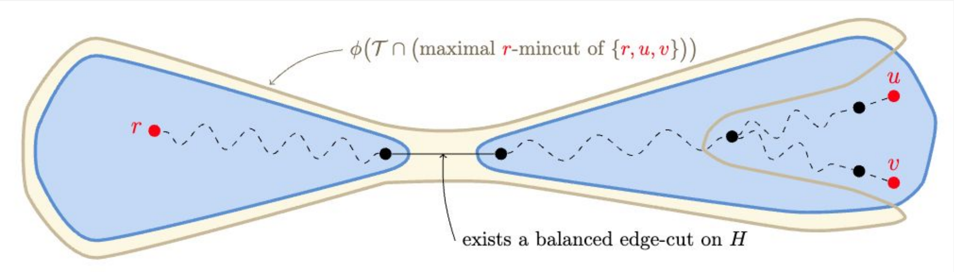


# Maximal Isolating Mincuts + Sample 3 Terminals ⇒ Balanced D&C

Is there a "balanced" edge-cut in the current cactus?

Yes

Both sides contain at least $|T|/4$ terminals.




The set (yellow) $\mathbf{X}$ corresponds to maximal $\boldsymbol{r}$-mincut.

$\boldsymbol{u}, \boldsymbol{v}$ are uniformly sampled. With constant probability $\mathbf{V} \setminus \mathbf{X}$ contains at least $|\mathbf{T}|/4$ terminals.

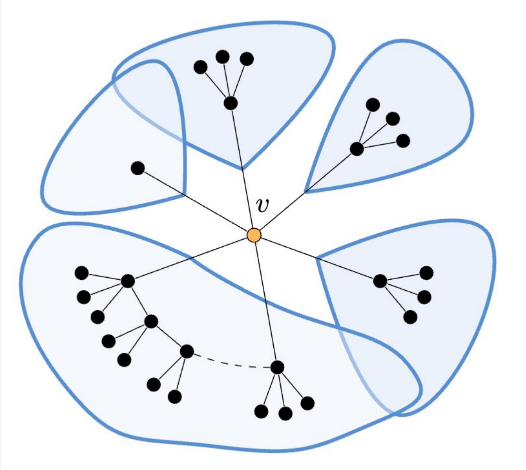
# Maximal Isolating Mincuts + Sample 3 Terminals ⇒ Balanced D&C

Is there a "balanced" edge-cut in the current cactus?

No

There exists a centroid $v$.

Want to obtain a set of cuts, corresponds to each of the blue sets.



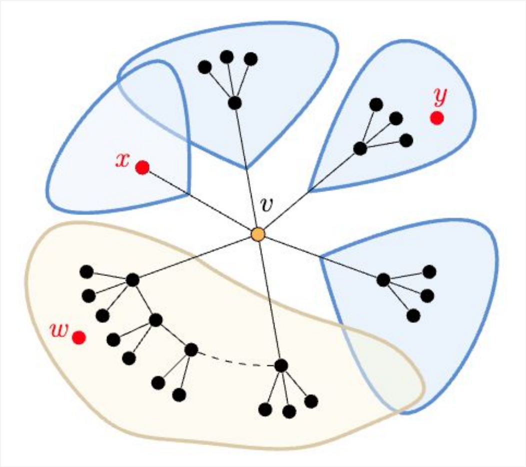
# Maximal Isolating Mincuts + Sample Terminals ⇒ Centroid D&C

Is there a "balanced" edge-cut in the current cactus?

For each set $T_i$, sample each terminal with probability $2^{-\lceil \log |T_i| \rceil}$.

No

There exists a centroid $\boldsymbol{v}$.



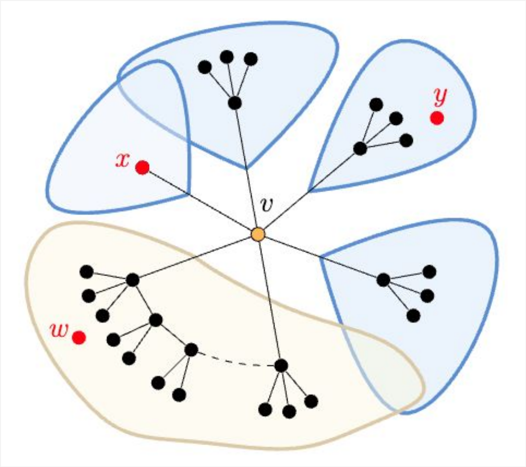
# Maximal Isolating Mincuts + Sample Terminals ⇒ Centroid D&C

Is there a "balanced" edge-cut in the current cactus?

No

There exists a centroid $\boldsymbol{v}$.

