Computing min-cuts in hypergraphs

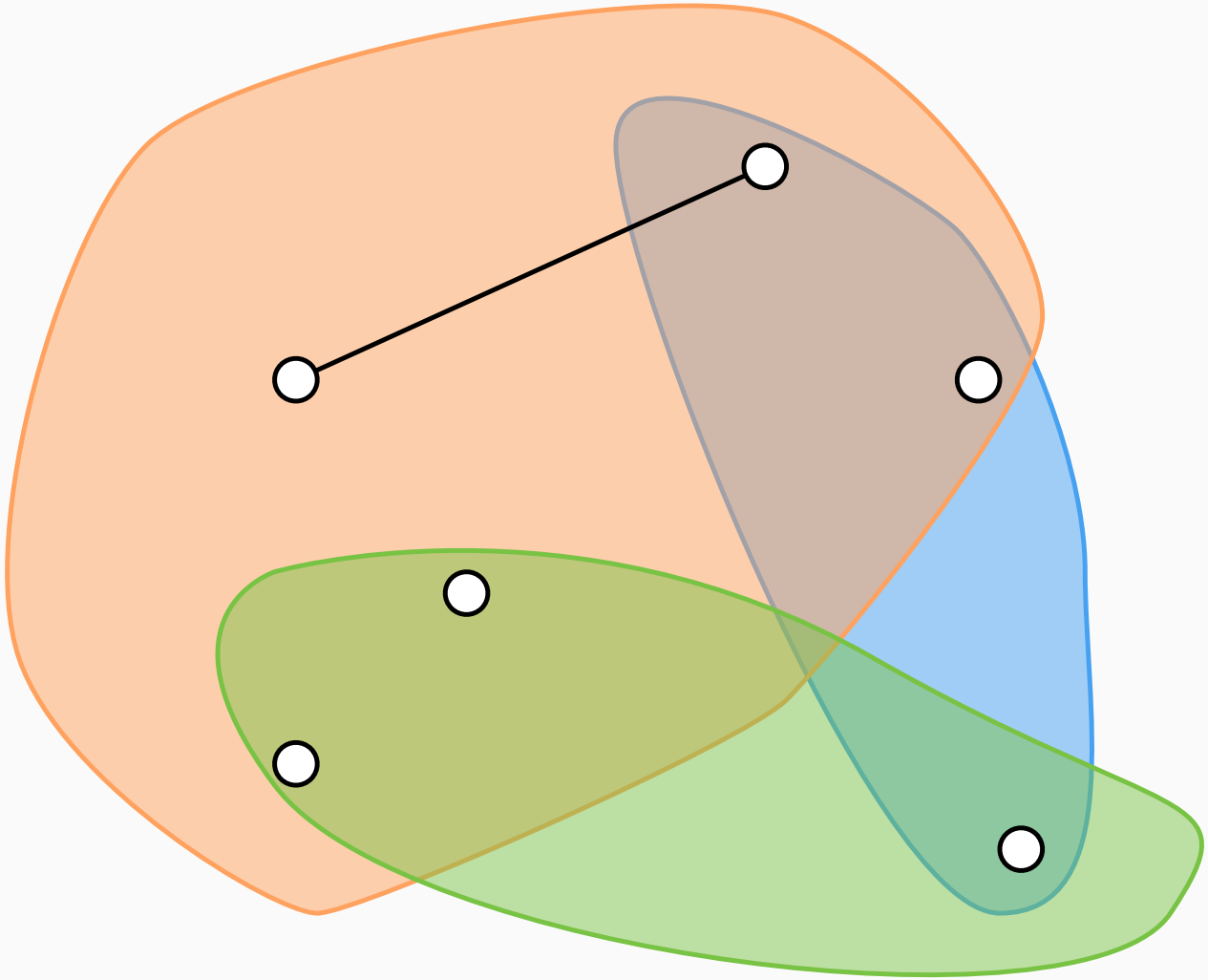
Chandra Chekuri and Chao Xu

January 17, 2017

University of Illinois, Urbana-Champaign

A hypergraph

A hypergraph $H = (V, E)$ consists of vertices V and edges E.

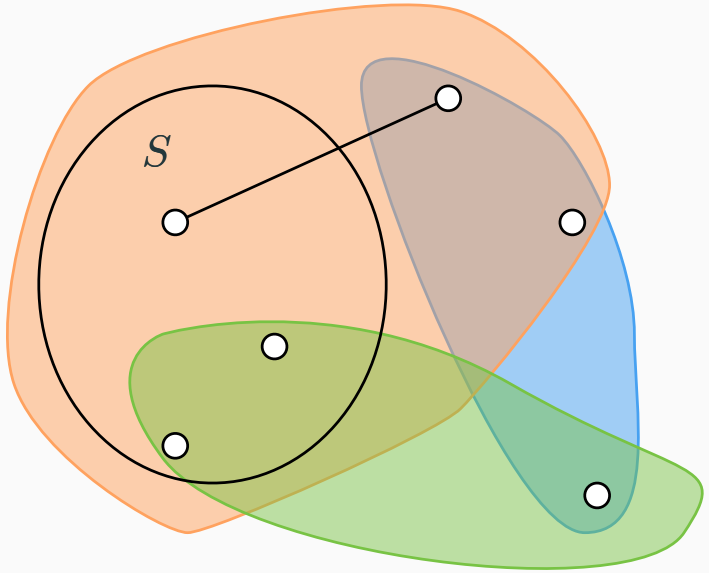


Cut function

- $\delta(S)$ is the set of all edges cross S, $w : E \to \mathbb{R}_+$ the weight function.
- The **cut function** $c : 2^V \to \mathbb{R}_+$

$$c(S) = \sum_{e \in \delta(S)} w(e)$$

- A min-cut is a cut with the smallest value.



Parameters

- n: # of vertices
- m: # of edges
- p: **size** of the hypergraph $\sum_{e \in E} |e|$.

Finding a min-cut

1. Graphs, deterministic: $O(nm + n^2 \log n)$ [Stoer and Wagner 1997]
2. Hypergraphs, deterministic: $O(np + n^2 \log n)$ [Klimmek and Wagner 1996, Queyranne 1998, Mak and Wong 2000].
3. Graphs, randomized: $\tilde{O}(m)$ [Karger 2000]
4. Hypergraphs, randomized: $\tilde{O}(n^2 p)$ [Ghaffari, Karger and Panigrahi 2017]

Our results

1. Sparse subgraph maintaining all small cuts.
   - size $O(kn)$ to preserve cuts of size k.
   - $O(p)$ time.
   - leads to $O(p + n^2\lambda)$ min-cut algorithm.
2. Find all min-cuts and hypercactus.
   - $O(np + n^2 \log n)$, same as finding one min-cut
3. $(2 + \epsilon)$-approximation for min-cut in near linear time.
   - $O(\epsilon^{-1}(p + n \log n) \log n)$.

A metatheorem?

