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 et al.*, 2023 :
  - Algorithmic proof of *Nash-Williams*, by flipping one edge 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 edge.
    - 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 et al.*, 2023 :
  - Algorithmic proof of *Nash-Williams*, by flipping one edge 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 edge.
    - The sequence can be obtained in polynomial time (in the size of the directed graph).

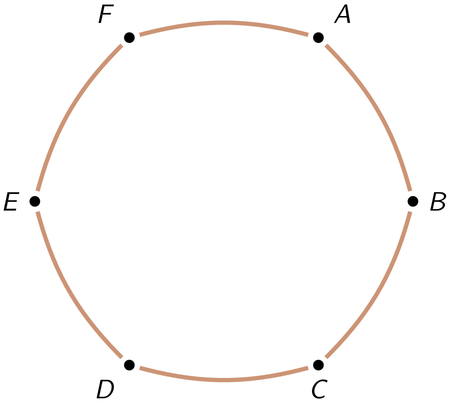
Goal of the article : Expanding the result of **Ito et al.** to hypergraphs.

State of the art, goal of the article

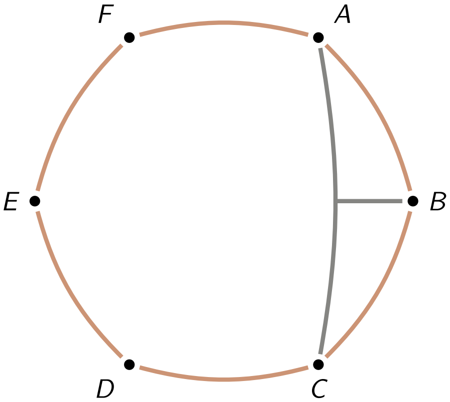
- *Nash-Williams*, 1960 :
  - G is $2k$-edge connected $\iff$ G admits a k-arc-connected orientation.
- *Ito et al.*, 2023 :
  - Algorithmic proof of *Nash-Williams*, by flipping one edge at a time.
  - Exhibiting a sequence of orientations such that :
    - The arc-connectivity does not decrease, and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