For each set $T_i$, sample each terminal with probability $2^{-\lceil \log |T_i| \rceil}$.
With constant probability, **exactly one** terminal in $T_i$ is sampled, together with **at least one** terminal from **any two other** subsets being sampled.



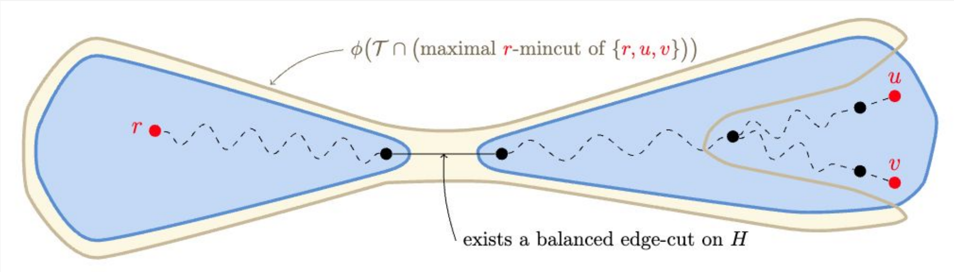
# Maximal Isolating Mincuts + Sample Terminals ⇒ D&C

Consider the underlying cactus representation.

Is there a "balanced" edge-cut in the current cactus?

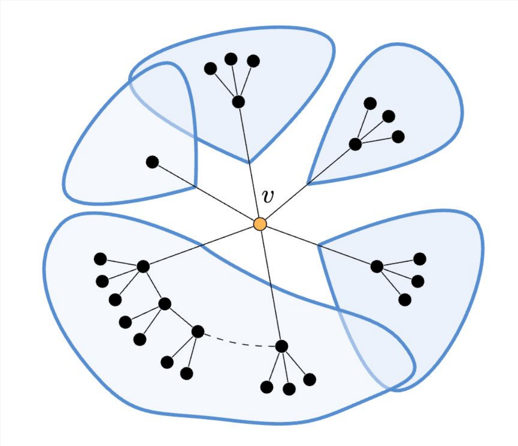
Yes

Both sides contain at least $|T|/4$ terminals.




No

There exists a centroid $v$.



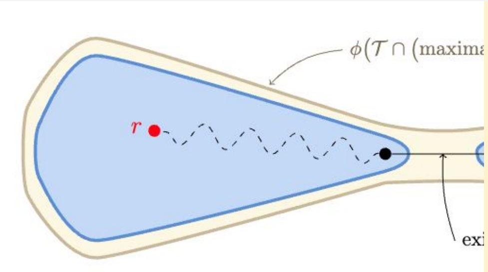
# Maximal Isolating Mincuts + Sample Terminals ⇒ D&C

Consider the underlying cactus representation.

Is there a “balanced” edge-cut in the current cactus?

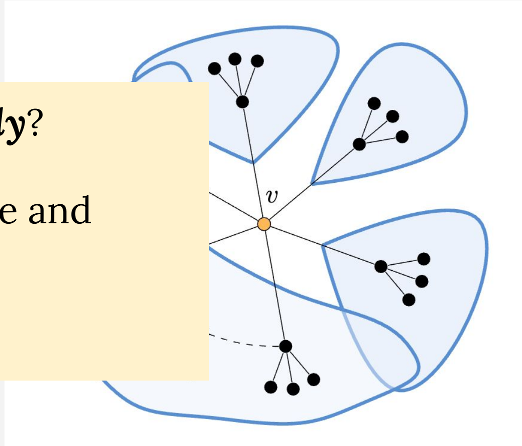
Yes

Both sides contain at least $|T|/4$ terminals.




No

There exists a centroid $v$.