- Generalization of Nagamochi-Ibaraki style vertex orderings.
- Results depending on vertex orderings can be lifted to hypergraphs.

k-certificate (unweighted)

k-certificate

Subgraph H' a k-**certificate** of H if it has $O(kn)$ edges and for all $S \subset V$

$$|\delta_{H'}(S)| \geq \min(|\delta_H(S)|, k),$$

k-certificate

Subgraph H' a k-**certificate** of H if it has $O(kn)$ edges and for all $S \subset V$

$$|\delta_{H'}(S)| \geq \min(|\delta_H(S)|, k),$$

Example: a 1-certificate of a graph is a maximal forest.

k-certificate

Subgraph H' a k-**certificate** of H if it has $O(kn)$ edges and for all $S \subset V$

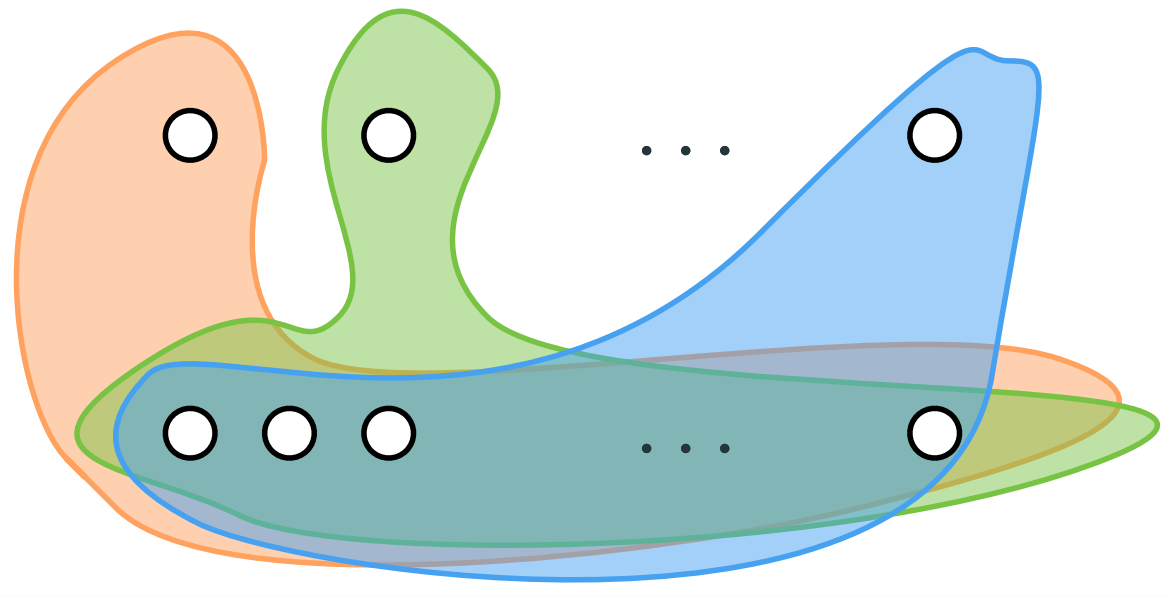
$$|\delta_{H'}(S)| \geq \min(|\delta_H(S)|, k),$$

Example: a 1-certificate of a graph is a maximal forest.

Every hypergraph has a k-certificate.

- Graph: packing k forests, $O(m)$ time. [Nagamochi and Ibaraki 1992]
- Hypergraph: packing k spanning subgraphs, $O(np)$ time. [Guha, McGregor and Tench 2015]

Edge deletion is not enough: large size



$k = 1$, size $\Omega(kn^2)$.

Small size k-certificate

Theorem

Every hypergraph has a k-certificate with size $O(kn)$, and can be found in $O(p)$ time.

Trimming

The **trimming** operation removes a vertex from an edge (removes the edge if it become a singleton).

Trimming

The **trimming** operation removes a vertex from an edge (removes the edge if it become a singleton).

Trimming and edge deletion are the same in graphs.

Trimming

Applications of trimming [Frank, Király, Kriesell 2003]

- k-partition-connected hypergraphs
- arborescence packing in directed hypergraphs

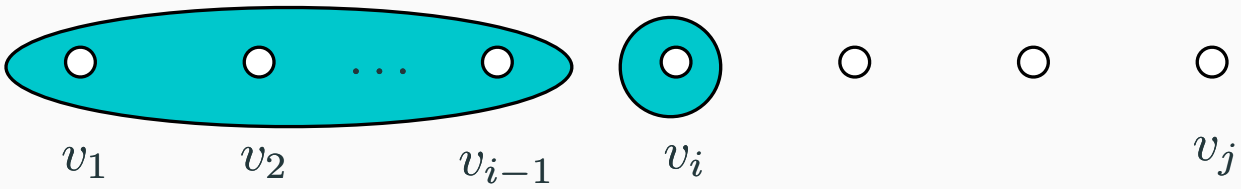
Adjacency

$$d(A, B) = \sum_{e \in \delta(A) \cap \delta(B)} w(e)$$

Adjacency

$$d(A, B) = \sum_{e \in \delta(A) \cap \delta(B)} w(e)$$

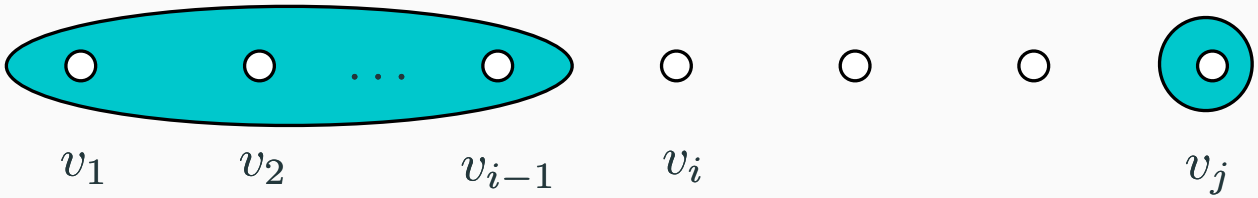
$v_1, \ldots, v_n$ is a maximum adjacency ordering(MA-ordering) if for all $1 \leq i \leq j \leq n$,



