Efficient Algorithms for Frequently Asked Questions

4. Hypertree Decompositions

Prof. Dan Olteanu

DaST

Data • (Systems+Theory)

University of Zurich UZH

March 14, 2022

https://lms.uzh.ch/url/RepositoryEntry/17185308706

FAQ Computation Time Depends on the Structure of the Hypergraph

Consider the following FAQ expression over the Boolean semiring:

$$\Phi() = \bigvee_{(x_1,\ldots,x_5)\in\prod_{i\in[5]}\mathsf{Dom}(X_i)} \psi_{12}(x_1,x_2) \wedge \psi_{23}(x_2,x_3) \wedge \psi_{34}(x_3,x_4) \wedge \psi_{15}(x_1,x_5)$$

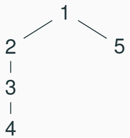
Φ asks whether there is a tuple $(x_1,\ldots,x_5)$ such that $\psi_{ij}(x_i,x_j) = \text{true}$

FAQ Computation Time Depends on the Structure of the Hypergraph

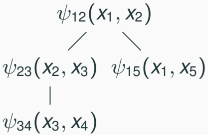
Consider the following FAQ expression over the Boolean semiring:

$$\Phi() = \bigvee_{(x_1,\ldots,x_5)\in\prod_{i\in[5]}\mathrm{Dom}(X_i)} \psi_{12}(x_1, x_2) \wedge \psi_{23}(x_2, x_3) \wedge \psi_{34}(x_3, x_4) \wedge \psi_{15}(x_1, x_5)$$

Φ asks whether there is a tuple $(x_1, \ldots, x_5)$ such that $\psi_{ij}(x_i, x_j) = \text{true}$



Hypergraph of Φ, all edges are binary

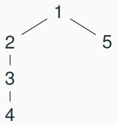


Possible bottom-up evaluation strategy

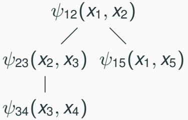
Evaluation strategy known for decades under different names:

- Message passing (in AI literature; Pearl'83)
- Semi-join reduction (in DB literature; Yannakakis'82; discussed in course)

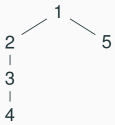
FAQ Computation Time Depends on the Structure of the Hypergraph



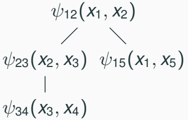
Hypergraph of Φ, all edges are binary



Possible bottom-up evaluation strategy



Hypergraph of Φ, all edges are binary

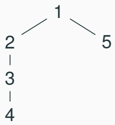


Possible bottom-up evaluation strategy

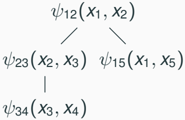
@ψ_{34} Send up its x_3-values:

$$V_{34\to 23}(x_3) = \bigvee_{x_4} \psi_{34}(x_3, x_4)$$

FAQ Computation Time Depends on the Structure of the Hypergraph



Hypergraph of Φ, all edges are binary



Possible bottom-up evaluation strategy

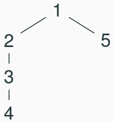
@ψ_{34} Send up its x_3-values:

$$V_{34\to 23}(x_3) = \bigvee_{x_4} \psi_{34}(x_3, x_4)$$

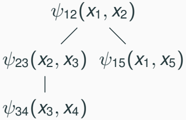
@ψ_{23} Send up its x_2-values that are paired with x_3 common to $V_{34\to 23}(x_3)$ and ψ_{23}:

$$V_{23\to 12}(x_2) = \bigvee_{x_3} \psi_{23}(x_2, x_3) \wedge V_{34\to 23}(x_3)$$

FAQ Computation Time Depends on the Structure of the Hypergraph



Hypergraph of Φ, all edges are binary



Possible bottom-up evaluation strategy

@ψ_{34} Send up its x_3-values:

$$V_{34\to 23}(x_3) = \bigvee_{x_4} \psi_{34}(x_3, x_4)$$

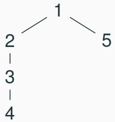
@ψ_{23} Send up its x_2-values that are paired with x_3 common to $V_{34\to 23}(x_3)$ and ψ_{23}:

$$V_{23\to 12}(x_2) = \bigvee_{x_3} \psi_{23}(x_2, x_3) \wedge V_{34\to 23}(x_3)$$

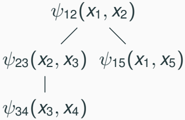
@ψ_{15} Send up its x_1-values:

$$V_{15\to 12}(x_1) = \bigvee_{x_5} \psi_{15}(x_1, x_5)$$

FAQ Computation Time Depends on the Structure of the Hypergraph



Hypergraph of Φ, all edges are binary



Possible bottom-up evaluation strategy

@ψ_{34} Send up its x_3-values:

$$V_{34\to 23}(x_3) = \bigvee_{x_4} \psi_{34}(x_3, x_4)$$

@ψ_{23} Send up its x_2-values that are paired with x_3 common to $V_{34\to 23}(x_3)$ and ψ_{23}:

$$V_{23\to 12}(x_2) = \bigvee_{x_3} \psi_{23}(x_2, x_3) \wedge V_{34\to 23}(x_3)$$

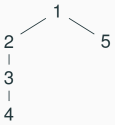
@ψ_{15} Send up its x_1-values:

$$V_{15\to 12}(x_1) = \bigvee_{x_5} \psi_{15}(x_1, x_5)$$

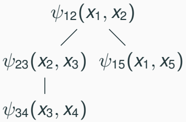
@ψ_{12} Is there a pair (x_1, x_2) of ψ_{12} with x_1 also in $V_{15\to 12}$ and x_2 also in $V_{23\to 12}$?

$$\Phi() = \bigvee_{x_1, x_2} \psi_{12}(x_1, x_2) \wedge V_{15\to 12}(x_1) \wedge V_{23\to 12}(x_2)$$