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

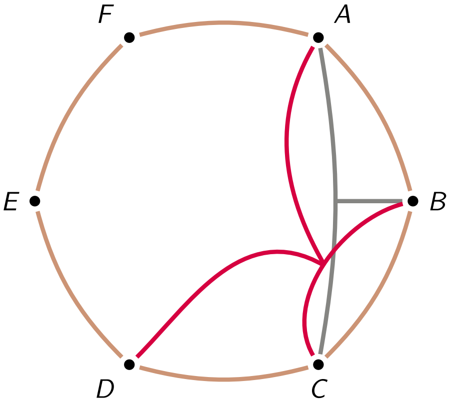
Hypergraphs



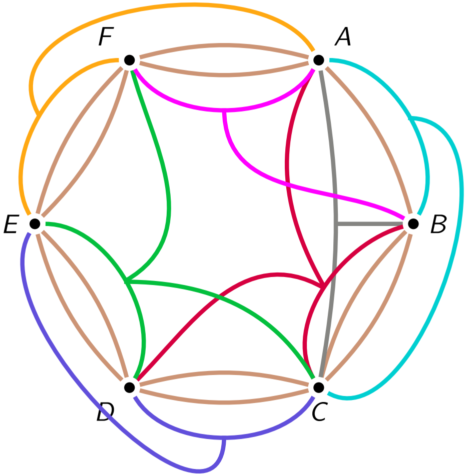
Hypergraphs



Hypergraphs

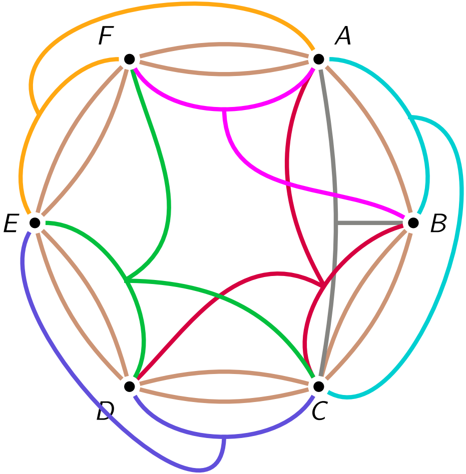


Hypergraphs



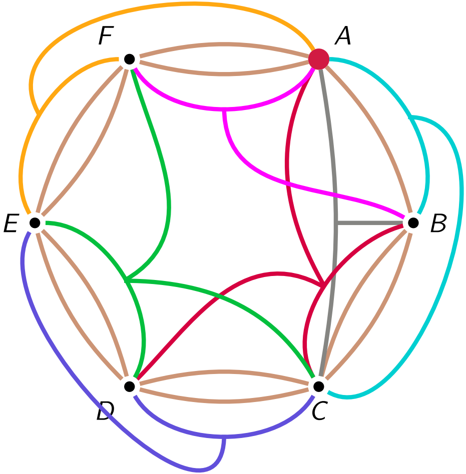
Degree of $\varnothing \neq X \subsetneq V$

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



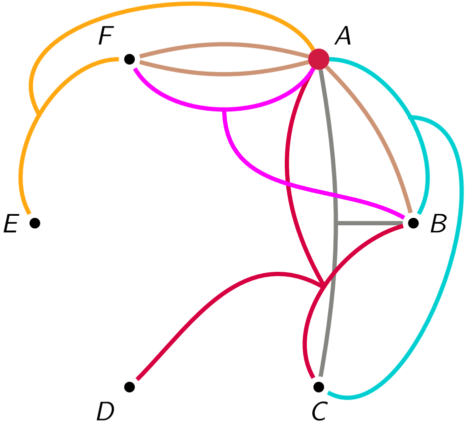
Degree of $\varnothing \neq X \subsetneq V$

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

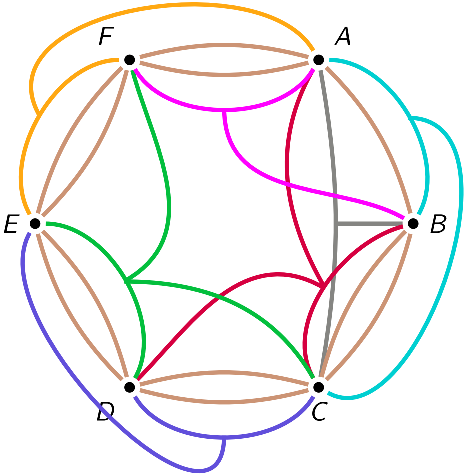


Degree of $\emptyset \neq X \subsetneq V$

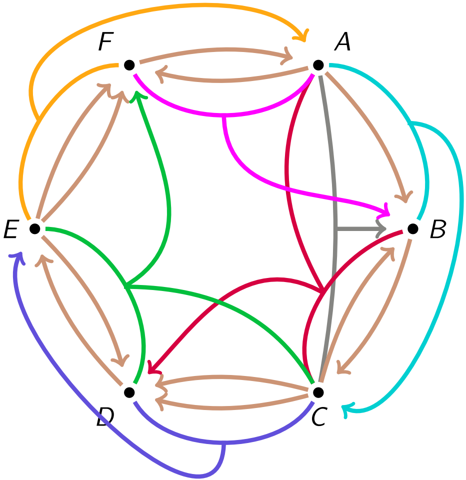
$d_{\mathcal{H}}(X)$ is the number of hyperedges intersecting both X and $V \setminus X$.