**Q1**: How do we find **T**-splits ***efficiently***?

**Q2**: Can we bound the **depth** of divide and conquer?

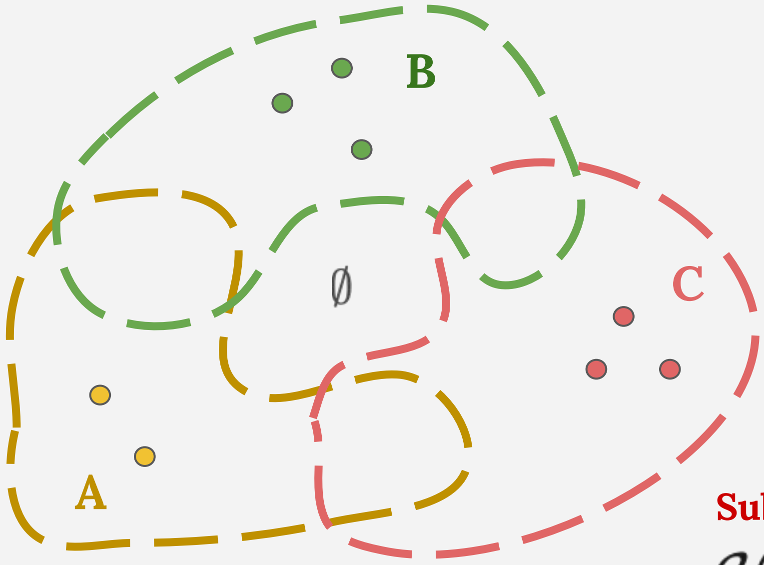
**Both challenges solved!**

# Outline

1. Divide and Conquer Framework to Compute the Cactus Representation
2. Minimal Isolating Mincuts
3. Why Maximal Isolating Mincuts (Novel Variant)
4. **Compute Maximal Isolating Mincuts**

# Posi-Modularity & Pairwise Intersection Only Lemma

"3-Star Lemma"
[Dinitz and Vainshtein 1994]




**Theorem** [Dinitz and Vainshtein 1994]

Let A, B, and C be disjoint sets of terminals. Then, the intersection of {any A-mincut, any B-mincut, and any C-mincut} is **empty**.

**Submodularity** for "cut values":

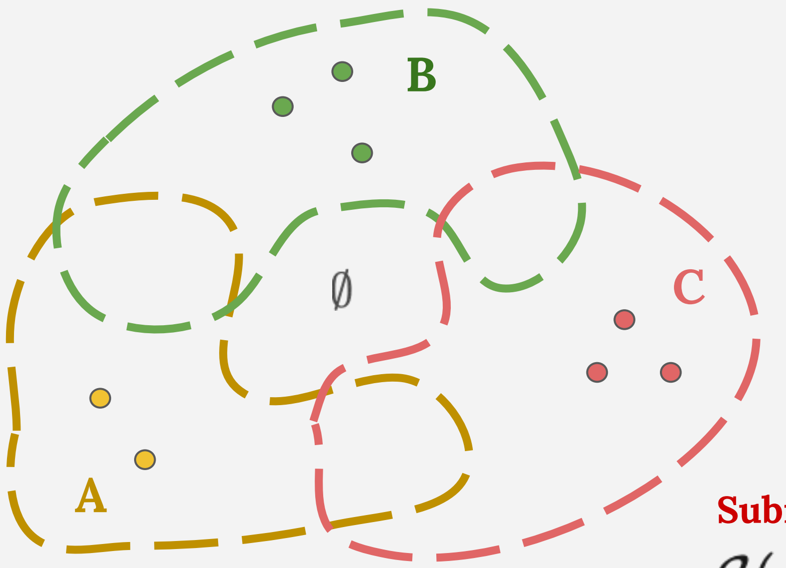
$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

**Posi-modularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \setminus Y) + \mathcal{C}(Y \setminus X)$$

“3-Star Lemma”
[Dinitz and Vainshtein 1994]

# Posi-Modularity & Pairwise Intersection Only Lemma




**Theorem** [Dinitz and Vainshtein 1994]

Let A, B, and C be disjoint sets of terminals. Then, the intersection of {any A-mincut, any B-mincut, and any C-mincut} is **empty**.

Each vertex appears in **at most 2** maximal isolating mincuts!

**Submodularity** for “cut values”:

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

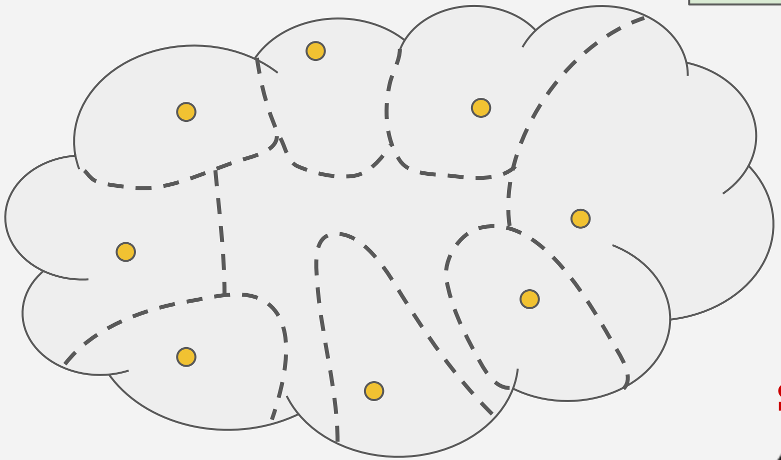
**Posi-modularity** for “cut values”:

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \setminus Y) + \mathcal{C}(Y \setminus X)$$

# Computing Maximal Isolating Mincuts

Suppose we arbitrarily partition $T$ into two halves $(\mathbf{A}, \mathbf{B})$, and we can compute maximal $\mathbf{A}$-mincut of $T$ and maximal $\mathbf{B}$-mincut of $T$.
Again, using submodularity, we can show that

maximal $x$-mincut $\subseteq$ maximal $\mathbf{A}$-mincut for all $x$ in $\mathbf{A}$.