Computation Time



Hypergraph of Φ, all edges are binary



Possible bottom-up evaluation strategy

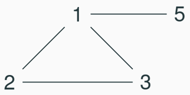
All computation steps are local and their cost upper bounded by the factor sizes

- Typical assumption: $|\psi_{ij}| \leq N$ for some value N
- We pass along at most N values between factors
- Local computation is just filtering local values with incoming values
- Overall: linear computation time – This is the best in worst case

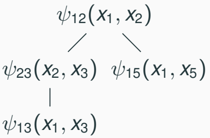
FAQ Computation for a Different Hypergraph

Now, consider a slightly different FAQ Φ': Same as Φ but $X_4 = X_1$

$$\Phi'() = \bigvee_{x_1, x_2, x_3, x_5} \psi_{12}(x_1, x_2) \wedge \psi_{23}(x_2, x_3) \wedge \psi_{13}(x_1, x_3) \wedge \psi_{15}(x_1, x_5)$$



Hypergraph of Φ', all edges are binary

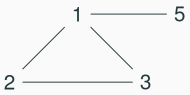


Possible bottom-up evaluation strategy

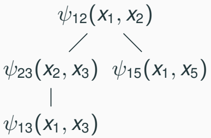
FAQ Computation for a Different Hypergraph

Now, consider a slightly different FAQ Φ': Same as Φ but $X_4 = X_1$

$$\Phi'() = \bigvee_{x_1, x_2, x_3, x_5} \psi_{12}(x_1, x_2) \wedge \psi_{23}(x_2, x_3) \wedge \psi_{13}(x_1, x_3) \wedge \psi_{15}(x_1, x_5)$$



Hypergraph of Φ', all edges are binary



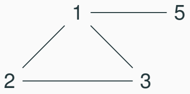
Possible bottom-up evaluation strategy

Computation not anymore local!

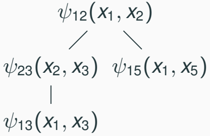
- x_1 needs to be propagated from ψ_{13} through ψ_{23} to ψ_{12}
- ψ_{23} does not have x_1, so it receives it and forwards it further
- This incurs the cost of carrying x_1 values along two computation steps

$\Rightarrow O(N^2)$ complexity (we will later learn how to do it in $O(N^{1.5})$)

FAQ Computation for a Different Hypergraph

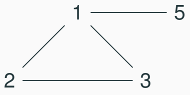


Hypergraph of Φ', all edges are binary

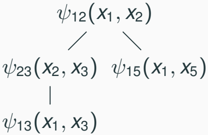


Possible bottom-up evaluation strategy

FAQ Computation for a Different Hypergraph



Hypergraph of Φ', all edges are binary

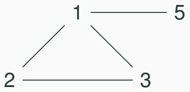


Possible bottom-up evaluation strategy

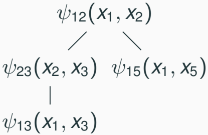
@ψ_{13} Send up (x_1, x_3)-values:

$$V_{13\to 23}(x_1, x_3) = \psi_{13}(x_1, x_3)$$

FAQ Computation for a Different Hypergraph



Hypergraph of Φ', all edges are binary



Possible bottom-up evaluation strategy

@ψ_{13} Send up (x_1, x_3)-values:

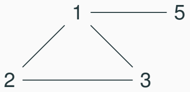
$$V_{13\to 23}(x_1, x_3) = \psi_{13}(x_1, x_3)$$

@ψ_{23} Send up (x_1, x_2) if there is x_3 such that $V_{13\to 23}(x_1, x_3)$ and $\psi_{23}(x_2, x_3)$:

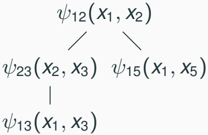
$$V_{23\to 12}(x_1, x_2) = \bigvee_{x_3} \psi_{23}(x_2, x_3) \wedge V_{13\to 23}(x_1, x_3)$$

Cost: $O(N^2)$

FAQ Computation for a Different Hypergraph



Hypergraph of Φ', all edges are binary



Possible bottom-up evaluation strategy

$@\psi_{13}$ Send up (x_1, x_3)-values:

$$V_{13\to 23}(x_1, x_3) = \psi_{13}(x_1, x_3)$$

$@\psi_{23}$ Send up (x_1, x_2) if there is x_3 such that $V_{13\to 23}(x_1, x_3)$ and $\psi_{23}(x_2, x_3)$:

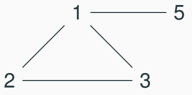
$$V_{23\to 12}(x_1, x_2) = \bigvee_{x_3} \psi_{23}(x_2, x_3) \wedge V_{13\to 23}(x_1, x_3)$$

Cost: $O(N^2)$

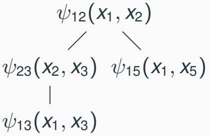
$@\psi_{15}$ Send up its x_1-values:

$$V_{15\to 12}(x_1) = \bigvee_{x_5} \psi_{15}(x_1, x_5)$$

FAQ Computation for a Different Hypergraph



Hypergraph of Φ', all edges are binary



Possible bottom-up evaluation strategy

@ψ_{13} Send up (x_1, x_3)-values:

$$V_{13\to 23}(x_1, x_3) = \psi_{13}(x_1, x_3)$$

@ψ_{23} Send up (x_1, x_2) if there is x_3 such that $V_{13\to 23}(x_1, x_3)$ and $\psi_{23}(x_2, x_3)$:

$$V_{23\to 12}(x_1, x_2) = \bigvee_{x_3} \psi_{23}(x_2, x_3) \wedge V_{13\to 23}(x_1, x_3)$$

Cost: $O(N^2)$

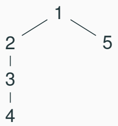
@ψ_{15} Send up its x_1-values:

$$V_{15\to 12}(x_1) = \bigvee_{x_5} \psi_{15}(x_1, x_5)$$

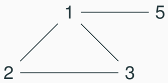
@ψ_{12} Is there (x_1, x_2) in ψ_{12} and in $V_{23\to 12}$ such that x_1 is also in $V_{15\to 12}$?

$$\Phi'() = \bigvee_{x_1, x_2} \psi_{12}(x_1, x_2) \wedge V_{15\to 12}(x_1) \wedge V_{23\to 12}(x_1, x_2)$$

Why is the Cost of the Second FAQ Higher than of the First One?