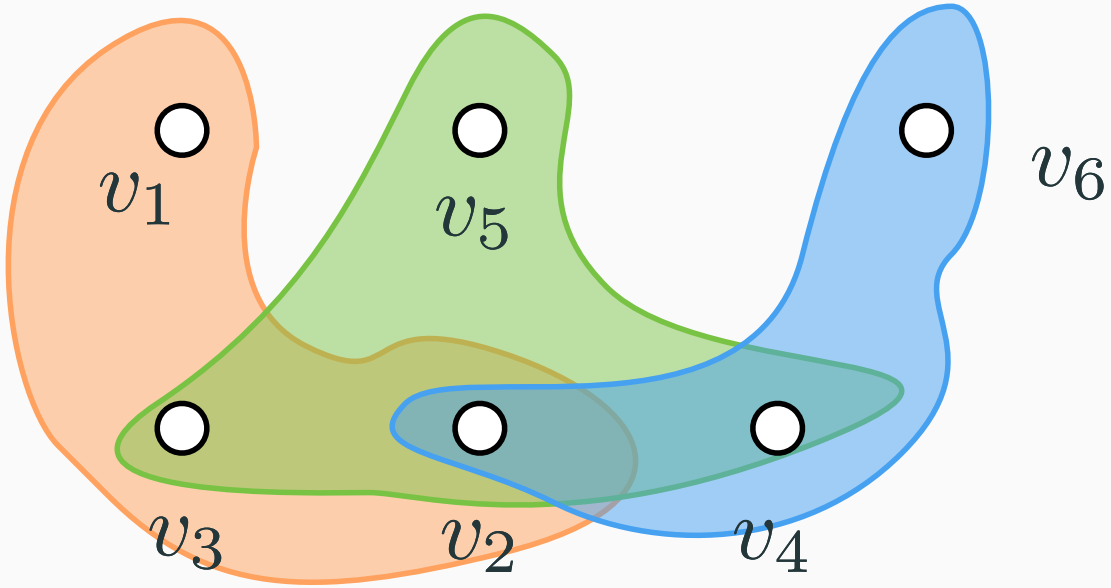
$$d(\{v_1, \ldots, v_{i-1}\}, v_i)$$

$$\geq$$

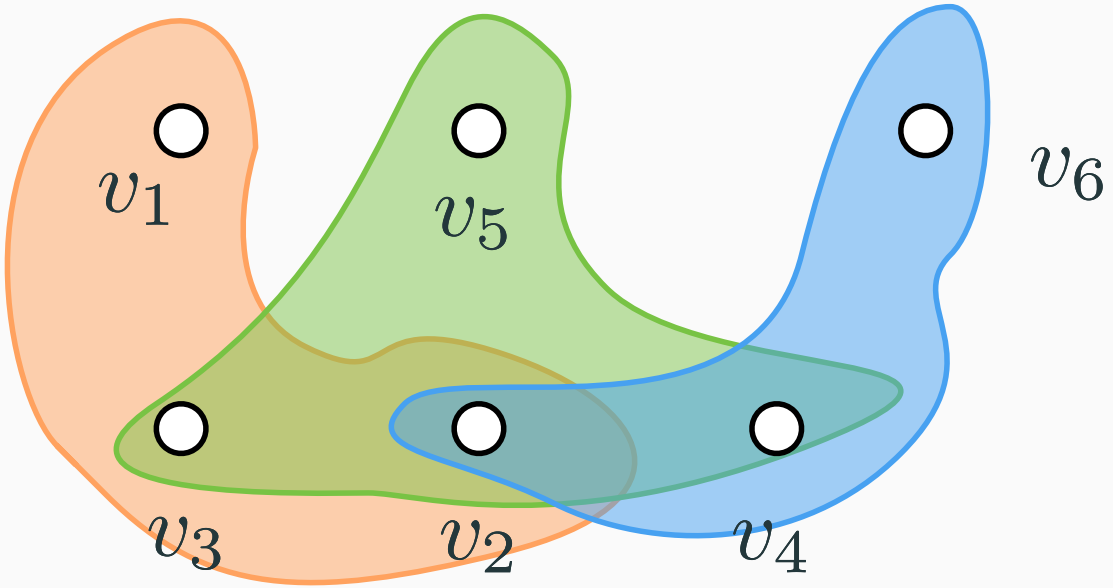
$$d(\{v_1, \ldots, v_{i-1}\}, v_j)$$

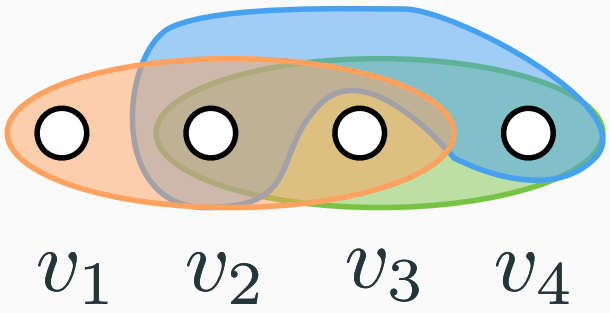


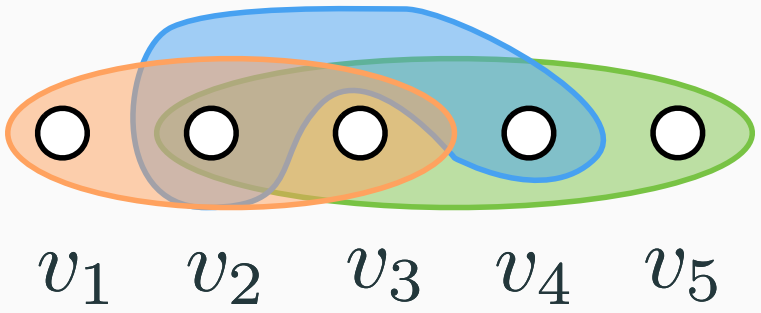
$$d(\{v_1, \ldots, v_{i-1}\}, v_i) \geq d(\{v_1, \ldots, v_{i-1}\}, v_j)$$



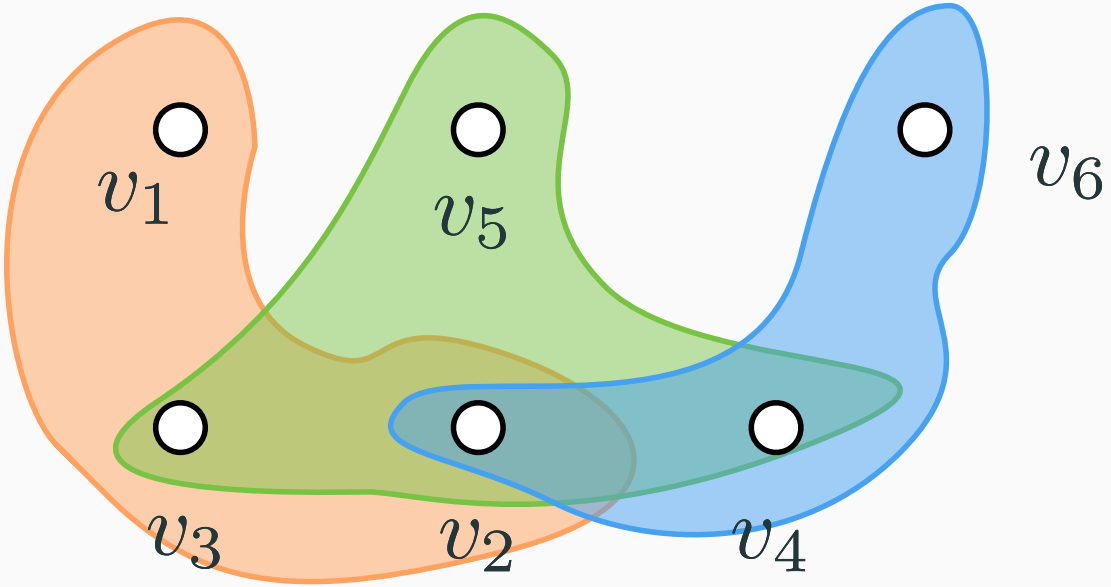
$$d(\{v_1, \ldots, v_{i-1}\}, v_i) \geq d(\{v_1, \ldots, v_{i-1}\}, v_j)$$