Orientation of an hypergraph

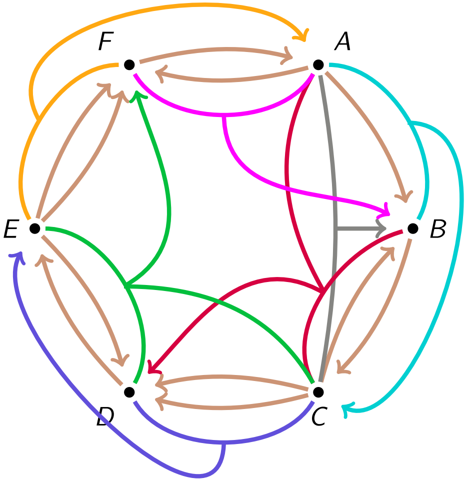


Orientation of an hypergraph



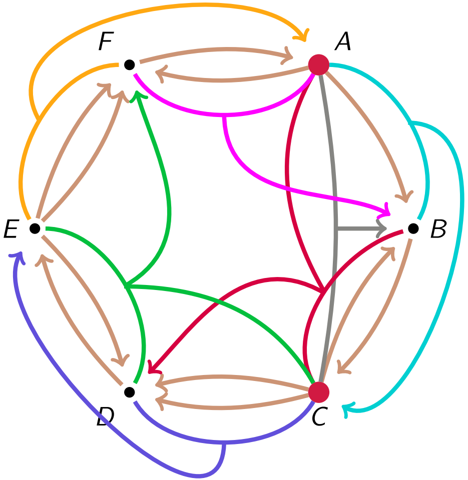
In-Degree of $\varnothing \neq X \subsetneq V$

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



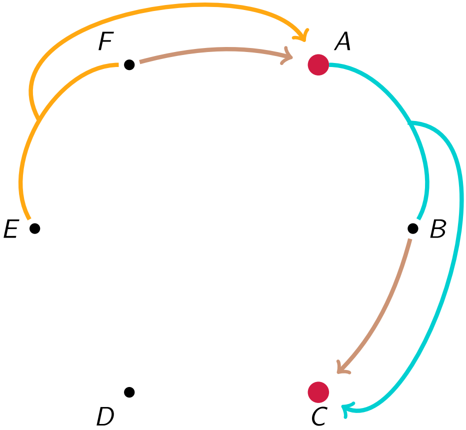
In-Degree of $\varnothing \neq X \subsetneq V$

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



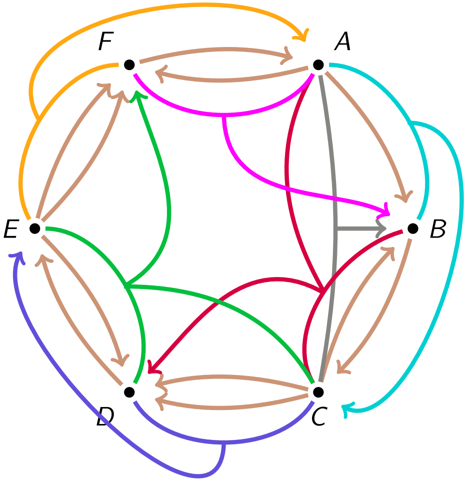
In-Degree of $\varnothing \neq X \subsetneq V$

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



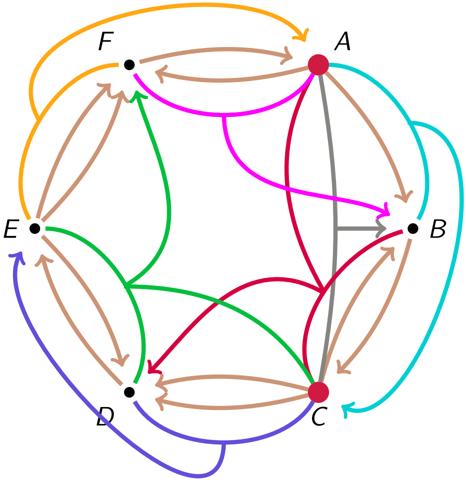
Out-Degree of $\varnothing \neq X \subsetneq V$