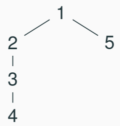
Acyclic Hypergraph



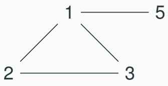
Cyclic Hypergraph

- Left: Only push up information of size $< N$ that is local at factor
- Right: Need to remember longer distance information and push it along
- The difference is reflected in the computational complexity: $O(N)$ vs $O(N^2)$

Why is the Cost of the Second FAQ Higher than of the First One?



Acyclic Hypergraph

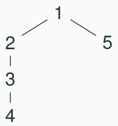


Cyclic Hypergraph

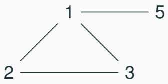
- Left: Only push up information of size $< N$ that is local at factor
- Right: Need to remember longer distance information and push it along
- The difference is reflected in the computational complexity: $O(N)$ vs $O(N^2)$

1. Can we distinguish syntactically the acyclic from the cyclic hypergraphs?

Why is the Cost of the Second FAQ Higher than of the First One?



Acyclic Hypergraph



Cyclic Hypergraph

- Left: Only push up information of size $< N$ that is local at factor
- Right: Need to remember longer distance information and push it along
- The difference is reflected in the computational complexity: $O(N)$ vs $O(N^2)$

1. Can we distinguish syntactically the acyclic from the cyclic hypergraphs?

2. Can we "transform" cyclic hypergraphs into acyclic ones?

Answer to Question 1: Acyclic Hypergraphs (Overview)

Several acyclicity notions exist. Studied in this course: α-acyclic & β-acyclic

FAQs without free variables can be computed in:

- Linear time in the size of input factors if its hypergraph is α-acyclic

  Assumption: Each factor ψ_S represented as list of tuples $\mathbf{x}_S$ with $\psi_S(\mathbf{x}_S) \neq \mathbf{0}$

- Linear time in the size of input factors if its hypergraph is β-acyclic

  Assumption: Each factor represented compactly as box, e.g., for (#)SAT

FAQs with free variables:

- *In principle* as above, BUT hypergraph is also free-connex
- Linear time for precomputation
- Then output the answer in constant time per tuple (enumeration delay)
- $\Rightarrow$ Linear time in input size plus output size

Answer to Question 2: Hypertree Decompositions (Overview)

Hypertree decompositions

- Transform an arbitrary hypergraph into a hypertree
- Measure of how close the hypergraph is to a hypertree: width
- Complexity of transformation is $\mathcal{O}(N^w)$, where
  - N is the maximal size of an input factor
  - w is the width of the hypergraph
- Once we have a hypertree $\longrightarrow$ see answer to Question 1

Hypertree Decompositions

Hypertree Decompositions: Definition

A hypertree decomposition $\mathcal{T}$ of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a pair (T, χ) with:

- T is a tree
- χ is a function mapping each node in T to a subset of $\mathcal{V}$ called *bag*

Properties of a decomposition $\mathcal{T} = (T, \chi)$:

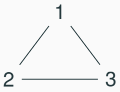
- *Coverage*: $\forall e \in \mathcal{E}$, there is a node $t \in T$ such that $e \subseteq \chi(t)$
- *Connectivity*: $\forall v \in \mathcal{V} : \{t \mid t \in T, v \in \chi(t)\}$ forms a connected subtree in $\mathcal{T}$

Observation: Each node $t \in T$ of the hypertree decomposition $\mathcal{T}$ represents the sub-hypergraph $\mathcal{H}'$ of $\mathcal{H}$ induced by the nodes $\chi(t)$ of $\mathcal{H}$

- The nodes of $\mathcal{H}'$ are $\chi(t)$
- The hyperedges of $\mathcal{H}'$ are $\mathcal{H}$'s hyperedges restricted to the nodes $\chi(t)$ of $\mathcal{H}$

Hypertree Decompositions for the Triangle Hypergraph

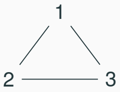
Triangle query: $\Phi(x_1, x_2, x_3) = \psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{13}(x_1, x_3)$



Task: Construct a hypertree decomposition
with one bag per edge

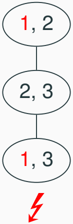
Hypertree Decompositions for the Triangle Hypergraph

Triangle query: $\Phi(x_1, x_2, x_3) = \psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{13}(x_1, x_3)$



Task: Construct a hypertree decomposition with one bag per edge

First trial

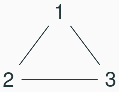


1 not included in the middle bag

Connectivity violated!

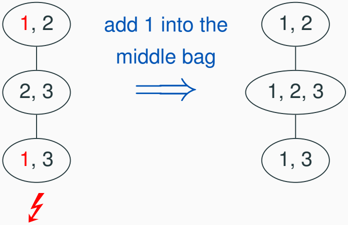
Hypertree Decompositions for the Triangle Hypergraph

Triangle query: $\Phi(x_1, x_2, x_3) = \psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{13}(x_1, x_3)$



Task: Construct a hypertree decomposition with one bag per edge

First trial

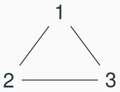


1 not included in the middle bag

Connectivity violated!