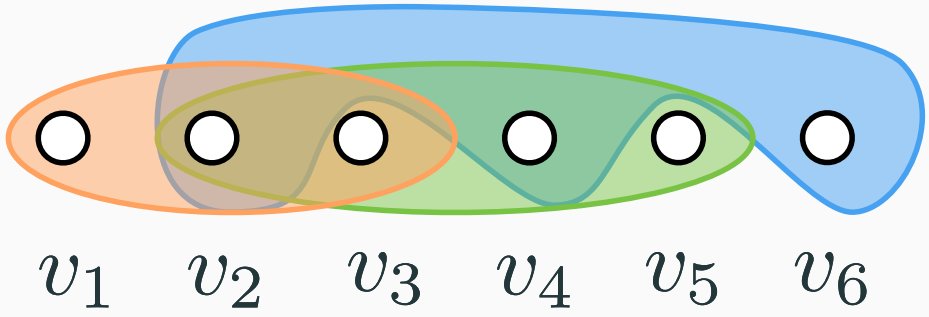






$$d(\{v_1, \ldots, v_{i-1}\}, v_i) \geq d(\{v_1, \ldots, v_{i-1}\}, v_j)$$





Find a k-certificate with small size

Order the edges by the smallest numbered vertex in the MA-ordering.

Find a k-certificate with small size

Order the edges by the smallest numbered vertex in the MA-ordering.

e is a **backedge** of $v \in e$ if v is not the smallest numbered vertex in e.

Find a k-certificate with small size

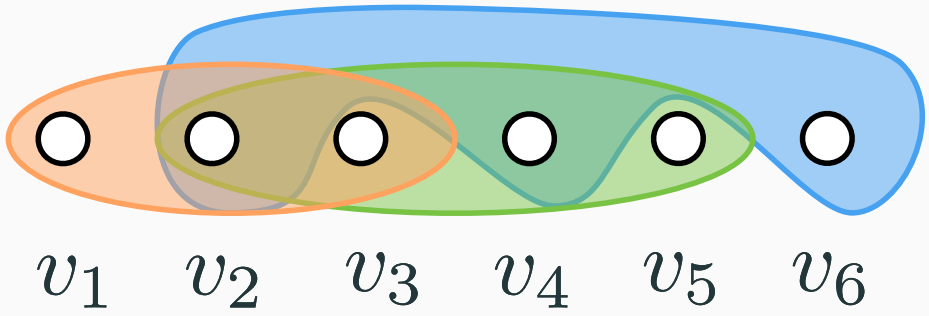
Order the edges by the smallest numbered vertex in the MA-ordering.

e is a **backedge** of $v \in e$ if v is not the smallest numbered vertex in e.

Algorithm: For each vertex, trim it from all but the first k backedges.

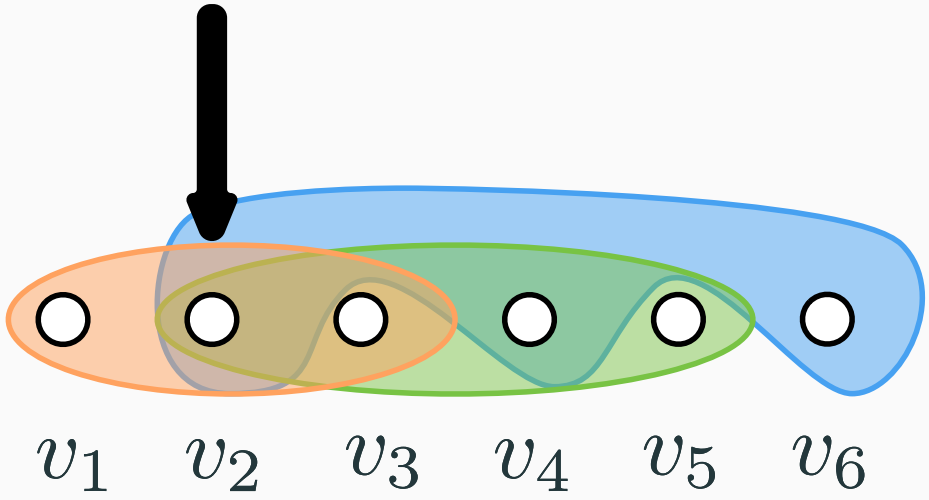
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



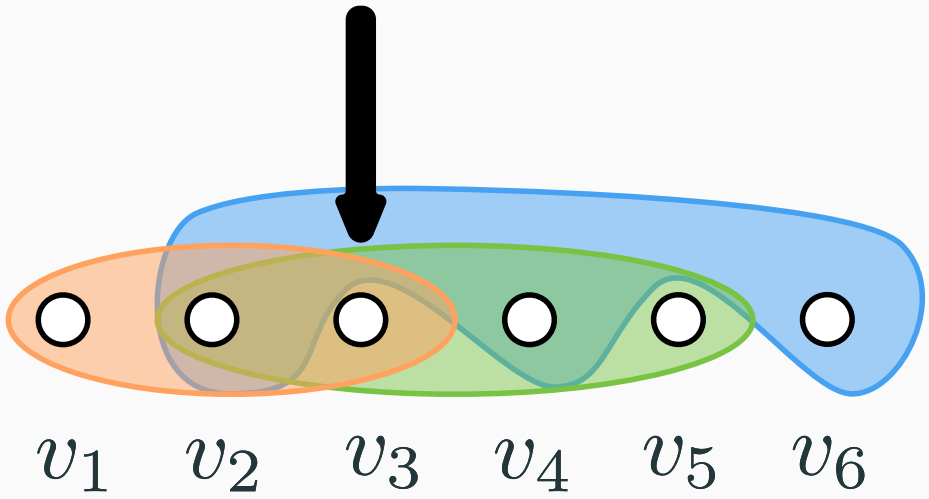
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



