Directed hypergraph connectivity augmentation by hyperarc re-orientations

Combinatorial Optimization and Graph Theory

Benoît Bompol, Armand Grenier

Thursday, Nov 23rd 2023

Table of contents

State of the art, goal of the article

State of the art, goal of the article

- *Nash-Williams*, 1960 :
  - G is $2k$-edge connected $\iff$ G admits a k-arc-connected orientation.

State of the art, goal of the article

- *Nash-Williams*, 1960 :
  - G is $2k$-edge connected $\iff$ G admits a k-arc-connected orientation.
- *Ito and al.*, 2023 :
  - Algorithmic proof of *Nash-Williams*, by flipping one arc at a time.
  - Exhibiting a sequence of orientations such that :
    - The arc-connectivity does not decrease, and the arc-connectivity of the last element of the sequence is k.
    - The next orientation in the sequence can be obtained from the previous one by flipping exactly one arc.
    - The sequence can be obtained in polynomial time (in the size of the directed graph).

State of the art, goal of the article

- *Nash-Williams*, 1960 :
  - G is $2k$-edge connected $\iff$ G admits a k-arc-connected orientation.
- *Ito and al.*, 2023 :
  - Algorithmic proof of *Nash-Williams*, by flipping one arc at a time.
  - Exhibiting a sequence of orientations such that :
    - The arc-connectivity does not decrease, and the arc-connectivity of the last element of the sequence is k.
    - The next orientation in the sequence can be obtained from the previous one by flipping exactly one arc.
    - The sequence can be obtained in polynomial time (in the size of the directed graph).

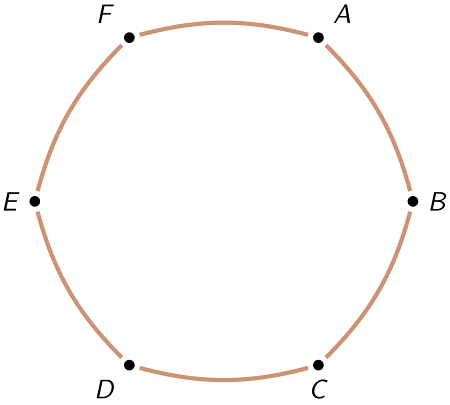
Goal of the article : Expanding the result of **Ito and al.** to hypergraphs.

State of the art, goal of the article

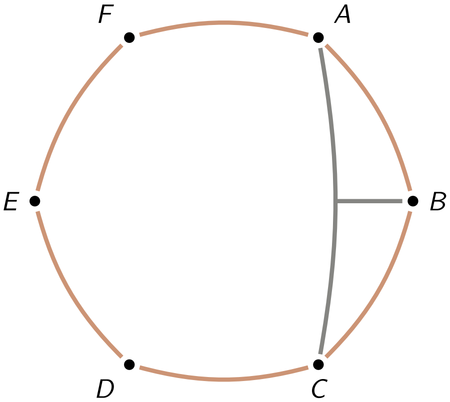
- *Nash-Williams*, 1960 :
  - G is $2k$-edge connected $\iff$ G admits a k-arc-connected orientation.
- *Ito and al.*, 2023 :
  - Algorithmic proof of *Nash-Williams*, by flipping one arc at a time.
  - Exhibiting a sequence of orientations such that :
    - The arc-connectivity does not decrease, and the arc-connectivity of the last element of the sequence is k.
    - The next orientation in the sequence can be obtained from the previous one by flipping exactly one arc.
    - The sequence can be obtained in polynomial time (in the size of the directed graph).

Goal of the article : Expanding the result of **Ito and al.** to hypergraphs.
Side note : This article generalise the results of **Ito and al.**, as directed graphs are special case of hypergraphs.

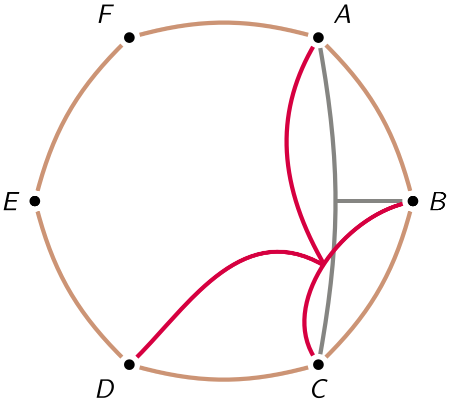
Hypergraphs



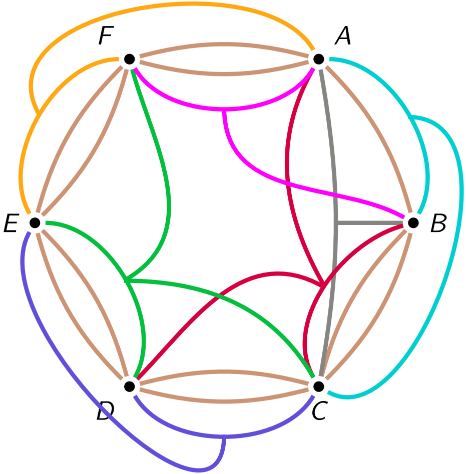
Hypergraphs



Hypergraphs

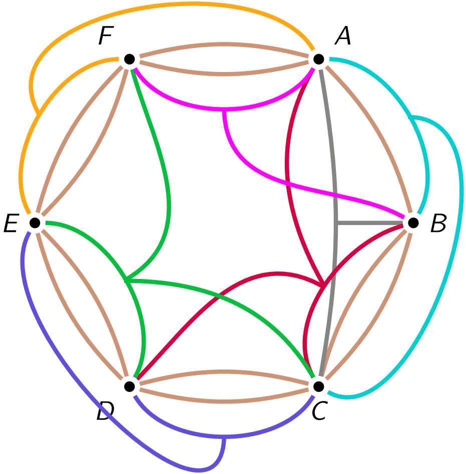


Hypergraphs



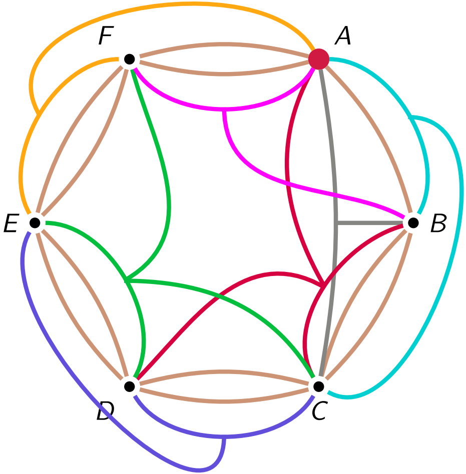
Degree of $\varnothing \neq X \subsetneq V$

$d_{\mathcal{H}}(X)$ is the number of hyperedges intersecting both X and $V \setminus X$.



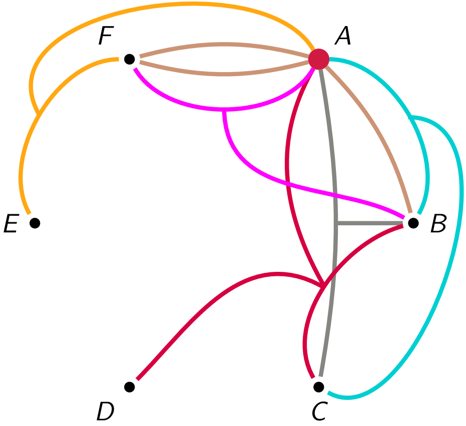
Degree of $\varnothing \neq X \subsetneq V$

$d_{\mathcal{H}}(X)$ is the number of hyperedges intersecting both X and $V \setminus X$.