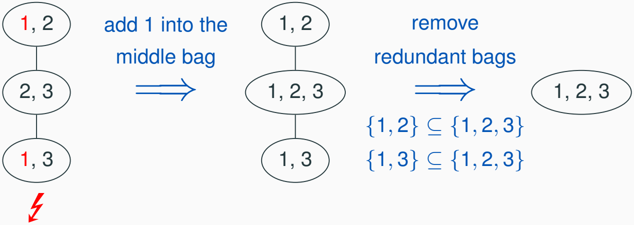
Hypertree Decompositions for the Triangle Hypergraph

Triangle query: $\Phi(x_1, x_2, x_3) = \psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{13}(x_1, x_3)$



Task: Construct a hypertree decomposition
with one bag per edge

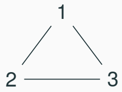
First trial



1 not included in the middle bag
Connectivity violated!

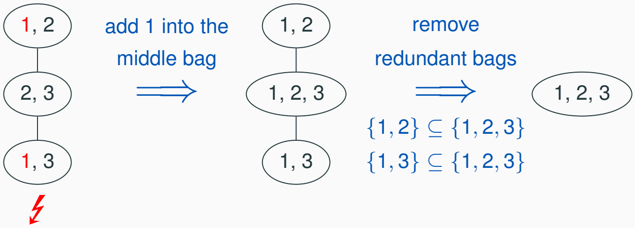
Hypertree Decompositions for the Triangle Hypergraph

Triangle query: $\Phi(x_1, x_2, x_3) = \psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{13}(x_1, x_3)$



Task: Construct a hypertree decomposition with one bag per edge

First trial



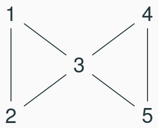
1 not included in the middle bag

Connectivity violated!

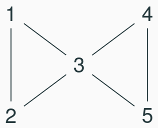
There are five other possibilities. All violate the connectivity condition.

The only hypertree decomposition without redundant bags has only one bag

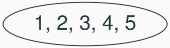
Hypertree Decompositions for the Bowtie Hypergraph



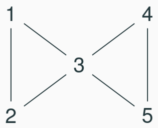
Hypertree Decompositions for the Bowtie Hypergraph



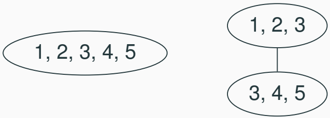
Possible hypertree decompositions



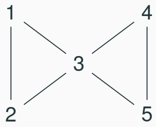
Hypertree Decompositions for the Bowtie Hypergraph



Possible hypertree decompositions



Hypertree Decompositions for the Bowtie Hypergraph

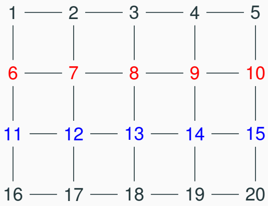


Possible hypertree decompositions

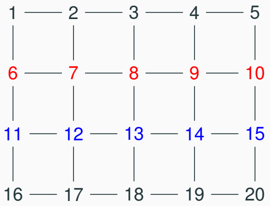
The bag $\{4, 5\}$ is redundant since it is included in bag $\{3, 4, 5\}$.

Redundant bags need not be considered as they add no extra information.

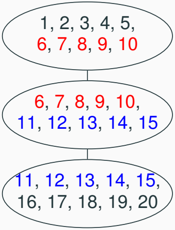
Hypertree Decompositions for the Grid Hypergraph



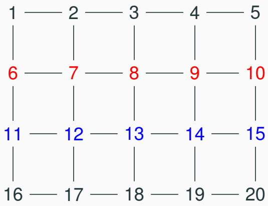
Hypertree Decompositions for the Grid Hypergraph



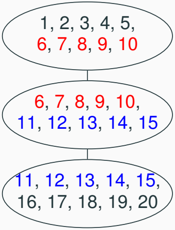
Possible hypertree decompositions

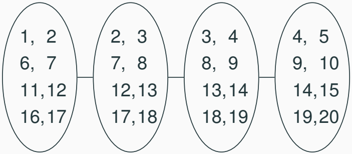


Hypertree Decompositions for the Grid Hypergraph



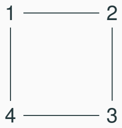
Possible hypertree decompositions





Hypertree Decompositions for 4-Cycle and 6-Cycle Hypergraphs

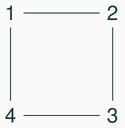
4-Cycle



Hypertree decompositions

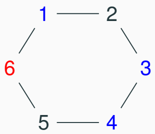
Hypertree Decompositions for 4-Cycle and 6-Cycle Hypergraphs

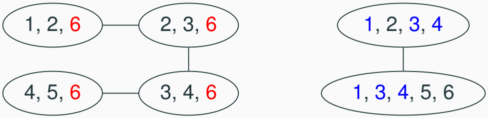
4-Cycle



Hypertree decompositions

6-Cycle



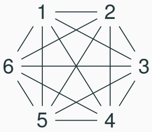


Hypertree Decompositions for Clique and Loomis-Whitney Hypergraphs

Clique of degree n :

Hypergraph $\left([n], \binom{[n]}{2}\right)$

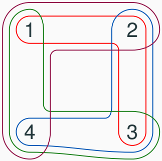
Clique-6



Loomis-Whitney of degree n:

Hypergraph $\left([n], \binom{[n]}{n-1}\right)$

Loomis-Whitney-4

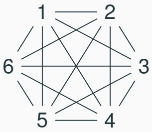


Hypertree Decompositions for Clique and Loomis-Whitney Hypergraphs

Clique of degree n :

Hypergraph $\left([n], \binom{[n]}{2}\right)$

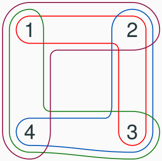
Clique-6



Loomis-Whitney of degree n:

Hypergraph $\left([n], \binom{[n]}{n-1}\right)$

Loomis-Whitney-4



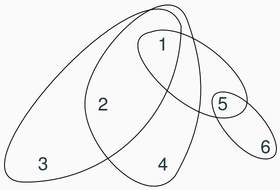
Both hypergraphs only admit the trivial decomposition with one bag

Join Trees: Hypertree Decompositions with One Bag per Hyperedge

α-acyclic hypergraphs admit hypertree decompositions with one bag per hyperdge

- Best decompositions, as no merging of factors in a bag is necessary
- Such decompositions are called Join Trees
- Hypergraphs are α-acyclic precisely when they admit join trees

Hypergraph

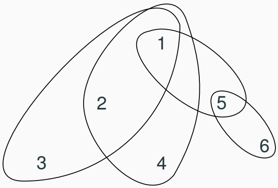


Join Trees: Hypertree Decompositions with One Bag per Hyperedge

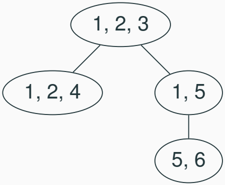
α-acyclic hypergraphs admit hypertree decompositions with one bag per hyperdge

- Best decompositions, as no merging of factors in a bag is necessary
- Such decompositions are called Join Trees
- Hypergraphs are α-acyclic precisely when they admit join trees

Hypergraph



Possible hypertree decomposition

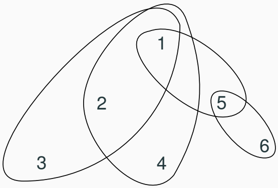


Join Trees: Hypertree Decompositions with One Bag per Hyperedge

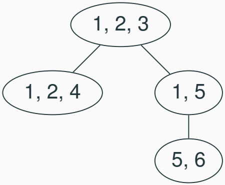
α-acyclic hypergraphs admit hypertree decompositions with one bag per hyperdge

- Best decompositions, as no merging of factors in a bag is necessary
- Such decompositions are called Join Trees
- Hypergraphs are α-acyclic precisely when they admit join trees

Hypergraph



Possible hypertree decomposition



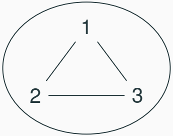
Question: Are α-acyclic precisely those hypergraphs without cycles?

α-acyclic Hypergraphs Can Have Cycles *IF* Covered by Hyperedges

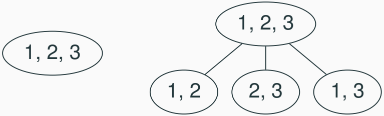
Consider the FAQ:

$$\Phi() = \bigoplus_{x_1, x_2, x_3} \psi_{123}(x_1, x_2, x_3) \otimes \psi_{12}(x_1, x_2) \otimes \psi_{13}(x_1, x_3) \otimes \psi_{23}(x_2, x_3)$$

Hypergraph



Hypertree decompositions

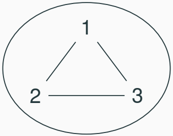


α-acyclic Hypergraphs Can Have Cycles *IF* Covered by Hyperedges

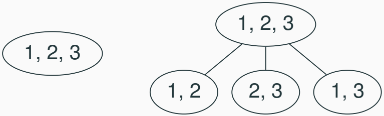
Consider the FAQ:

$$\Phi() = \bigoplus_{x_1, x_2, x_3} \psi_{123}(x_1, x_2, x_3) \otimes \psi_{12}(x_1, x_2) \otimes \psi_{13}(x_1, x_3) \otimes \psi_{23}(x_2, x_3)$$

Hypergraph



Hypertree decompositions



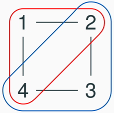
- Cycle formed by the factors ψ_{12}, ψ_{13}, and ψ_{23}
- BUT covered by the factor ψ_{123}
- We can evaluate Φ efficiently by absorbing each other factor into ψ_{123}
  $\psi_{123}(x_1, x_2, x_3) := \psi_{123}(x_1, x_2, x_3) \otimes \psi_{ij}(x_i, x_j)$, $(i, j) \in \{(1, 2), (1, 3), (2, 3)\}$

Non-trivial α-Acyclicity Example

$$\Phi() = \bigoplus_{x_1,x_2,x_3,x_4} \psi_{124}(x_1, x_2, x_4) \otimes \psi_{234}(x_2, x_3, x_4) \otimes$$
$$\psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{34}(x_3, x_4) \otimes \psi_{14}(x_1, x_4)$$

- Cycle formed by the factors ψ_{12}, ψ_{23}, ψ_{34}, and ψ_{14}
- BUT covered by the factors ψ_{124} and ψ_{234}
- We can evaluate Φ efficiently:
  - Absorb the factors ψ_{12} and ψ_{14} into the factor ψ_{124}
  - Absorb the factors ψ_{23} and ψ_{34} into the factor ψ_{234}
  - Multiply the factors ψ_{124} and ψ_{234} and aggregate away the variables

Hypergraph

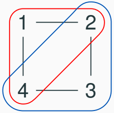


Non-trivial α-Acyclicity Example

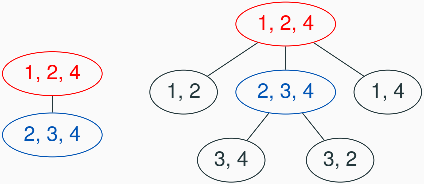
$$\Phi() = \bigoplus_{x_1,x_2,x_3,x_4} \psi_{124}(x_1, x_2, x_4) \otimes \psi_{234}(x_2, x_3, x_4) \otimes$$
$$\psi_{12}(x_1, x_2) \otimes \psi_{23}(x_2, x_3) \otimes \psi_{34}(x_3, x_4) \otimes \psi_{14}(x_1, x_4)$$