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



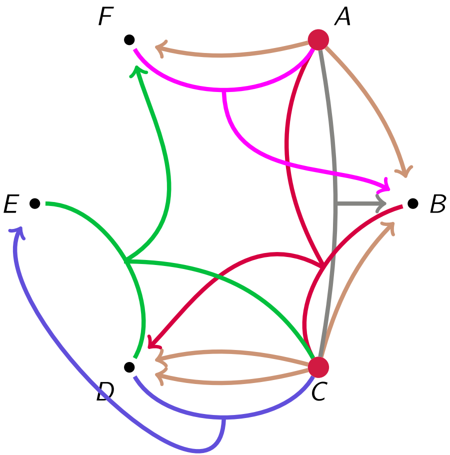
Out-Degree of $\varnothing \neq X \subsetneq V$

$d^+_{\mathcal{H}}(X)$ 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, $\forall e \in \mathcal{E}$, $d^+_{\vec{\mathcal{H}}}(e) \geq k$.
- The hyperarc-connectivity of a graph, 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, $\forall e \in \mathcal{E}$, $d^+_{\vec{\mathcal{H}}}(e) \geq k$.
- The hyperarc-connectivity of a graph, 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, $\forall e \in \mathcal{E}$, $d^+_{\vec{\mathcal{H}}}(e) \geq k$.
- The hyperarc-connectivity of a graph, 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. There is a 3-hyperarc-connected orientation

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

- $\vec{\mathcal{H}}$ is *k-hyperarc-connected*, if, $\forall e \in \mathcal{E}$, $d^+_{\vec{\mathcal{H}}}(e) \geq k$.
- The hyperarc-connectivity of a graph, 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, $\forall e \in \mathcal{E}$, $d^+_{\vec{\mathcal{H}}}(e) \geq k$.
- The hyperarc-connectivity of a graph, 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, $\forall e \in \mathcal{E}$, $d^+_{\vec{\mathcal{H}}}(e) \geq k$.
- The hyperarc-connectivity of a graph, 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 orientation of $\mathcal{H}$. Then there exists a sequence of hyperarcs $(\vec{\mathcal{H}}_i)_{i \in 0 \dots \ell}$ 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 $\mathcal{H}$).

Main result

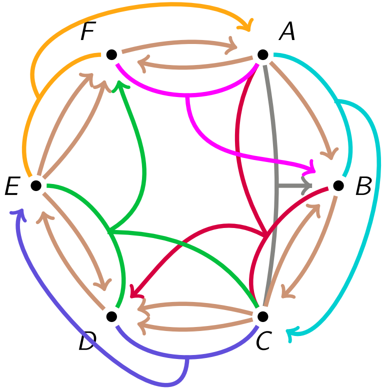
Main result (Theorem 7)

Let $\mathcal{H} = (V, E)$ be a $(k+1, k+1)$-partition-connected hypergraph and $\vec{\mathcal{H}}$ is a k-hyperarc orientation of $\mathcal{H}$. Then there exists a sequence of hyperarcs $(\vec{\mathcal{H}}_i)_{i \in 0 \ldots \ell}$ 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 $\mathcal{H}$).

Generalization of **Ito et al.**, as digraphs are special cases of hypergraphs.

"High-Level"-running of the algorithm

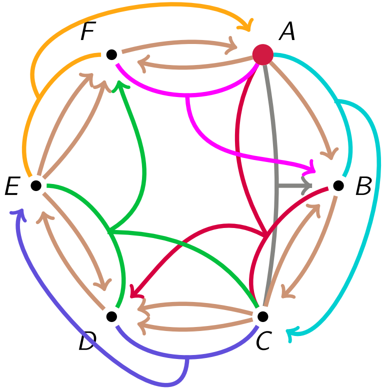
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

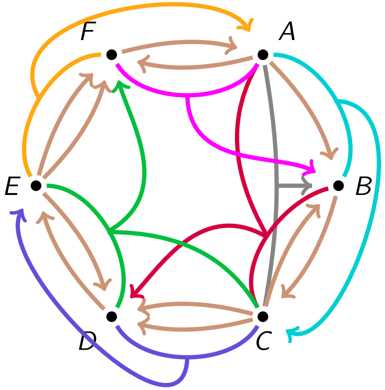
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