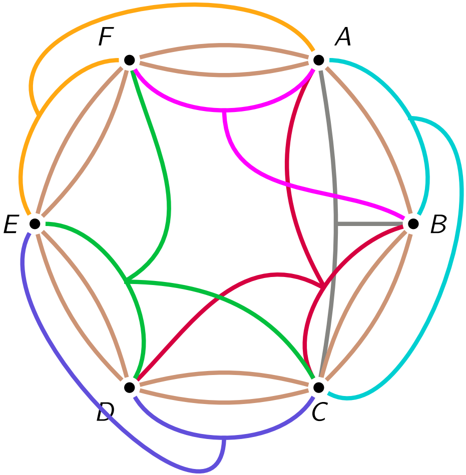


Degree of $\varnothing \neq X \subsetneq V$

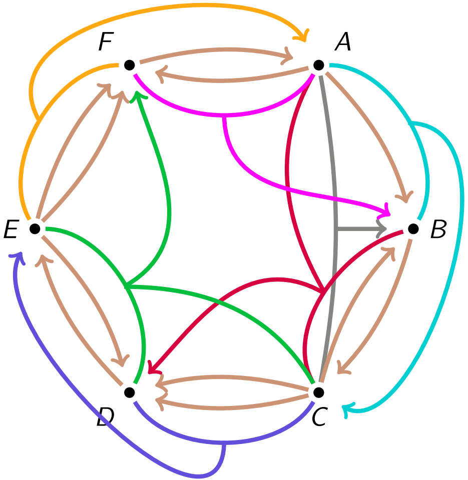
$d_{\mathcal{H}}(X)$ is the number of hyperedges intersecting both X and $V \setminus X$.



Orientation of an hypergraph

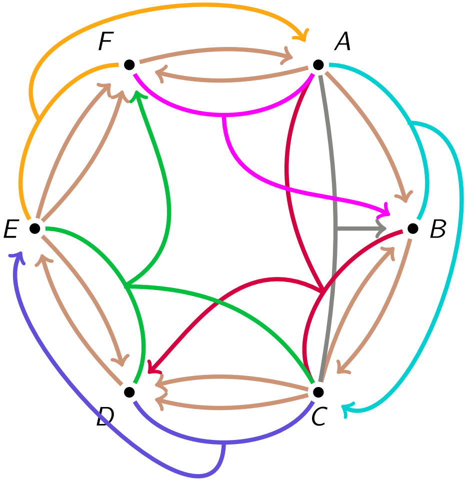


Orientation of an hypergraph



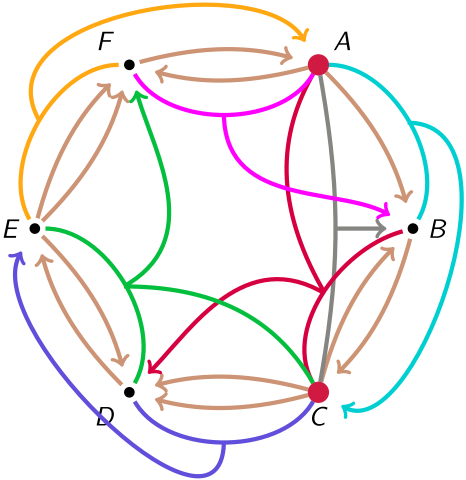
In-Degree of $\varnothing \neq X \subsetneq V$

$d^-_{\mathcal{H}}(X)$ is the number of hyperarcs (Y, v) such that : $v \in X, \exists u \in Y \setminus X$.



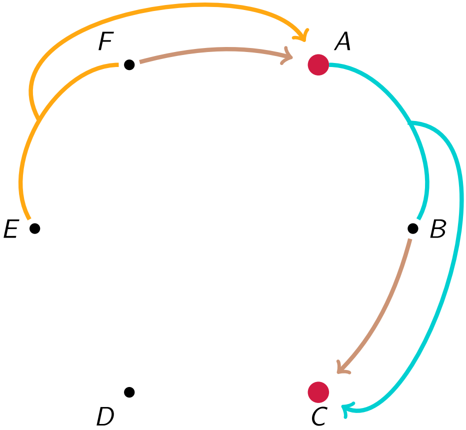
In-Degree of $\varnothing \neq X \subsetneq V$

$d^-_{\mathcal{H}}(X)$ is the number of hyperarcs (Y, v) such that : $v \in X, \exists u \in Y \setminus X$.



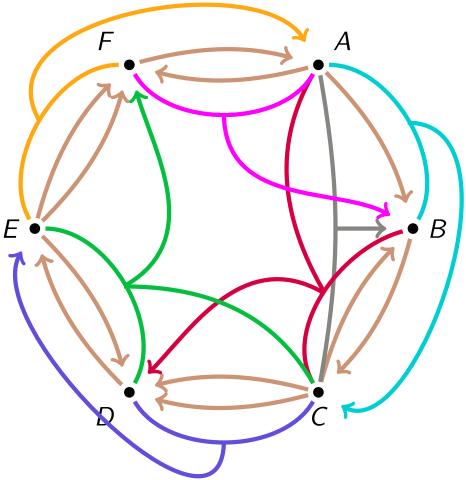
In-Degree of $\emptyset \neq X \subsetneq V$

$d^-_{\mathcal{H}}(X)$ is the number of hyperarcs (Y, v) such that : $v \in X$, $\exists u \in Y \setminus X$.



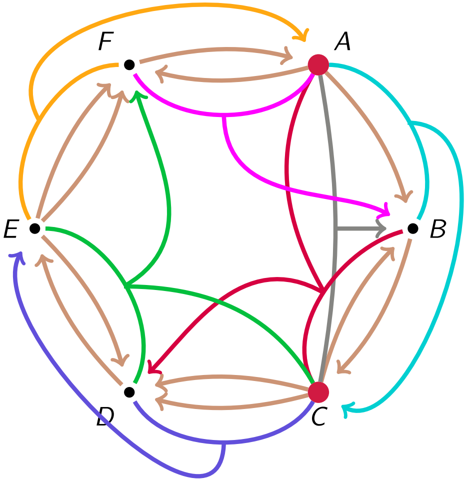
Out-Degree of $\varnothing \neq X \subsetneq V$

$d^+_{\mathcal{H}}(X)$ is the number of hyperarcs (Y, v) such that $v \notin X$ and $\exists u \in Y \cap X$.



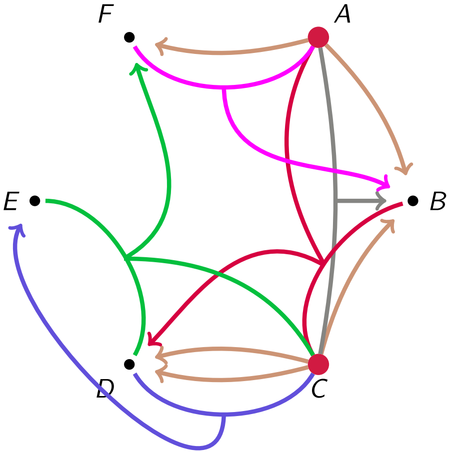
Out-Degree of $\varnothing \neq X \subsetneq V$

$d^+_{\mathcal{H}}(X)$ is the number of hyperarcs (Y, v) such that $v \notin X$ and $\exists u \in Y \cap X$.



Out-Degree of $\varnothing \neq X \subsetneq V$

$d^+_{\mathcal{H}}(X)$ is the number of hyperarcs (Y, v) such that $v \notin X$ and $\exists u \in Y \cap X$.



Hyperarc-connectivity and (k, k)-partition connected hypergraphs

- $\vec{\mathcal{H}}$ is *k-hyperarc-connected*, if, $\forall \varnothing \neq X \subsetneq V$, $d^+_{\vec{\mathcal{H}}}(X) \geq k$.
- The hyperarc-connectivity of a hypergraph, denoted $\lambda(\vec{\mathcal{H}})$, is the maximum value of k such that $\vec{\mathcal{H}}$ is *k-hyperarc-connected*.

Hyperarc-connectivity and (k, k)-partition connected hypergraphs

- $\vec{\mathcal{H}}$ is *k-hyperarc-connected*, if, $\forall \varnothing \neq X \subsetneq V$, $d^+_{\vec{\mathcal{H}}}(X) \geq k$.
- The hyperarc-connectivity of a hypergraph, denoted $\lambda(\vec{\mathcal{H}})$, is the maximum value of k such that $\vec{\mathcal{H}}$ is *k-hyperarc-connected*.
- The previous orientation given was 2-hyperarc-connected.