- Cycle formed by the factors ψ_{12}, ψ_{23}, ψ_{34}, and ψ_{14}
- BUT covered by the factors ψ_{124} and ψ_{234}
- We can evaluate Φ efficiently:
  - Absorb the factors ψ_{12} and ψ_{14} into the factor ψ_{124}
  - Absorb the factors ψ_{23} and ψ_{34} into the factor ψ_{234}
  - Multiply the factors ψ_{124} and ψ_{234} and aggregate away the variables

Hypergraph

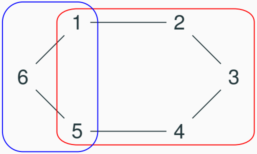


Hypertree decompositions (Second is join tree)



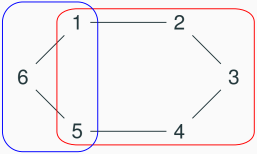
Further α-acyclicity Examples

Hypergraph

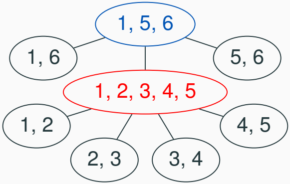


Further α-acyclicity Examples

Hypergraph

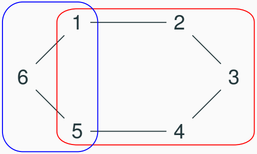


Hypertree decompositions (Join trees)

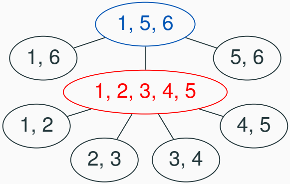


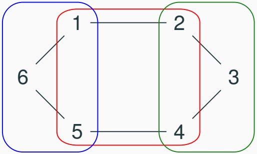
Further α-acyclicity Examples

Hypergraph



Hypertree decompositions (Join trees)

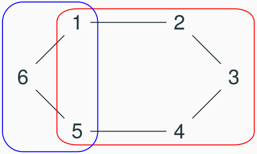


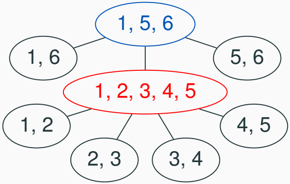


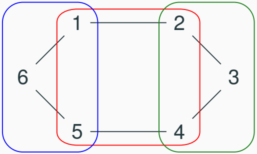
Further α-acyclicity Examples

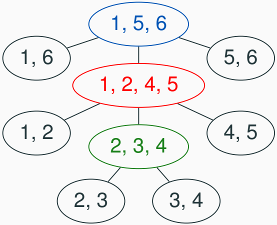
Hypergraph

Hypertree decompositions (Join trees)









The GYO Algorithm

The GYO (Graham, Yu, Ozsoyoglu) algorithm is used to decide α-acyclicity:

Input: Hypergraph $\mathcal{H}$

Output: Hypergraph obtained by repeating the following rules as long as possible:

- Eliminate a node that is contained in only one hyperedge
- Eliminate a hyperedge that is contained in another hyperedge

The GYO Algorithm

The GYO (Graham, Yu, Ozsoyoglu) algorithm is used to decide α-acyclicity:

Input: Hypergraph $\mathcal{H}$

Output: Hypergraph obtained by repeating the following rules as long as possible:

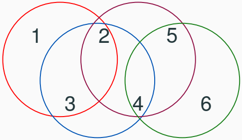
- Eliminate a node that is contained in only one hyperedge
- Eliminate a hyperedge that is contained in another hyperedge

$$\boxed{\mathcal{H} \text{ is } \alpha\text{-acyclic if and only if } \mathrm{GYO}(\mathcal{H}) = (\emptyset, \{\emptyset\})}$$

In words: $\mathcal{H}$ is α-acyclic if and only if the application of GYO to $\mathcal{H}$ returns a hypergraph with no vertices and one empty hyperedge