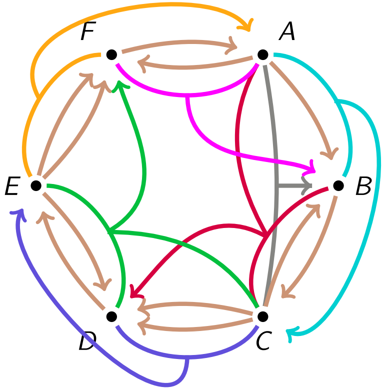
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

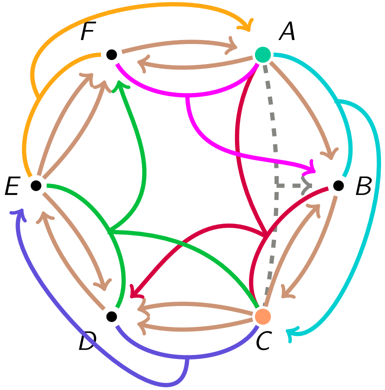
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

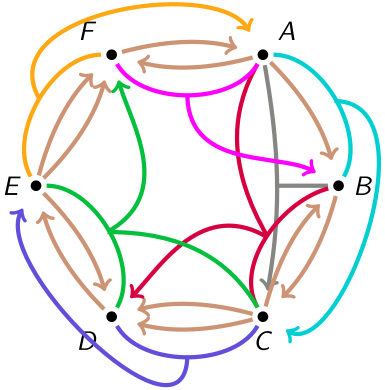
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

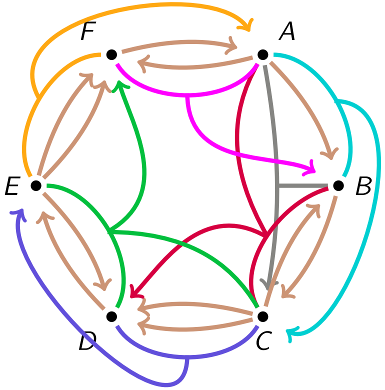
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

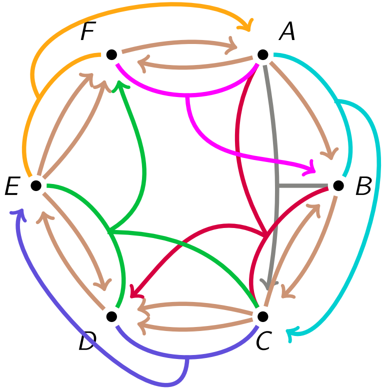
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

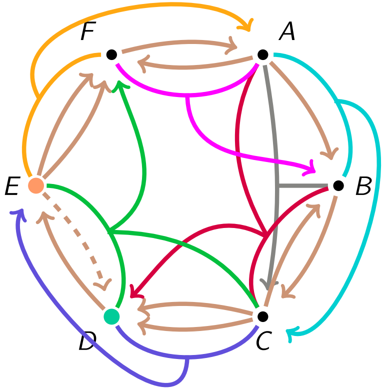
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

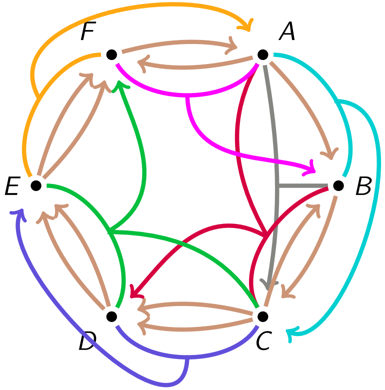
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

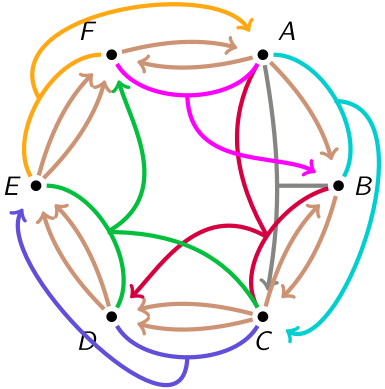
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