Hyperarc-connectivity and (k, k)-partition connected hypergraphs

- $\vec{\mathcal{H}}$ is *k-hyperarc-connected*, if, $\forall \varnothing \neq X \subsetneq V$, $d^+_{\vec{\mathcal{H}}}(X) \geq k$.
- The hyperarc-connectivity of a hypergraph, denoted $\lambda(\vec{\mathcal{H}})$, is the maximum value of k such that $\vec{\mathcal{H}}$ is *k-hyperarc-connected*.
- The previous orientation given was 2-hyperarc-connected.
- Let $\mathcal{P}$ be a partition of V :
- $e_{\mathcal{H}}(\mathcal{P})$ is the number of hyperedges intersecting at least 2 elements of $\mathcal{P}$

Hyperarc-connectivity and (k, k)-partition connected hypergraphs

- $\vec{\mathcal{H}}$ is *k-hyperarc-connected*, if, $\forall \varnothing \neq X \subsetneq V$, $d^+_{\vec{\mathcal{H}}}(X) \geq k$.
- The hyperarc-connectivity of a hypergraph, denoted $\lambda(\vec{\mathcal{H}})$, is the maximum value of k such that $\vec{\mathcal{H}}$ is *k-hyperarc-connected*.
- The previous orientation given was 2-hyperarc-connected.
- Let $\mathcal{P}$ be a partition of V :
- $e_{\mathcal{H}}(\mathcal{P})$ is the number of hyperedges intersecting at least 2 elements of $\mathcal{P}$
- $\mathcal{H}$ is (k, k)-partition-connected, if :
  - $\forall \mathcal{P}, e_{\mathcal{H}}(\mathcal{P}) \geq k \times |\mathcal{P}|$

Hyperarc-connectivity and (k, k)-partition connected hypergraphs

- $\vec{\mathcal{H}}$ is *k-hyperarc-connected*, if, $\forall \varnothing \neq X \subsetneq V$, $d^+_{\vec{\mathcal{H}}}(X) \geq k$.
- The hyperarc-connectivity of a hypergraph, denoted $\lambda(\vec{\mathcal{H}})$, is the maximum value of k such that $\vec{\mathcal{H}}$ is *k-hyperarc-connected*.
- The previous orientation given was 2-hyperarc-connected.
- Let $\mathcal{P}$ be a partition of V :
- $e_{\mathcal{H}}(\mathcal{P})$ is the number of hyperedges intersecting at least 2 elements of $\mathcal{P}$
- $\mathcal{H}$ is (k, k)-partition-connected, if :
  - $\forall \mathcal{P}, e_{\mathcal{H}}(\mathcal{P}) \geq k \times |\mathcal{P}|$

We use a result of Frank : $\mathcal{H}$ is (k, k)-partition-connected if and only if it admits a k-hyperarc-connected orientation.

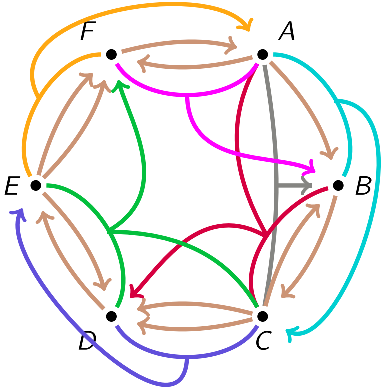
Main result

Main result (Theorem 7)

Let $\mathcal{H} = (V, E)$ be a $(k+1, k+1)$-partition-connected hypergraph and $\vec{\mathcal{H}}$ is a k-hyperarc connected orientation of $\mathcal{H}$. Then there exists a sequence of hypergraphs $(\vec{\mathcal{H}}_i)_{i \in 0 \dots \ell}$ such that $\vec{\mathcal{H}}_{i+1}$ is obtained from $\vec{\mathcal{H}}_i$ by reorienting exactly one hyperarc and $\lambda(\vec{\mathcal{H}}_{i+1}) \geq \lambda(\vec{\mathcal{H}}_i)$ and $\lambda(\vec{\mathcal{H}}_\ell) = k+1$. Such a sequence of orientations can be obtained with $\ell \leq |V|^3$ and found in polynomial time (in the size of $\mathcal{H}$).

"High-Level"-running of the algorithm

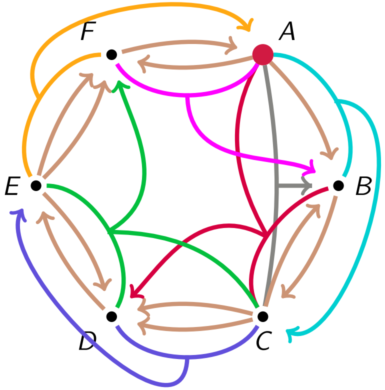
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

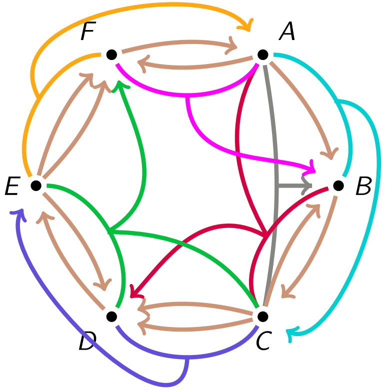
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

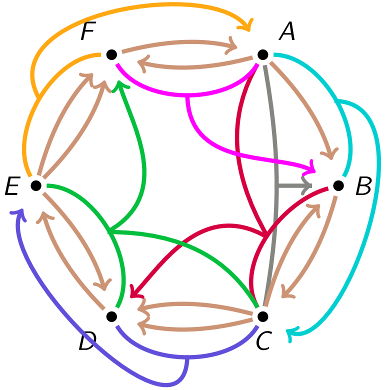
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

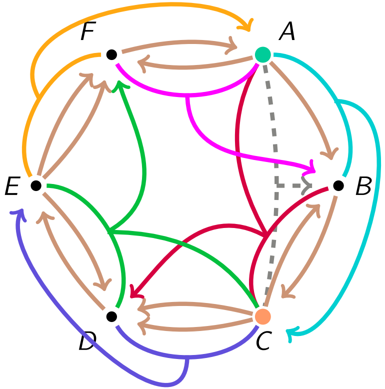
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

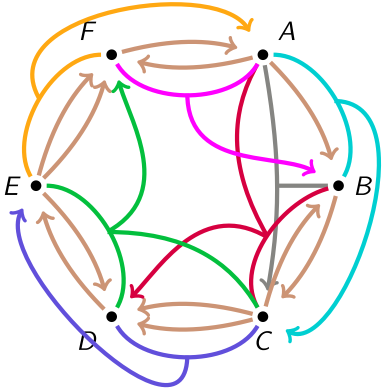
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

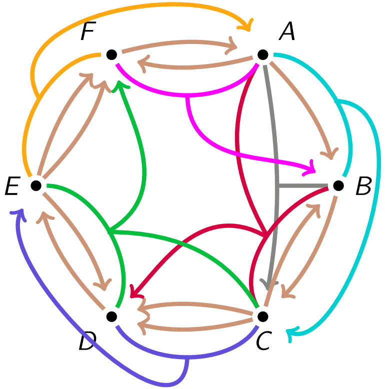
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

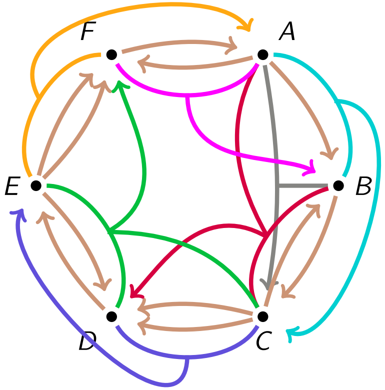
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s,t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

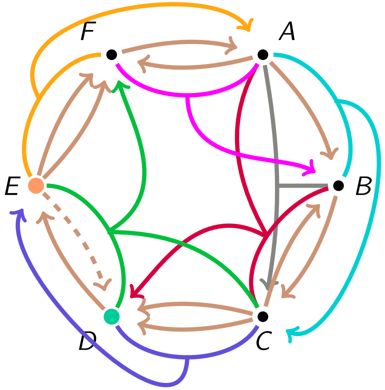
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s,t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

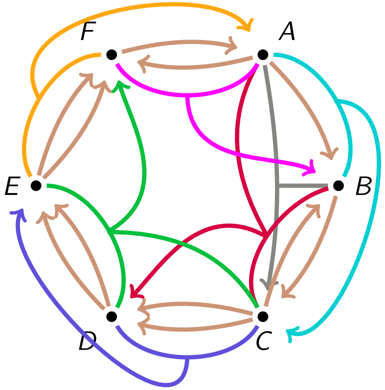
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