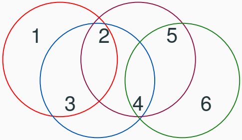
The GYO Algorithm: Example 1/3

initial hypergraph $\mathcal{H}$

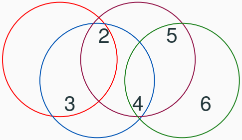


The GYO Algorithm: Example 1/3

initial hypergraph $\mathcal{H}$

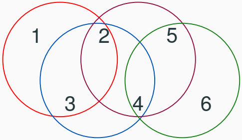


node 1 removed

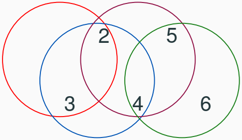


The GYO Algorithm: Example 1/3

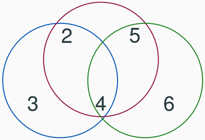
initial hypergraph $\mathcal{H}$



node 1 removed

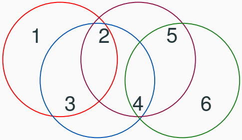


edge $\{2, 3\}$ removed

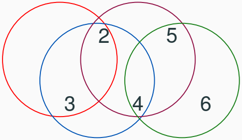


The GYO Algorithm: Example 1/3

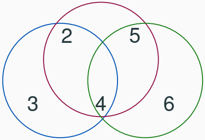
initial hypergraph $\mathcal{H}$



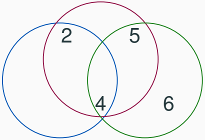
node 1 removed



edge $\{2, 3\}$ removed

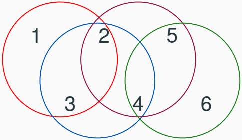


node 3 removed

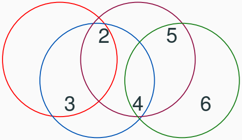


The GYO Algorithm: Example 1/3

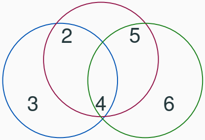
initial hypergraph $\mathcal{H}$



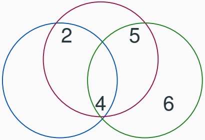
node 1 removed



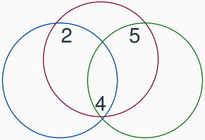
edge $\{2, 3\}$ removed



node 3 removed

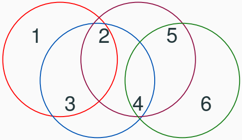


node 6 removed

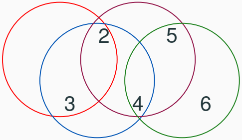


The GYO Algorithm: Example 1/3

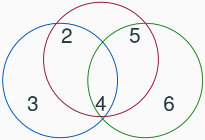
initial hypergraph $\mathcal{H}$



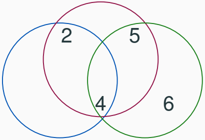
node 1 removed



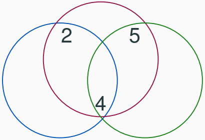
edge $\{2, 3\}$ removed



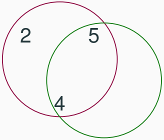
node 3 removed



node 6 removed

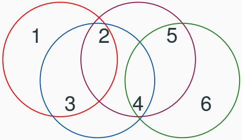


edge $\{2, 4\}$ removed

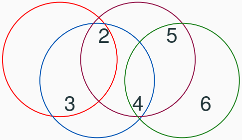


The GYO Algorithm: Example 1/3

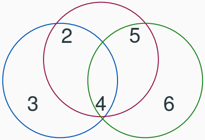
initial hypergraph $\mathcal{H}$



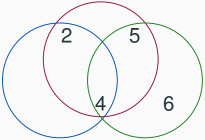
node 1 removed



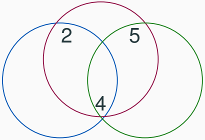
edge $\{2, 3\}$ removed



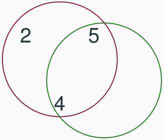
node 3 removed



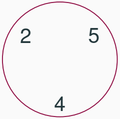
node 6 removed



edge $\{2, 4\}$ removed

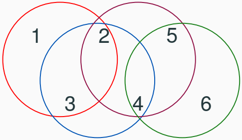


edge $\{4, 5\}$ removed

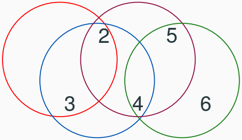


The GYO Algorithm: Example 1/3

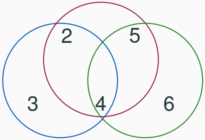
initial hypergraph $\mathcal{H}$



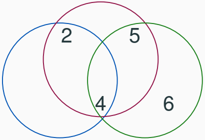
node 1 removed



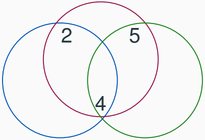
edge $\{2, 3\}$ removed



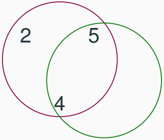
node 3 removed



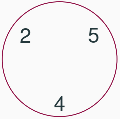
node 6 removed



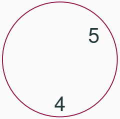
edge $\{2, 4\}$ removed



edge $\{4, 5\}$ removed

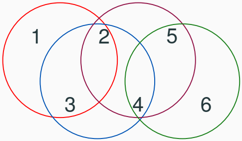


node 2 removed

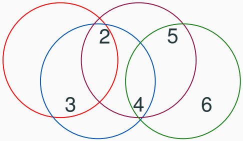


The GYO Algorithm: Example 1/3

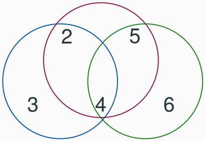
initial hypergraph $\mathcal{H}$



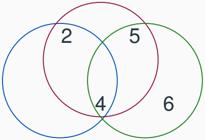
node 1 removed



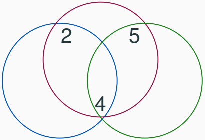
edge $\{2, 3\}$ removed



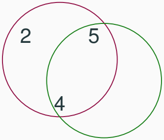
node 3 removed



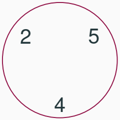
node 6 removed



edge $\{2, 4\}$ removed



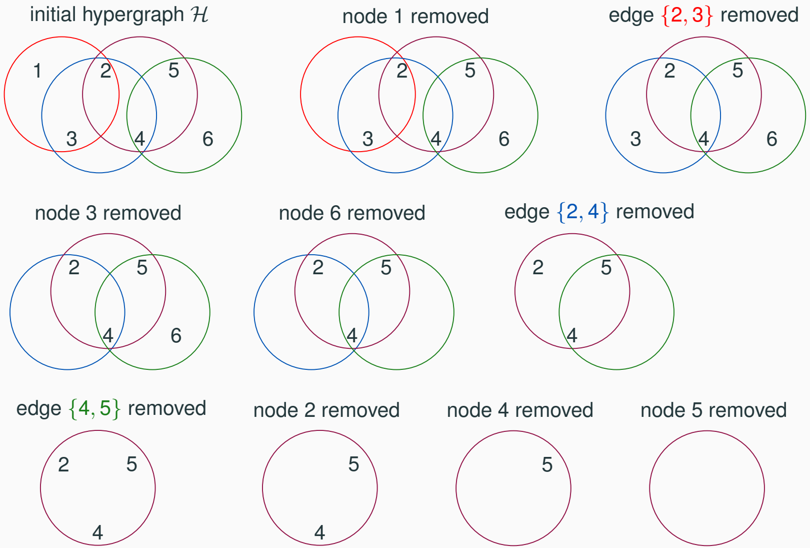
edge $\{4, 5\}$ removed



node 2 removed

node 4 removed

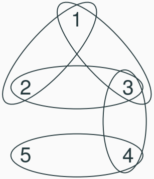
The GYO Algorithm: Example 1/3



$$\Longrightarrow \text{GYO}(\mathcal{H}) = (\emptyset, \{\emptyset\}) \Longrightarrow \mathcal{H} \text{ is } \alpha\text{-acyclic}$$