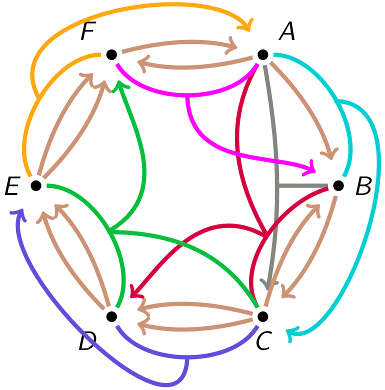
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

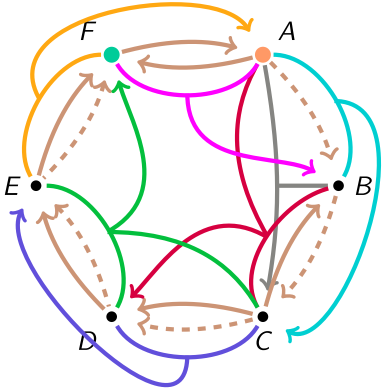
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

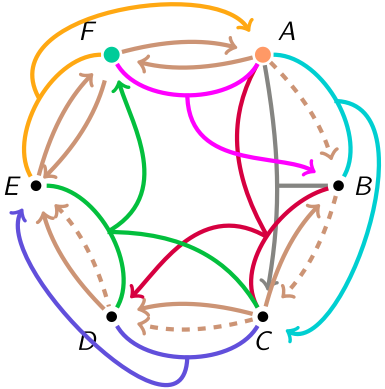
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

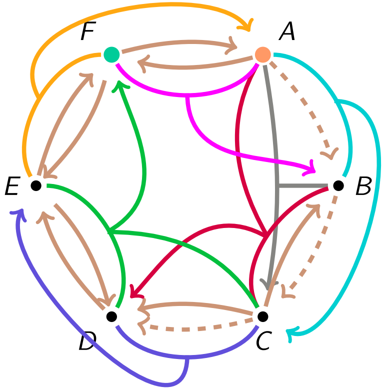
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

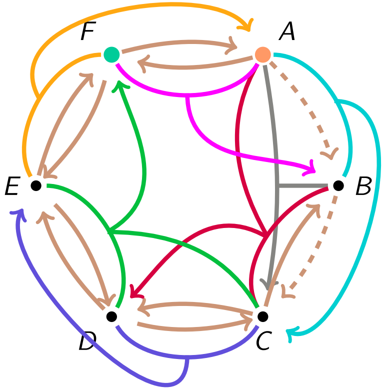
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

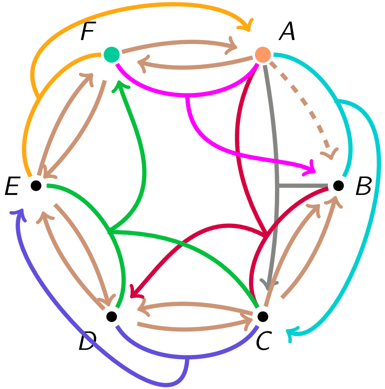
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

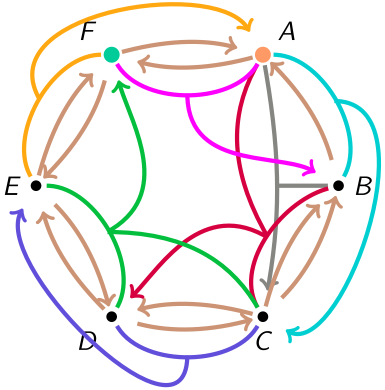
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

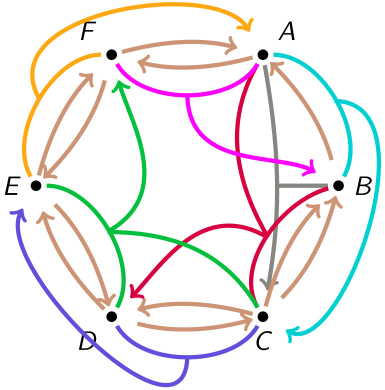
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