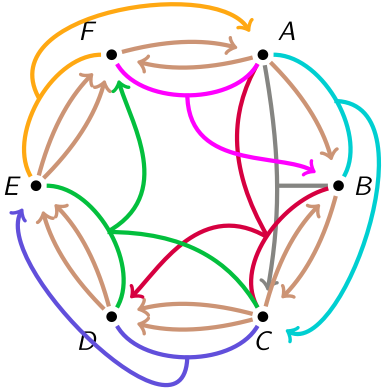
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

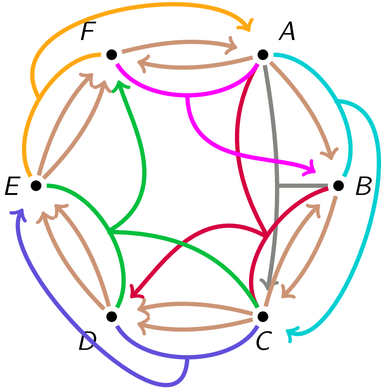
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

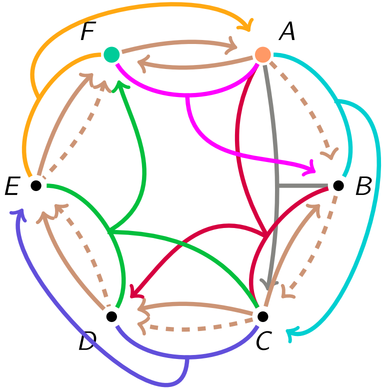
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

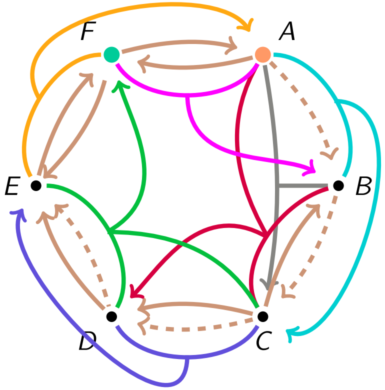
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

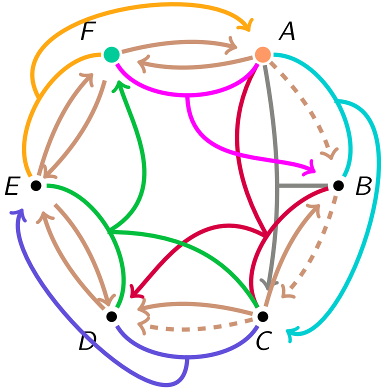
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

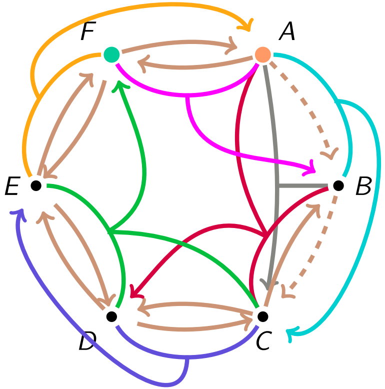
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

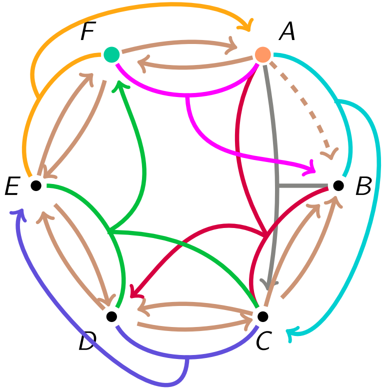
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

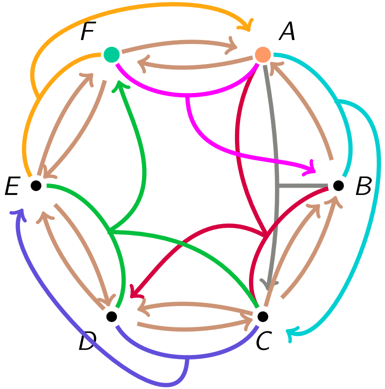
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

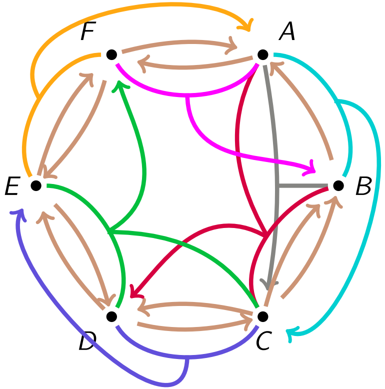
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s,t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

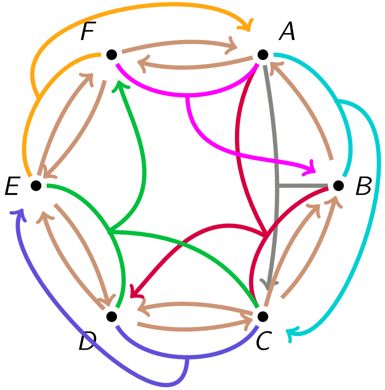
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

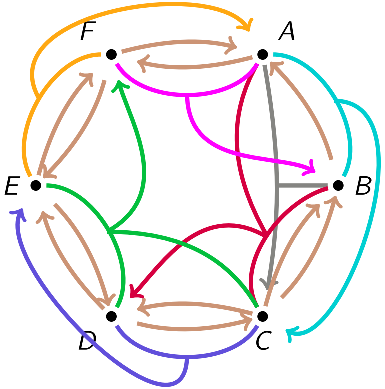
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. Find an admissible (s, t)-hyperpath in R to reorient
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

"High-Level"-running of the algorithm

Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



1. Take r in $V(\mathcal{H})$.
2. Compute sets of vertices.
3. Stopping Criteria
4. Select a set R (cf. 2.)
5. **Find an admissible (s, t)-hyperpath in R to reorient**
6. Reorient the corresponding hyperpath.
7. `Goto` (2.)

Finding *admissible* (s, t)-hyperpaths in R

- Crucial segment of the algorithm.
- Three criterion for P to be an admissible (s, t)-hyperpath in R:
  1. s is a safe source in $S \subseteq R$, t is a safe sink in $T \subseteq R$.
  2. Reorient each hyperarc, **one by one**, does not decrease the hyperarc-connectivity.
  3. After reorientation of P, there is a set whose cardinality is a guarantee that the algorithm will stop.

A brief detour...

Finding *admissible* (s, t)-hyperpaths in R

- Crucial segment of the algorithm.
- Three criterion for P to be an admissible (s, t)-hyperpath in R:
  1. s is a safe source in $S \subseteq R$, t is a safe sink in $T \subseteq R$.
  2. Reorient each hyperarc, **one by one**, does not decrease the hyperarc-connectivity.
  3. After reorientation of P, there is a set whose cardinality is a guarantee that the algorithm will stop.

A brief detour...

Finding *admissible* (s, t)-hyperpaths in R

- Crucial segment of the algorithm.
- Three criterion for P to be an admissible (s, t)-hyperpath in R:
  1. s is a safe source in $S \subseteq R$, t is a safe sink in $T \subseteq R$.
  2. Reorient each hyperarc, **one by one**, does not decrease the hyperarc-connectivity.
  3. After reorientation of P, there is a set whose cardinality is a guarantee that the algorithm will stop.

A brief detour...

Finding *admissible* (s, t)-hyperpaths in R

- Crucial segment of the algorithm.
- Three criterion for P to be an admissible (s, t)-hyperpath in R:
  1. s is a safe source in $S \subseteq R$, t is a safe sink in $T \subseteq R$.
  2. Reorient each hyperarc, **one by one**, does not decrease the hyperarc-connectivity.
  3. After reorientation of P, there is a set whose cardinality is a guarantee that the algorithm will stop.

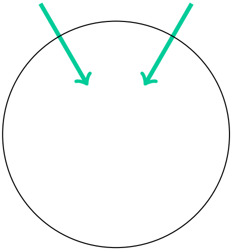
A brief detour...