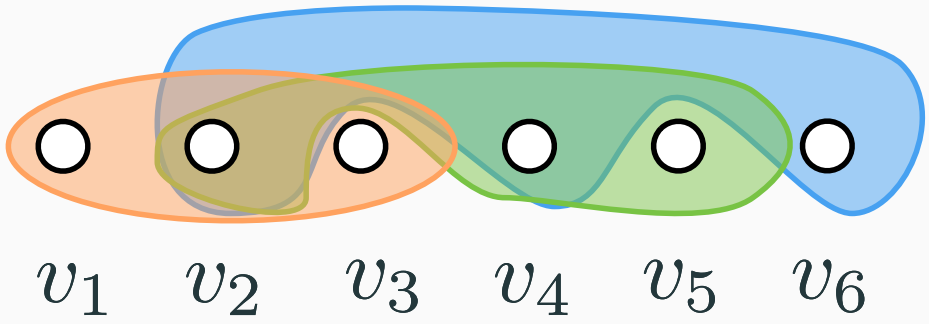
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



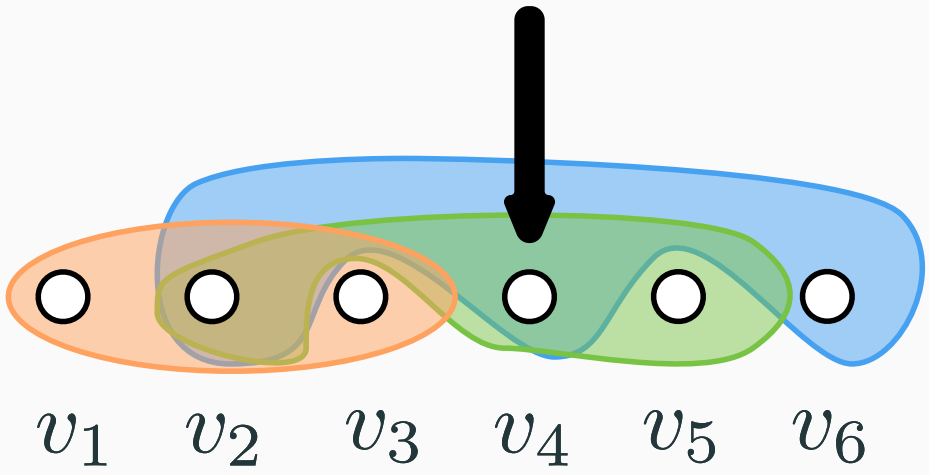
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



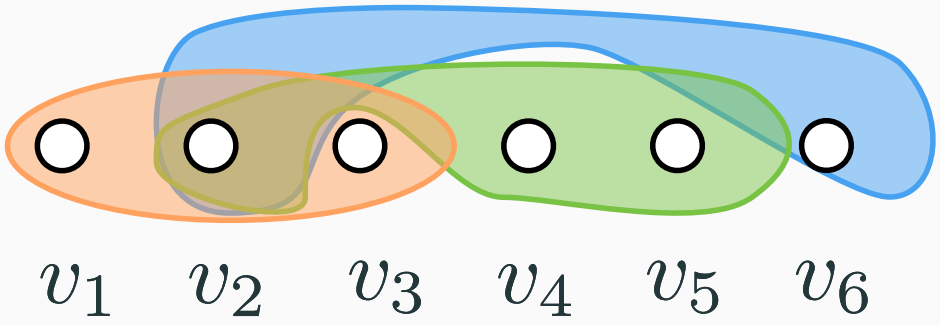
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



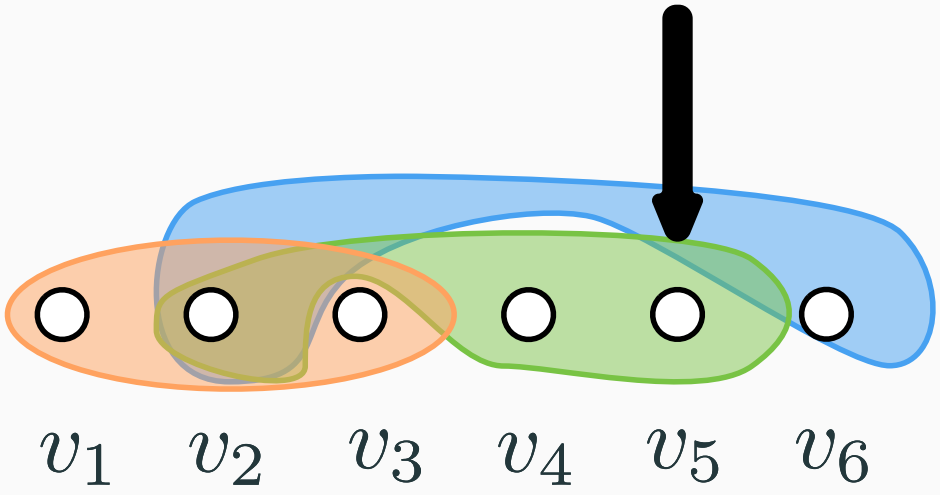
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



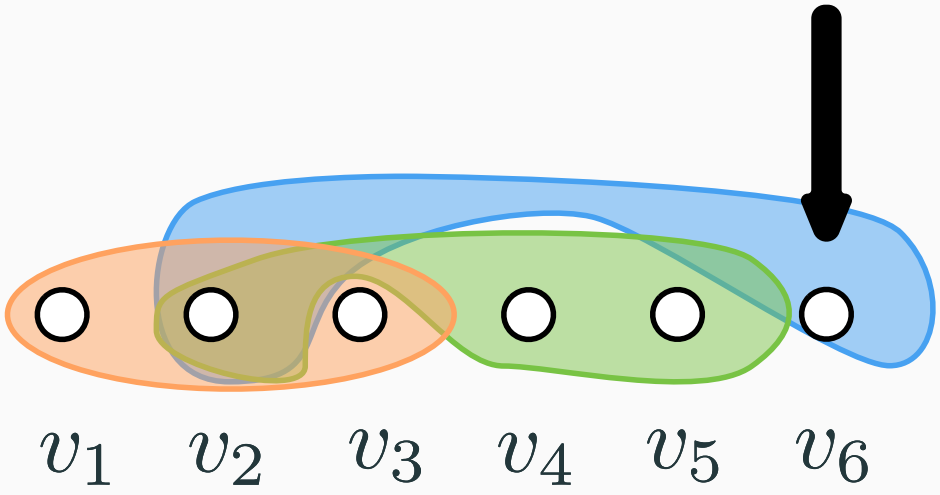
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



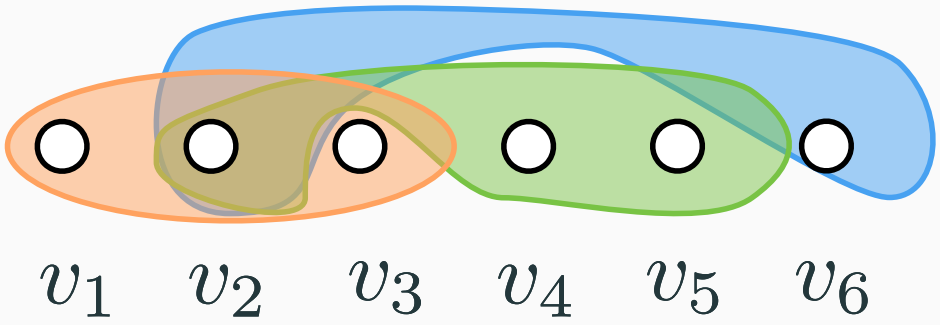
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