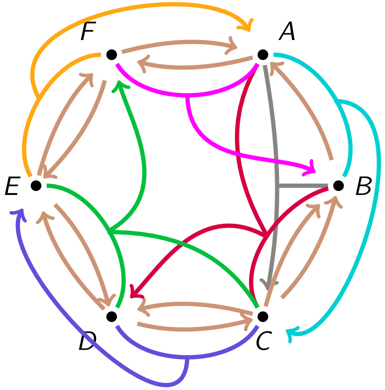
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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

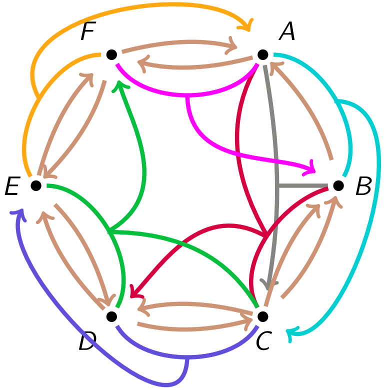
Our algorithm will provide a 3-hyperarc-connected orientation of $\mathcal{H}$, starting from a 2-hyperarc-connected.



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.

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.

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.

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.

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.

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.

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.

What are safe sources and safe sinks ?

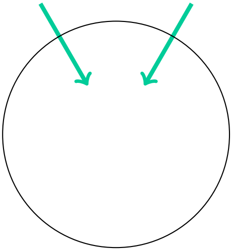
A brief detour...

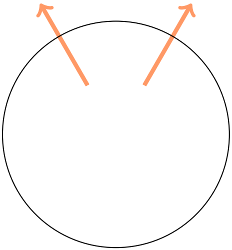
Tight and Dangerous sets

Remainder of the algorithm :

- `Input` : A k-hyperarc-connected orientation of a $(k+1, k+1)$-partition-connected hypergraph.
- `Output` : A $k+1$-hyperarc-connected hypergraph.