Finding *admissible* (s, t)-hyperpaths in R

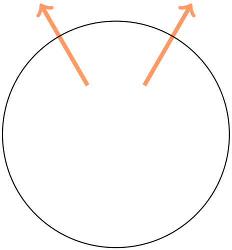
- Crucial segment of the algorithm.
- Three criterion for P to be an admissible (s, t)-hyperpath in R:
  1. s is a safe source in $S \subseteq R$, t is a safe sink in $T \subseteq R$.
  2. Reorient each hyperarc, **one by one**, does not decrease the hyperarc-connectivity.
  3. After reorientation of P, there is a set whose cardinality is a guarantee that the algorithm will stop.

A brief detour...

Tight and Dangerous sets



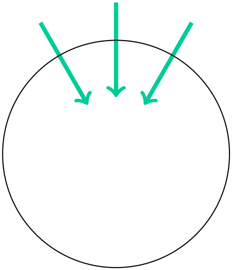
In-Tight sets



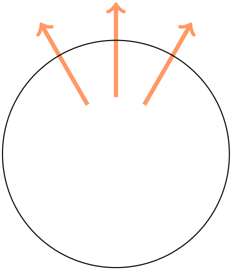
Out-Tight sets

- $\mathcal{T}_- = \{X \subseteq V - r, d^-(X) = k\} \cup \{V\}$
- $\mathcal{T}_+ = \{X \subseteq V - r, d^+(X) = k\} \cup \{V\}$
- $\mathcal{D}_- = \{X \subseteq V - r, d^-(X) = k + 1\}$
- $\mathcal{D}_+ = \{X \subseteq V - r, d^+(X) = k + 1\}$
- $\mathcal{M}_-$: Inclusion-wise minimal members of $\mathcal{T}_-$
- $\mathcal{M}_+$: Inclusion-wise minimal members of $\mathcal{T}_+$
- $\mathcal{M}$: Inclusion-wise minimal members of $\mathcal{M}_- \cup \mathcal{M}_+$

Tight and Dangerous sets



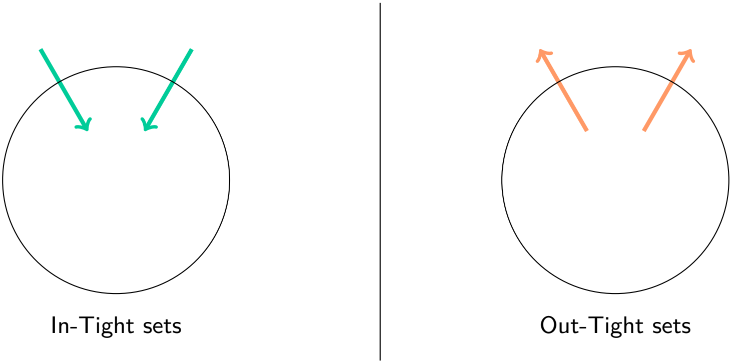
In-Dangerous sets



Out-Dangerous sets

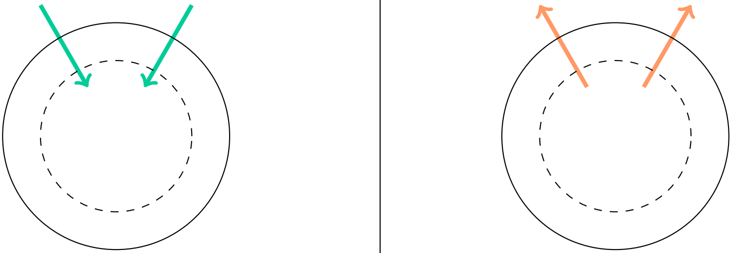
- $\mathcal{T}_- = \{X \subseteq V - r, d^-(X) = k\} \cup \{V\}$
- $\mathcal{T}_+ = \{X \subseteq V - r, d^+(X) = k\} \cup \{V\}$
- $\mathcal{D}_- = \{X \subseteq V - r, d^-(X) = k + 1\}$
- $\mathcal{D}_+ = \{X \subseteq V - r, d^+(X) = k + 1\}$
- $\mathcal{M}_-$: Inclusion-wise minimal members of $\mathcal{T}_-$
- $\mathcal{M}_+$: Inclusion-wise minimal members of $\mathcal{T}_+$
- $\mathcal{M}$: Inclusion-wise minimal members of $\mathcal{M}_- \cup \mathcal{M}_+$

Tight and Dangerous sets



- $\mathcal{T}_- = \{X \subseteq V - r, d^-(X) = k\} \cup \{V\}$
- $\mathcal{T}_+ = \{X \subseteq V - r, d^+(X) = k\} \cup \{V\}$
- $\mathcal{D}_- = \{X \subseteq V - r, d^-(X) = k + 1\}$
- $\mathcal{D}_+ = \{X \subseteq V - r, d^+(X) = k + 1\}$
- $\mathcal{M}_-$: Inclusion-wise minimal members of $\mathcal{T}_-$
- $\mathcal{M}_+$: Inclusion-wise minimal members of $\mathcal{T}_+$
- $\mathcal{M}$: Inclusion-wise minimal members of $\mathcal{M}_- \cup \mathcal{M}_+$

Tight and Dangerous sets



Minimal In-Tight sets

Minimal Out-Tight sets

- $\mathcal{T}_- = \{X \subseteq V - r, d^-(X) = k\} \cup \{V\}$
- $\mathcal{T}_+ = \{X \subseteq V - r, d^+(X) = k\} \cup \{V\}$
- $\mathcal{D}_- = \{X \subseteq V - r, d^-(X) = k + 1\}$
- $\mathcal{D}_+ = \{X \subseteq V - r, d^+(X) = k + 1\}$
- $\mathcal{M}_-$: Inclusion-wise minimal members of $\mathcal{T}_-$
- $\mathcal{M}_+$: Inclusion-wise minimal members of $\mathcal{T}_+$
- $\mathcal{M}$: Inclusion-wise minimal members of $\mathcal{M}_- \cup \mathcal{M}_+$

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Proof of Claim 1(b)

- We have $\lambda(\vec{\mathcal{H}}) = k$
- Since X, Y are crossing, $X \cap Y \neq \varnothing$, $X \cup Y \neq V$.
- $k + k = d^+(X) + d^+(Y)$
- By submodularity, $d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y)$
- By $\lambda(\vec{\mathcal{H}}) = k$, $d^+(X \cup Y) \geq k$ and $d^+(X \cap Y) \geq k$
- Grouping these equations, we obtain :
  $k + k = d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y) \geq k + k$.
- This implies $d^+(X \cup Y) = k$, $d^+(X \cap Y)$, i.e. $X \cap Y, X \cup Y \in \mathcal{T}_+$

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Proof of Claim 1(b)

- We have $\lambda(\vec{\mathcal{H}}) = k$
- Since X, Y are crossing, $X \cap Y \neq \varnothing$, $X \cup Y \neq V$.
- $k + k = d^+(X) + d^+(Y)$
- By submodularity, $d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y)$
- By $\lambda(\vec{\mathcal{H}}) = k$, $d^+(X \cup Y) \geq k$ and $d^+(X \cap Y) \geq k$
- Grouping these equations, we obtain : $k + k = d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y) \geq k + k$.
- This implies $d^+(X \cup Y) = k$, $d^+(X \cap Y)$, i.e. $X \cap Y, X \cup Y \in \mathcal{T}_+$

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Proof of Claim 1(b)

- We have $\lambda(\vec{\mathcal{H}}) = k$
- Since X, Y are crossing, $X \cap Y \neq \varnothing$, $X \cup Y \neq V$.
- $k + k = d^+(X) + d^+(Y)$
- By submodularity, $d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y)$
- By $\lambda(\vec{\mathcal{H}}) = k$, $d^+(X \cup Y) \geq k$ and $d^+(X \cap Y) \geq k$
- Grouping these equations, we obtain :
  $k + k = d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y) \geq k + k$.
- This implies $d^+(X \cup Y) = k$, $d^+(X \cap Y)$, i.e. $X \cap Y, X \cup Y \in \mathcal{T}_+$

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Proof of Claim 1(b)