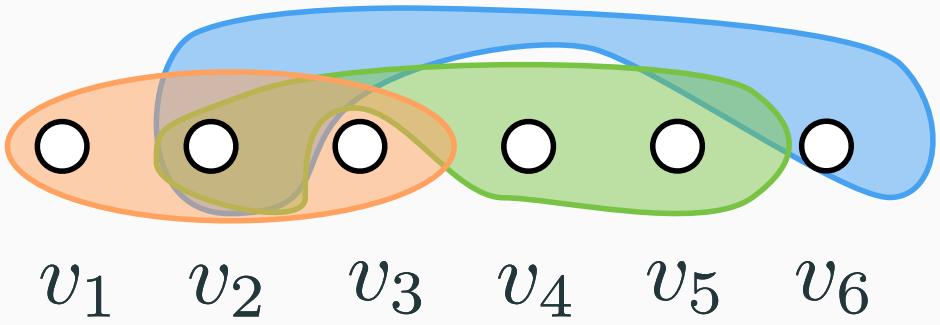
Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



Example: Find a 1-certificate

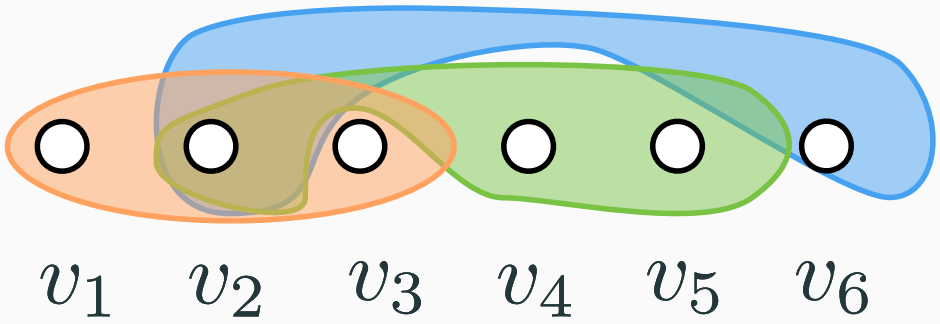
Algorithm: For each vertex, trim it from all but the first k backedges.



v_i, $i > 1$ appear in at most k backedges. Charge the remaining appearance to number of edges.

Example: Find a 1-certificate

Algorithm: For each vertex, trim it from all but the first k backedges.



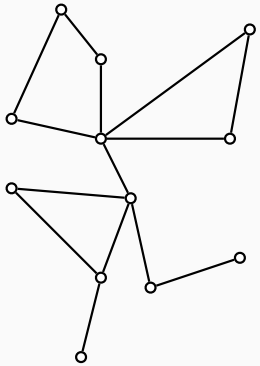
v_i, $i > 1$ appear in at most k backedges. Charge the remaining appearance to number of edges.

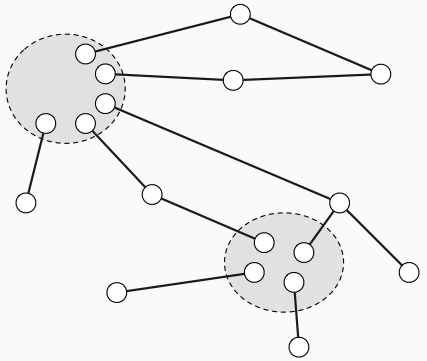
$\#$ edges $\leq k(n-1)$, size $\leq 2k(n-1)$.

Applications

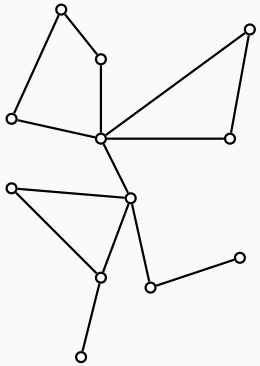
- Find a min-cut in $O(p + \lambda n^2)$ time.
- Speed up s-t connectivity computation.
- Speed up α-approximate min-cuts computation.

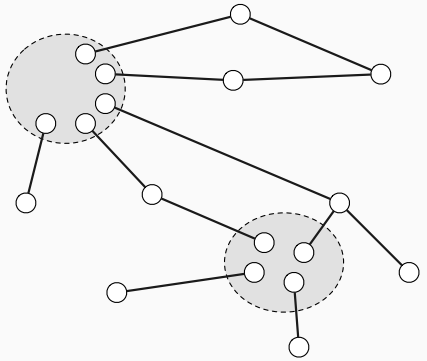
All min-cuts





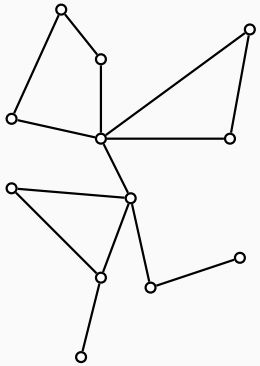
Cactus: compact representation of all min-cuts in graph. [Karzanov and Timofeev 1986]

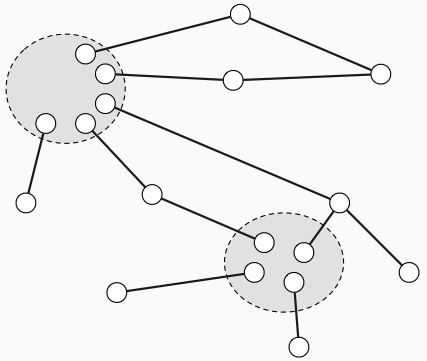




Cactus: compact representation of all min-cuts in graph. [Karzanov and Timofeev 1986]

- Deterministic: $O(nm + n^2 \log n)$ running time. [Nagamochi, Nakamura, Ishii 2003]
- Randomized: $\tilde{O}(m)$ running time. [Karger 2009]

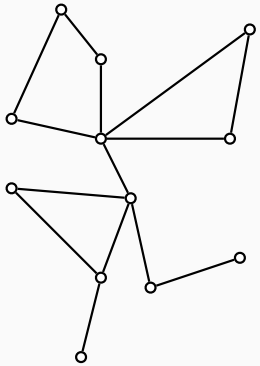


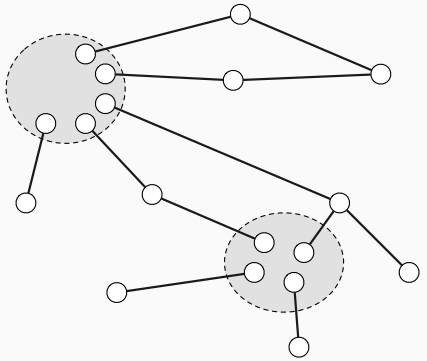


Cactus: compact representation of all min-cuts in graph. [Karzanov and Timofeev 1986]

- Deterministic: $O(nm + n^2 \log n)$ running time. [Nagamochi, Nakamura, Ishii 2003]
- Randomized: $\tilde{O}(m)$ running time. [Karger 2009]

Hypercactus: compact representation of all min-cuts in hypergraphs. [Cheng 1999, Fleiner and Jordán 1999]





Cactus: compact representation of all min-cuts in graph. [Karzanov and Timofeev 1986]

- Deterministic: $O(nm + n^2 \log n)$ running time. [Nagamochi, Nakamura, Ishii 2003]
- Randomized: $\tilde{O}(m)$ running time. [Karger 2009]

Hypercactus: compact representation of all min-cuts in hypergraphs. [Cheng 1999, Fleiner and Jordán 1999]

- Deterministic: $O(np + n^2 \log n)$ running time. (Our result)

Our approach

The decomposition framework [Cunningham 1983].