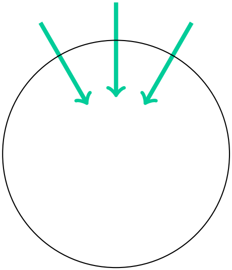
Tight and Dangerous sets


In-Tight sets

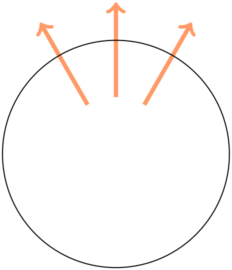

Out-Tight sets

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

Tight and Dangerous sets



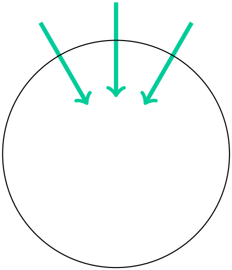
In-Dangerous sets



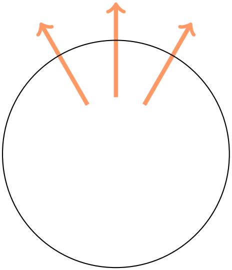
Out-Dangerous sets

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

Tight and Dangerous sets



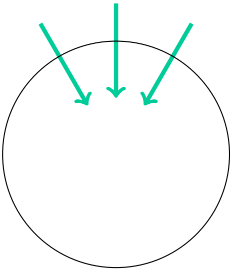
In-Dangerous sets



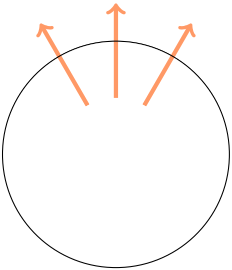
Out-Dangerous sets

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

Tight and Dangerous sets



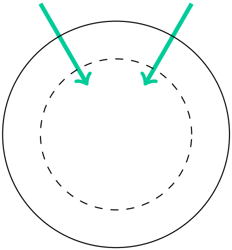
In-Dangerous sets



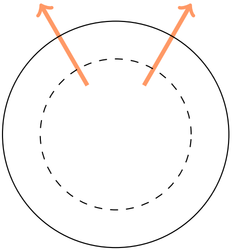
Out-Dangerous sets

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

Tight and Dangerous sets



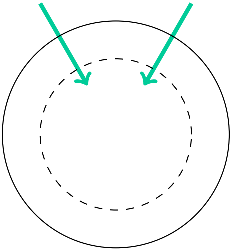
In-Tight sets



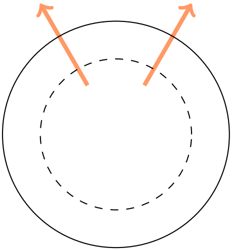
Out-Tight sets

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

Tight and Dangerous sets



Minimal In-Tight sets



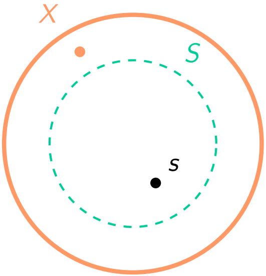
Minimal Out-Tight sets

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

Safe Sources and Safe Sinks

Definitions are symmetric (but proofs are not).

- For $S \in \mathcal{M}_-$, s is a safe source in S if :
  - a For every $s \in X \in \mathcal{T}_+$, we have $S \subsetneq X$.
  - b For every $s \in X \in \mathcal{D}_+$ such that $S \setminus X \neq \varnothing$, there exists $Y \in \mathcal{T}_+$ such that $s \notin Y \subsetneq X$.



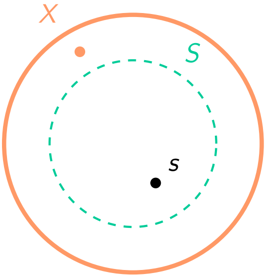
Condition (a)

Finding a safe sink t in $T \in \mathcal{M}_+$ can be done by checking each vertex if they correspond to the definition.

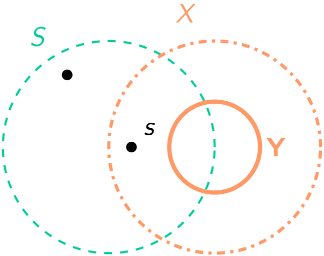
Safe Sources and Safe Sinks

Definitions are symmetric (but proofs are not).

- For $S \in \mathcal{M}_-$, s is a safe source in S if :
  - a For every $s \in X \in \mathcal{T}_+$, we have $S \subsetneq X$.
  - b For every $s \in X \in \mathcal{D}_+$ such that $S \setminus X \neq \varnothing$, there exists $Y \in \mathcal{T}_+$ such that $s \notin Y \subsetneq X$.



Condition (a)



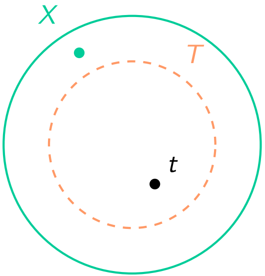
Condition (b)

Finding a safe sink t in $T \in \mathcal{M}_+$ can be done by checking each vertex if they correspond to the definition.

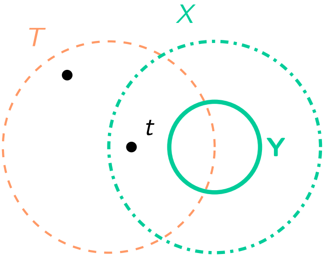
Safe Sources and Safe Sinks

Definitions are symmetric (but proofs are not).

- For $T \in \mathcal{M}_+$, t is a safe sink in T if :
  - c For every $t \in X \in \mathcal{T}_-$, we have $T \subsetneq X$.
  - d For every $t \in X \in \mathcal{D}_-$ such that $T \setminus X \neq \varnothing$, there exists $Y \in \mathcal{T}_-$ such that $t \notin Y \subsetneq X$.



Condition (c)



Condition (d)

Finding a safe sink t in $T \in \mathcal{M}_+$ can be done by checking each vertex if they correspond to the definition.

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 sketch 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.