- We have $\lambda(\vec{\mathcal{H}}) = k$
- Since X, Y are crossing, $X \cap Y \neq \varnothing$, $X \cup Y \neq V$.
- $k + k = d^+(X) + d^+(Y)$
- By submodularity, $d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y)$
- By $\lambda(\vec{\mathcal{H}}) = k$, $d^+(X \cup Y) \geq k$ and $d^+(X \cap Y) \geq k$
- Grouping these equations, we obtain : $k + k = d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y) \geq k + k$.
- This implies $d^+(X \cup Y) = k$, $d^+(X \cap Y)$, i.e. $X \cap Y, X \cup Y \in \mathcal{T}_+$

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Proof of Claim 1(b)

- We have $\lambda(\vec{\mathcal{H}}) = k$
- Since X, Y are crossing, $X \cap Y \neq \varnothing$, $X \cup Y \neq V$.
- $k + k = d^+(X) + d^+(Y)$
- By submodularity, $d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y)$
- By $\lambda(\vec{\mathcal{H}}) = k$, $d^+(X \cup Y) \geq k$ and $d^+(X \cap Y) \geq k$
- Grouping these equations, we obtain :
  $k + k = d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y) \geq k + k$.
- This implies $d^+(X \cup Y) = k$, $d^+(X \cap Y)$, i.e. $X \cap Y, X \cup Y \in \mathcal{T}_+$

Crossing sets and structural results

Let X, Y two crossing sets in V.

Claim 1(b)

If $X, Y \in \mathcal{T}_+$, then both $X \cup Y \in \mathcal{T}_+$ and $X \cap Y \in \mathcal{T}_+$.

Proof of Claim 1(b)

- We have $\lambda(\vec{\mathcal{H}}) = k$
- Since X, Y are crossing, $X \cap Y \neq \varnothing$, $X \cup Y \neq V$.
- $k + k = d^+(X) + d^+(Y)$
- By submodularity, $d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y)$
- By $\lambda(\vec{\mathcal{H}}) = k$, $d^+(X \cup Y) \geq k$ and $d^+(X \cap Y) \geq k$
- Grouping these equations, we obtain :
  $k + k = d^+(X) + d^+(Y) \geq d^+(X \cup Y) + d^+(X \cap Y) \geq k + k$.
- This implies $d^+(X \cup Y) = k$, $d^+(X \cap Y)$, i.e. $X \cap Y, X \cup Y \in \mathcal{T}_+$

Existence of a safe source (*a safe sink*)

Lemma 10

$\forall S \in \mathcal{M}_-$, there is a safe source $s \in S$.

Likewise,

Lemma 11

$\forall T \in \mathcal{M}_+$, there is a safe sink $t \in T$.

Quick outline of a proof for Lemma 10 :

- Let $S \in \mathcal{M}_-$.
- Considering a family of vertex sets (χ) that cover as many vertices of S as possible, but using as little as vertex sets possible.
- We can prove that, under given assumptions, χ cannot cover every vertex of S.
- Vertices that are not covered by χ are "potential" safe sources, the last part of the proof is verifying that they are effectively safe sources.

Towards hyperarc connectivity augmentation

$\mathcal{R}$: $R \subseteq V - r$ inclusion-wise minimal such that either :

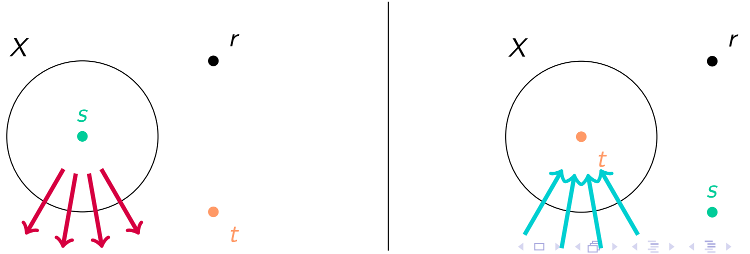
- $R \in \mathcal{T}_-$, and contains a member of $\mathcal{T}_+$
- or $R \in \mathcal{T}_+$, and contains a member of $\mathcal{T}_-$.

Lemma 13

Let $R \in \mathcal{R}, S \in \mathcal{M}_-, T \in \mathcal{M}_+$ such that $S, T \subseteq R$. Let s be a safe source in S, t a safe sink in T.

a. $\forall X \subseteq V - r$ such that $s \in X$, $t \notin X$, we have $d^+(X) \geq k + 1$.

b. $\forall X \subseteq V - r$ such that $s \notin X$, $t \in X$, we have $d^-(X) \geq k + 1$.



Towards hyperarc connectivity augmentation

Lemma 13

Let $R \in \mathcal{R}, S \in \mathcal{M}_{-}, T \in \mathcal{M}_{+}$ such that $S, T \subseteq R$. Let s be a safe source in S, t a safe sink in T.
Then :

- $\forall X \subseteq V - r$ such that $s \in X$, $t \notin X$, we have $d^{+}(X) \geq k + 1$.
- $\forall X \subseteq V - r$ such that $s \notin X$, $t \in X$, we have $d^{-}(X) \geq k + 1$.

Towards hyperarc connectivity augmentation

Lemma 13

Let $R \in \mathcal{R}, S \in \mathcal{M}_-$, $T \in \mathcal{M}_+$ such that $S, T \subseteq R$. Let s be a safe source in S, t a safe sink in T.
Then :

- $\forall X \subseteq V - r$ such that $s \in X$, $t \notin X$, we have $d^+(X) \geq k+1$.
- $\forall X \subseteq V - r$ such that $s \notin X$, $t \in X$, we have $d^-(X) \geq k+1$.

Proof of Lemma 13

By contradiction, either :

a. $s \in X, t \notin X, d^+(X) = k$, i.e. $s \in X, t \notin X, X \in \mathcal{T}_+$.
   - a1. $R \in \mathcal{R} \cap \mathcal{T}_-$
   - a2. $R \in \mathcal{R} \cap \mathcal{T}_+$

b. $s \notin X, t \in X, d^-(X) = k$, i.e. $s \notin X, t \in X, X \in \mathcal{T}_-$.
   - b1. $R \in \mathcal{R} \cap \mathcal{T}_-$
   - b2. $R \in \mathcal{R} \cap \mathcal{T}_+$

Towards hyperarc connectivity augmentation

Lemma 13

Let $R \in \mathcal{R}, S \in \mathcal{M}_-, T \in \mathcal{M}_+$ such that $S, T \subseteq R$. Let s be a safe source in S, t a safe sink in T.
Then :

- $\forall X \subseteq V - r$ such that $s \in X$, $t \notin X$, we have $d^+(X) \geq k+1$.
- $\forall X \subseteq V - r$ such that $s \notin X$, $t \in X$, we have $d^-(X) \geq k+1$.

Proof of Lemma 13

By contradiction, either :

a. $s \in X, t \notin X, d^+(X) = k$, i.e. $s \in X, t \notin X, X \in \mathcal{T}_+$.

  a1. $R \in \mathcal{R} \cap \mathcal{T}_-$
  a2. $R \in \mathcal{R} \cap \mathcal{T}_+$

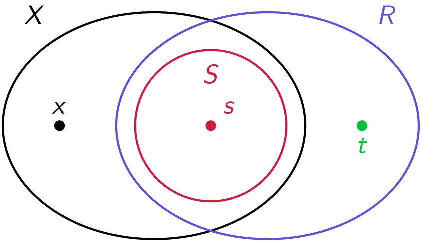
b. $s \notin X, t \in X, d^-(X) = k$, i.e. $s \notin X, t \in X, X \in \mathcal{T}_-$.

  b1. $R \in \mathcal{R} \cap \mathcal{T}_-$
  b2. $R \in \mathcal{R} \cap \mathcal{T}_+$

Proof of Lemma 13

$a : \exists X \subseteq V - r, s \in X, t \notin X, X \in \mathcal{T}_+$

- Since $s \in S$ is a **safe source** and $s \in X \in \mathcal{T}_+$, we have $S \subsetneq X$
- We also have $t \in R \setminus X$ by [a.], so $X \setminus R \neq \varnothing$.