Our approach

The decomposition framework [Cunningham 1983].

- decompose the hypergraph to structurally simple hypergraphs

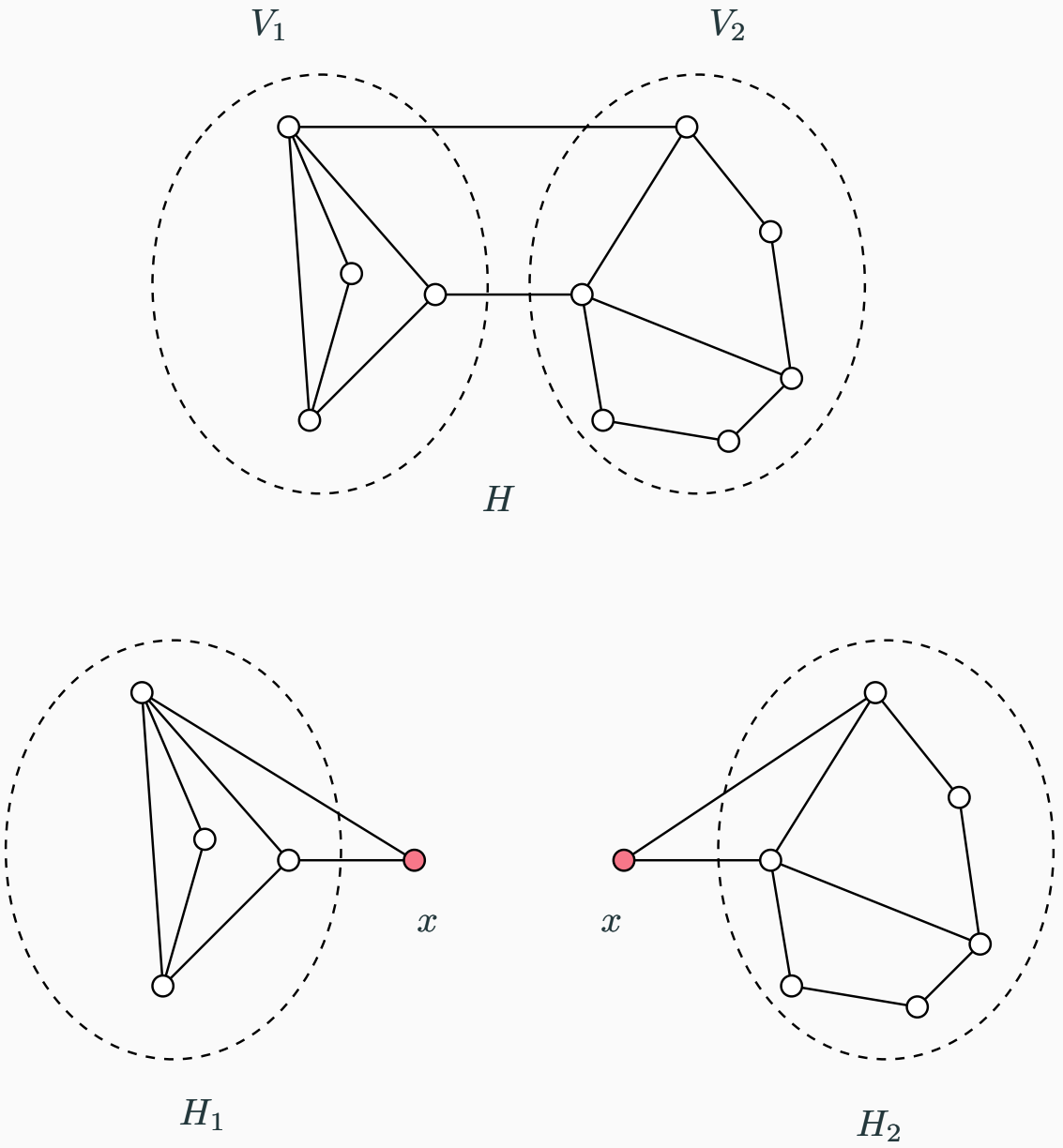
Our approach

The decomposition framework [Cunningham 1983].

- decompose the hypergraph to structurally simple hypergraphs
- preserves information on min-cuts, and might lose information on other cuts

A min-cut with at least two vertices on each side is called a **split**.

Decomposition is based on finding non-crossing splits.



Split oracle

A split oracle returns either:

- a split
- a pair of vertices not separated by any split

Split oracle

A split oracle returns either:

- a split
- a pair of vertices not separated by any split

Theorem

A hypercactus can be found through $O(n)$ queries to the split oracle.

Split oracle

A split oracle returns either:

- a split
- a pair of vertices not separated by any split

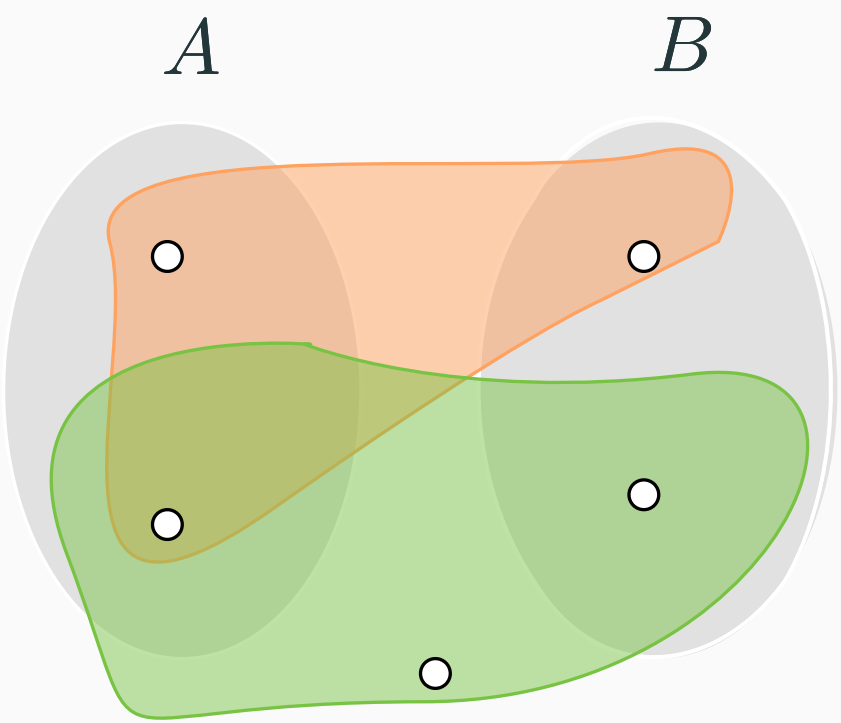
Theorem

A hypercactus can be found through $O(n)$ queries to the split oracle.