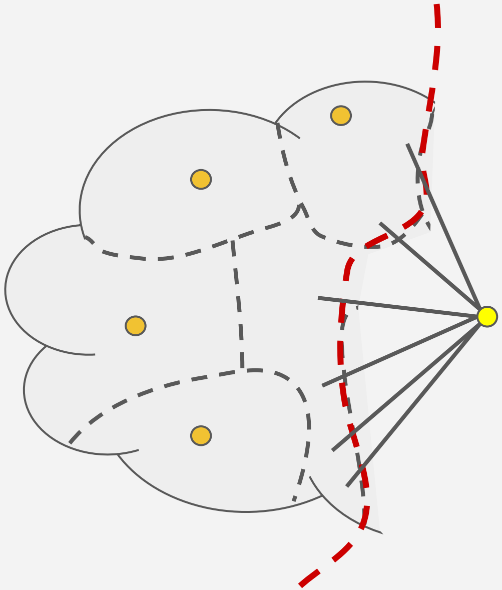


**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Computing Maximal Isolating Mincuts

Suppose we arbitrarily partition $T$ into two halves $(\mathbf{A}, \mathbf{B})$, and we can compute maximal $\mathbf{A}$-mincut of $T$ and maximal $\mathbf{B}$-mincut of $T$.
Again, using submodularity, we can show that



maximal $x$-mincut $\subseteq$ maximal $\mathbf{A}$-mincut for all $x$ in $\mathbf{A}$.

**Contract & Recurse!**

**Submodularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

# Posi-Modularity

**Submodularity** for "cut values":

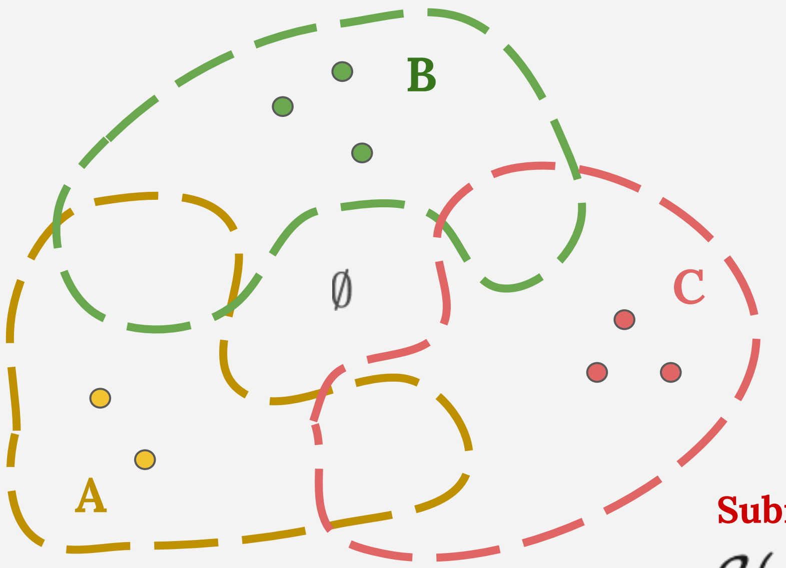
$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

**Posi-modularity** for "cut values":

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \setminus Y) + \mathcal{C}(Y \setminus X)$$

“3-Star Lemma”
[Dinitz and Vainshtein 1994]

# Posi-Modularity & Pairwise Intersection Only Lemma




**Theorem** [Dinitz and Vainshtein 1994]

Let A, B, and C be disjoint sets of terminals. Then, the intersection of {any A-mincut, any B-mincut, and any C-mincut} is **empty**.

Each vertex appears in **at most 2** recursion paths!

**Submodularity** for “cut values”:

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \cup Y) + \mathcal{C}(X \cap Y)$$

**Posi-modularity** for “cut values”:

$$\mathcal{C}(X) + \mathcal{C}(Y) \geq \mathcal{C}(X \setminus Y) + \mathcal{C}(Y \setminus X)$$

# Summary

| | | |
|---|---|---|
| Standard Cactus Representation | **Solved!** | $O(m \log^3 n)$ |
| Steiner Cactus Representation | **Solved!** | $O(m^{1+o(1)})$ |
| Hypergraph Cactus Representation | **Solved!** | $O(p^{1+o(1)})$ |
| Element Cut Cactus Representation | **Open** | |
| General submodular functions | **Open** | |

From mincuts to **near-mincuts**: Polygon Representations **Open**

## Thank you!

# Thank you!