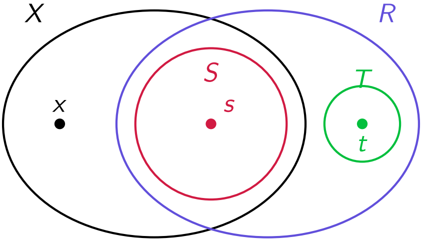


Proper representation of a

Proof of Lemma 13

$a : \exists X \subseteq V - r, s \in X, t \notin X, X \in \mathcal{T}_+$

- Since $s \in S$ is a **safe source** and $s \in X \in \mathcal{T}_+$, we have $S \subsetneq X$
- We also have $t \in R \setminus X$ by [a.], so $X \setminus R \neq \varnothing$.



Proper representation of a

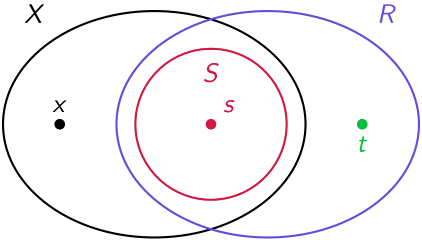
$a1.$: $R \in \mathcal{R} \cap \mathcal{T}_-, \exists X \subseteq V - r, s \in X, t \notin X, X \in \mathcal{T}_+$.

- As $t \in R \setminus X \neq \varnothing$, and using Claim 1, we have $R \setminus X \in \mathcal{T}_-$.
- $T \cap X \neq \varnothing$ would contradict the minimality of T, so T and X are disjoint.
- As $R \setminus X \in \mathcal{T}_-$, $T \in \mathcal{T}_+$, and $T \subseteq R \setminus X$, this contradicts R minimal.

Proof of Lemma 13

$a : \exists X \subseteq V - r, s \in X, t \notin X, X \in \mathcal{T}_+$

- Since $s \in S$ is a **safe source** and $s \in X \in \mathcal{T}_+$, we have $S \subsetneq X$
- We also have $t \in R \setminus X$ by [a.], so $X \setminus R \neq \varnothing$.



Proper representation of a

$a2.$: $R \in \mathcal{R} \cap \mathcal{T}_+, \exists X \subseteq V - r, s \in X, t \notin X, X \in \mathcal{T}_+$.

- $R \in \mathcal{T}_+, X \in \mathcal{T}_+$, and $X \cap R \neq \varnothing \implies X \cap R \in \mathcal{T}_+$
- $S \in \mathcal{T}_-, S \subseteq R \cap X$. Since $r \in R \setminus X, X \cap R \subsetneq R$.
- This contradicts the minimality of R.

Proof of Lemma 13

Proof of Lemma 13

By contradiction, either :

~~a~~ $s \in X, t \notin X, d^+(X) = k$, i.e. $s \in X, t \notin X, X \in \mathcal{T}_+$.

- ~~a1.~~ $R \in \mathcal{R} \cap \mathcal{T}_-$
- ~~a2.~~ $R \in \mathcal{R} \cap \mathcal{T}_+$

b. $s \notin X, t \in X, d^-(X) = k$, i.e. $s \notin X, t \in X, X \in \mathcal{T}_-$.

- b1. $R \in \mathcal{R} \cap \mathcal{T}_-$
- b2. $R \in \mathcal{R} \cap \mathcal{T}_+$

Finding *admissible* (s,t)-hyperpaths in $R \in \mathcal{R}$

Three criterion for P to be an admissible (s,t)-hyperpath in R:

1. s is a safe source in $S \subseteq R$, t is a safe sink in $T \subseteq R$.
2. Reorienting each hyperarc, **one by one**, does not decrease the hyperarc-connectivity
3. Let $\vec{\mathcal{H}}'$ the hypergraph obtained after reorientation of P.
   - $\mathcal{M}'$: Inclusion-wise minimal members of $\mathcal{M}'_- \cup \mathcal{M}'_+$
   - Either $|\mathcal{M}'| < |\mathcal{M}|$, either $|\mathcal{M}'| = |\mathcal{M}|$ and $\mathcal{M}'$ covers more vertices than $\mathcal{M}$.

Point 3. is the stopping criteria for the main algorithm :

- $\mathcal{M} = \{V\}$ implies both $\mathcal{M}_- = \{V\}$ and $\mathcal{M}_+ = \{V\}$.
- $\mathcal{T}_- = \{X \subseteq V - r, d^-(X) = k\} \cup \{V\}$
- $\mathcal{M}_-$: Inclusion-wise minimal members of $\mathcal{T}_-$
- Finally, if $\lambda(\vec{\mathcal{H}}) \geq k$ and $\mathcal{T}_- = \mathcal{T}_+ = \{V\}$, $\vec{\mathcal{H}}$ is $(k+1)$-hyperarc-connected.

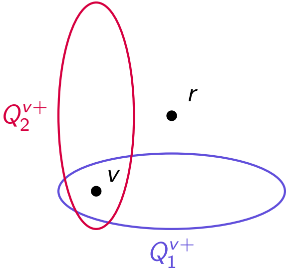
Introduction of Q_+^v

Definition of Q_+^v

Consider the sets of $\mathcal{T}_+$ containing v. Q_+^v is **the** minimal (inclusion-wise) one.

Unicity of Q_+^v :

If it exists, Q_+^v is unique.



Let Q_1^{v+}, Q_2^{v+} verifying the above definition.

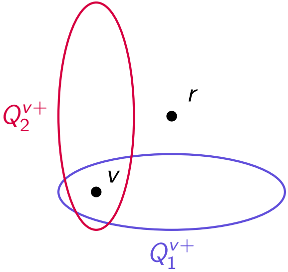
Introduction of Q_+^v

Definition of Q_+^v

Consider the sets of $\mathcal{T}_+$ containing v. Q_+^v is **the** minimal (inclusion-wise) one.

Unicity of Q_+^v :

If it exists, Q_+^v is unique.



By definition, $Q_1^{v+} \not\subseteq Q_2^{v+}$ and $Q_2^{v+} \not\subseteq Q_1^{v+}$.

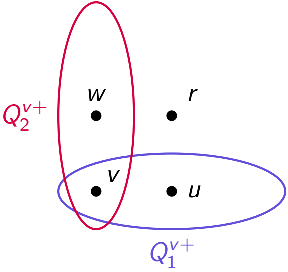
Introduction of Q_+^v

Definition of Q_+^v

Consider the sets of $\mathcal{T}_+$ containing v. Q_+^v is **the** minimal (inclusion-wise) one.

Unicity of Q_+^v :

If it exists, Q_+^v is unique.



Denote $u \in Q_1^{v+} \setminus Q_2^{v+}$, $w \in Q_2^{v+} \setminus Q_1^{v+}$.

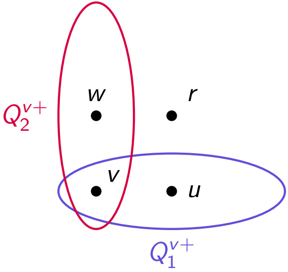
Introduction of Q_+^v

Definition of Q_+^v

Consider the sets of $\mathcal{T}_+$ containing v. Q_+^v is **the** minimal (inclusion-wise) one.

Unicity of Q_+^v :

If it exists, Q_+^v is unique.



As $r \notin Q_1^{v+}, Q_2^{v+}$, both are are crossing sets.

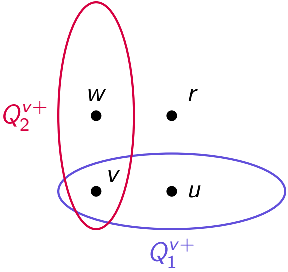
Introduction of Q_+^v

Definition of Q_+^v

Consider the sets of $\mathcal{T}_+$ containing v. Q_+^v is **the** minimal (inclusion-wise) one.

Unicity of Q_+^v :

If it exists, Q_+^v is unique.



By submodularity, if $X, Y \in \mathcal{T}_+$, both $X \cup V \in \mathcal{T}_+$ and $X \cap V \in \mathcal{T}_+$.

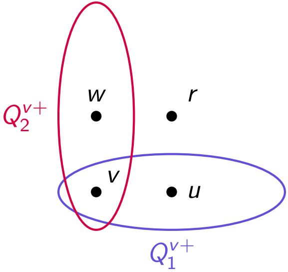
Introduction of Q_+^v

Definition of Q_+^v

Consider the sets of $\mathcal{T}_+$ containing v. Q_+^v is **the** minimal (inclusion-wise) one.