Goal: Split oracle in near linear time.

Tight adjacency

$$d'(A,B) = \sum_{\substack{e \in \delta(A) \cap \delta(B) \\ e \subset A \cup B}} w(e)$$

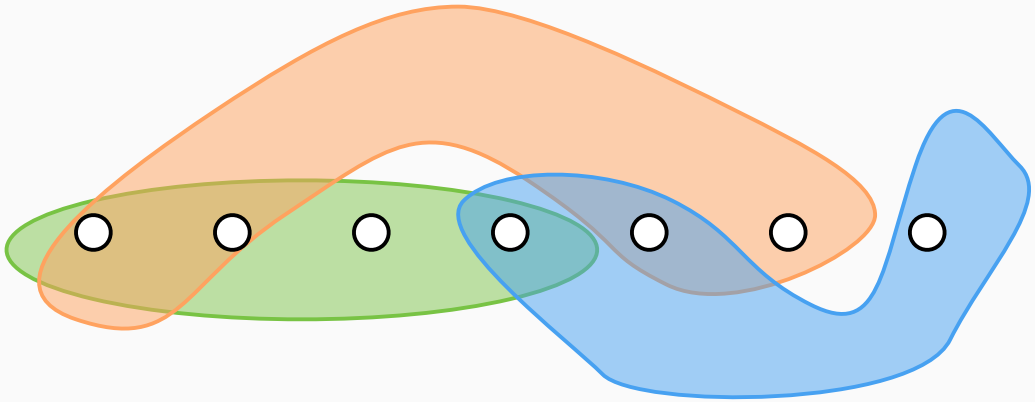


Tight ordering

$v_1, \ldots, v_n$ is a **tight ordering** if for all $1 \leq i \leq j \leq n$,

$$d'(\{v_1, \ldots, v_{i-1}\}, v_i) \geq d'(\{v_1, \ldots, v_{i-1}\}, v_j)$$

.

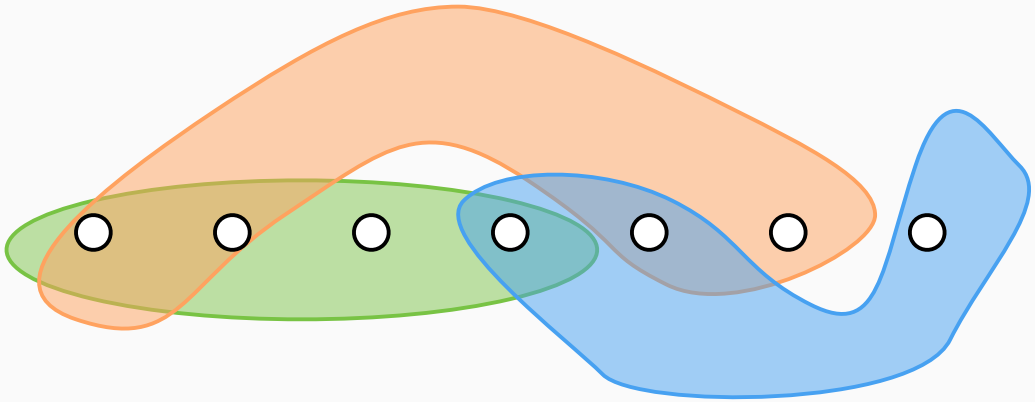


Tight ordering

$v_1, \ldots, v_n$ is a **tight ordering** if for all $1 \leq i \leq j \leq n$,

$$d'(\{v_1, \ldots, v_{i-1}\}, v_i) \geq d'(\{v_1, \ldots, v_{i-1}\}, v_j)$$

.



Tight ordering can be found in $O(p + n \log n)$ [Mak and Wong 2000].

Implement split oracle

1. Trim H to a *graph*:

Implement split oracle

1. Trim H to a *graph*: maintain $d'(\{v_1, \ldots, v_{i-1}\}, v_i)$ for all i.

Implement split oracle

1. Trim H to a *graph*: maintain $d'(\{v_1, \ldots, v_{i-1}\}, v_i)$ for all i.
2. Find max $v_{n-1}v_n$ flow:

Implement split oracle

1. Trim H to a *graph*: maintain $d'(\{v_1, \ldots, v_{i-1}\}, v_i)$ for all i.
2. Find max $v_{n-1}v_n$ flow: $O(p)$ time. [Arikati and Melhorn 1999]

Implement split oracle

1. Trim H to a *graph*: maintain $d'(\{v_1, \ldots, v_{i-1}\}, v_i)$ for all i.
2. Find max $v_{n-1}v_n$ flow: $O(p)$ time. [Arikati and Melhorn 1999]
3. Obtain min $v_{n-1}v_n$-cuts in H.

Implement split oracle

1. Trim H to a *graph*: maintain $d'(\{v_1, \ldots, v_{i-1}\}, v_i)$ for all i.
2. Find max $v_{n-1}v_n$ flow: $O(p)$ time. [Arikati and Melhorn 1999]
3. Obtain min $v_{n-1}v_n$-cuts in H.

Split oracle in $O(p + n \log n)$ time.

Implement split oracle

1. Trim H to a *graph*: maintain $d'(\{v_1, \ldots, v_{i-1}\}, v_i)$ for all i.
2. Find max $v_{n-1}v_n$ flow: $O(p)$ time. [Arikati and Melhorn 1999]
3. Obtain min $v_{n-1}v_n$-cuts in H.

Split oracle in $O(p + n \log n)$ time.

Hypercactus representation in $O(np + n^2 \log n)$ time.

$(2+\epsilon)$-min-cut approximation

$(2+\epsilon)$-min-cut approximation

- Inspired by Matula's $(2+\epsilon)$-approximation for graphs.

$(2+\epsilon)$-min-cut approximation

- Inspired by Matula's $(2+\epsilon)$-approximation for graphs.
- Uses either MA-ordering or Queyranne ordering.

$(2+\epsilon)$-min-cut approximation

- Inspired by Matula's $(2+\epsilon)$-approximation for graphs.
- Uses either MA-ordering or Queyranne ordering.
- Running time $O(\epsilon^{-1}(p + n \log n) \log n)$

Thank you.