Unicity of Q_+^v :

If it exists, Q_+^v is unique.



$Q_1^{v+} \cap Q_2^{v+}$ is smaller (inclusion-wise) than Q_1^{v+} and Q_2^{v+}.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d_{\mathcal{H}}^+(Q_+^s) \geq d_{\mathcal{H}}^+(Z)$
  - $d_{\mathcal{H}}^+(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d_{\mathcal{H}}^+(Q_+^s)$ by definition.
- We can deduce that $d_{\mathcal{H}}^+(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d_{\mathcal{H}}^+(Q_+^s) \geq d_{\mathcal{H}}^+(Z)$
  - $d_{\mathcal{H}}^+(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d_{\mathcal{H}}^+(Q_+^s)$ by definition.
- We can deduce that $d_{\mathcal{H}}^+(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d^+_{\vec{\mathcal{H}}}(Q_+^s) \geq d^+_{\vec{\mathcal{H}}}(Z)$
  - $d^+_{\vec{\mathcal{H}}}(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d^+_{\vec{\mathcal{H}}}(Q_+^s)$ by definition.
- We can deduce that $d^+_{\vec{\mathcal{H}}}(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d^+_{\vec{\mathcal{H}}}(Q_+^s) \geq d^+_{\vec{\mathcal{H}}}(Z)$
  - $d^+_{\vec{\mathcal{H}}}(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d^+_{\vec{\mathcal{H}}}(Q_+^s)$ by definition.
- We can deduce that $d^+_{\vec{\mathcal{H}}}(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d^+_{\vec{\mathcal{H}}}(Q_+^s) \geq d^+_{\vec{\mathcal{H}}}(Z)$
  - $d^+_{\vec{\mathcal{H}}}(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d^+_{\vec{\mathcal{H}}}(Q_+^s)$ by definition.
- We can deduce that $d^+_{\vec{\mathcal{H}}}(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d^+_{\vec{\mathcal{H}}}(Q_+^s) \geq d^+_{\vec{\mathcal{H}}}(Z)$
  - $d^+_{\vec{\mathcal{H}}}(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d^+_{\vec{\mathcal{H}}}(Q_+^s)$ by definition.
- We can deduce that $d^+_{\vec{\mathcal{H}}}(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d^+_{\vec{\mathcal{H}}}(Q_+^s) \geq d^+_{\vec{\mathcal{H}}}(Z)$
  - $d^+_{\vec{\mathcal{H}}}(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d^+_{\vec{\mathcal{H}}}(Q_+^s)$ by definition.
- We can deduce that $d^+_{\vec{\mathcal{H}}}(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Existence of an hyperpath that does not leave Q_+^v

Lemma 12 (a)

$\forall s \in V, \forall t \in Q_+^s$, there exists an (s,t)-hyperpath that does not leave Q_+^s.

Proof of Lemma 12 (a)

- By contradiction, assume that there is $s \in V, t \in Q_+^s$ such that any (s,t)-hyperpath leaves Q_+^s.
- There is $s \in Z \subseteq Q_+^s \setminus \{t\}$ such that any hyperarc leaving Z will also leave Q_+^s.
- We have the following inequalities
  - $d^+_{\vec{\mathcal{H}}}(Q_+^s) \geq d^+_{\vec{\mathcal{H}}}(Z)$
  - $d^+_{\vec{\mathcal{H}}}(Z) \geq k$, as $\mathcal{H}$ is k-hyperarc-connected.
  - $k = d^+_{\vec{\mathcal{H}}}(Q_+^s)$ by definition.
- We can deduce that $d^+_{\vec{\mathcal{H}}}(Z) = k$, which automatically implies that $Z \in \mathcal{T}_+$.
- Q_+^s is not minimal, hence the contradiction.

Finding an admissible (s, t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_{-}$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_{-}$
   - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_{-}$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
   - F : (Directed) arborescence, rooted in s
   - Z : Explored (yet) vertices
   - V' : Allowed remaining vertices to explore

Finding an admissible (s, t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_{-}$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_{-}$
   - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_{-}$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
   - F : (Directed) arborescence, rooted in s
   - Z : Explored (yet) vertices
   - V' : Allowed remaining vertices to explore

Finding an admissible (s, t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_{-}$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_{-}$
   - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_{-}$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
   - F : (Directed) arborescence, rooted in s
   - Z : Explored (yet) vertices
   - V' : Allowed remaining vertices to explore

Finding an admissible (s, t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_-$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_-$
   - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_-$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
   - F : (Directed) arborescence, rooted in s
   - Z : Explored (yet) vertices
   - V' : Allowed remaining vertices to explore

Finding an admissible (s,t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_-$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_-$
   - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_-$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
   - F : (Directed) arborescence, rooted in s
   - Z : Explored (yet) vertices
   - V' : Allowed remaining vertices to explore

Finding an admissible (s, t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_{-}$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_{-}$
   - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_{-}$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
   - F : (Directed) arborescence, rooted in s
   - Z : Explored (yet) vertices
   - V' : Allowed remaining vertices to explore

Finding an admissible (s, t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_{-}$

1. Only input of the algorithm $R \in \mathcal{R} \in \mathcal{T}_{-}$
    - s, t are constrained (maybe not unique) by the choice of R.
2. Choosing $S \in \mathcal{M}_{-}$ such that $S \subseteq R$, then a safe source $s \in S$.
3. Main part of the algorithm : s-out arborescence
    - F : (Directed) arborescence, rooted in s
    - Z : Explored (yet) vertices
    - V' : Allowed remaining vertices to explore

Finding an admissible (s,t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_-$

Algorithm Admissible (s,t)-hyperpath in $R \in \mathcal{R} \cap \mathcal{T}_-$

```
Take a set S ∈ M_−, with S ⊆ R, then a safe source s ∈ S.
Z = {s}, F = (Z, ∅), V′ = R
while h = (X, v) exists such that v ∈ V′ − Z and X ∩ Z ≠ ∅ do
    Let u ∈ X ∩ Z.
    Z ← Z ∪ {v}
    F ← F + uv
    if Q_+^v ⊊ V′ then
        V′ ← Q_+^v
    end if
end while
T = V′
Take a safe sink t ∈ T
P′ = F[s, t]
P is the corresponding hyperpath in H⃗, obtained with P′.
Return S, T, s, t, P
```

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s,t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

A few words about complexity

- Computing $\mathcal{R}$, $\mathcal{M}_-$, $\mathcal{M}_+$ in polynomial time :
  - We can transform our hypergraph in a network (by trimming),
  - In which we can apply **Edmonds-Karp** to compute (s, t)-cuts.
  - It can be shown that $\mathcal{R}, \mathcal{M}_-, \mathcal{M}_+ \in \mathcal{Q}$, with $\mathcal{Q}$ obtained while applying **Edmonds-Karp**.
- Finding an *admissible* hyperpath runs in polynomial time :
  - Simple search that runs in polynomial time of number of hyperedges
  - Finding a safe source and safe sink in polynomial time thanks to **Edmonds-Karp**.
- The main algorithm runs in polynomial time.

Conclusion

Final words

- Generalization to hypergraphs of a previous article (by **Ito and al.**).
- Algorithmic Proof of Frank's characterisation of (k, k)-partition connected hypergraphs.
- Extensions of Theorem 7 : Starting without conditions on $\lambda(\vec{\mathcal{H}})$.

Conclusion

Final words

- Generalization to hypergraphs of a previous article (by **Ito and al.**).
- Algorithmic Proof of Frank's characterisation of (k, k)-partition connected hypergraphs.
- Extensions of Theorem 7 : Starting without conditions on $\lambda(\vec{\mathcal{H}})$.

Conclusion

Final words

- Generalization to hypergraphs of a previous article (by **Ito and al.**).
- Algorithmic Proof of Frank's characterisation of (k, k)-partition connected hypergraphs.
- Extensions of Theorem 7 : Starting without conditions on $\lambda(\vec{\mathcal{H}})$.

Conclusion

Thank you for your attention.