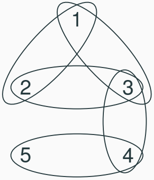
The GYO Algorithm: Example 2/3

initial hypergraph $\mathcal{H}$

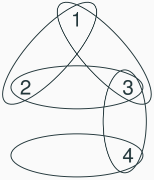


The GYO Algorithm: Example 2/3

initial hypergraph $\mathcal{H}$

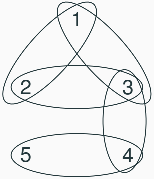


node 5 removed

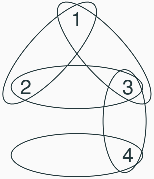


The GYO Algorithm: Example 2/3

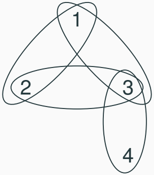
initial hypergraph $\mathcal{H}$



node 5 removed

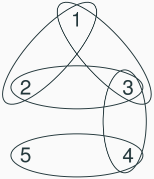


edge $\{4\}$ removed

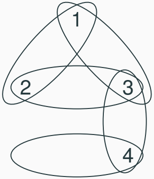


The GYO Algorithm: Example 2/3

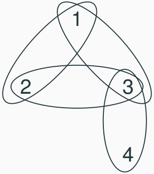
initial hypergraph $\mathcal{H}$



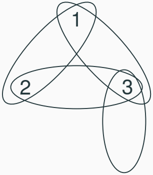
node 5 removed



edge $\{4\}$ removed

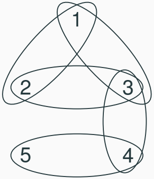


node 4 removed

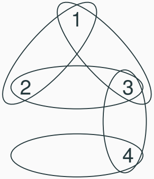


The GYO Algorithm: Example 2/3

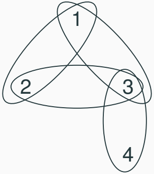
initial hypergraph $\mathcal{H}$



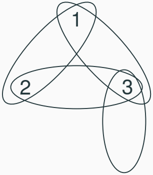
node 5 removed



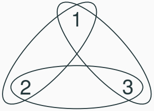
edge $\{4\}$ removed



node 4 removed

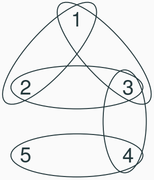


edge $\{3\}$ removed

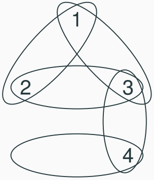


The GYO Algorithm: Example 2/3

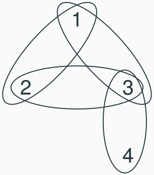
initial hypergraph $\mathcal{H}$



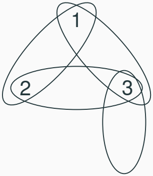
node 5 removed



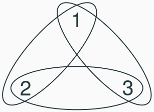
edge $\{4\}$ removed



node 4 removed



edge $\{3\}$ removed

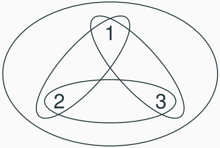


no more rule applicable

$\Longrightarrow$ GYO($\mathcal{H}$) $\neq (\emptyset, \{\emptyset\}) \Longrightarrow \mathcal{H}$ is not α-acyclic

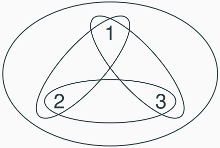
The GYO Algorithm: Example 3/3

initial hypergraph $\mathcal{H}$

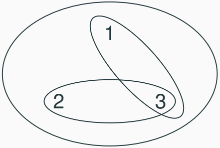


The GYO Algorithm: Example 3/3

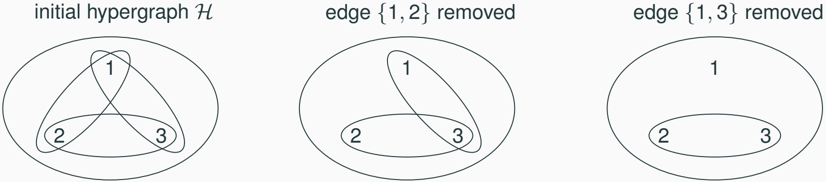
initial hypergraph $\mathcal{H}$



edge $\{1, 2\}$ removed

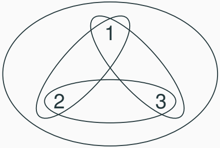


The GYO Algorithm: Example 3/3

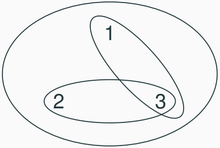


The GYO Algorithm: Example 3/3

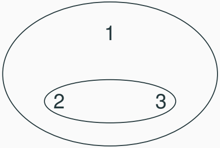
initial hypergraph $\mathcal{H}$



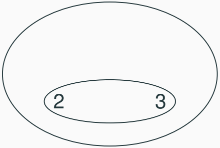
edge $\{1, 2\}$ removed



edge $\{1, 3\}$ removed

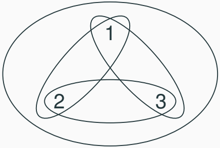


node 1 removed

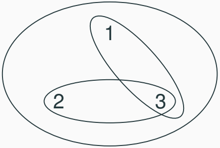


The GYO Algorithm: Example 3/3

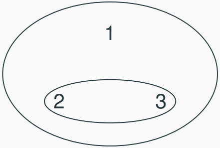
initial hypergraph $\mathcal{H}$



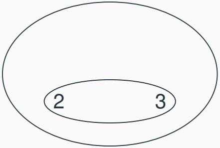
edge $\{1, 2\}$ removed



edge $\{1, 3\}$ removed



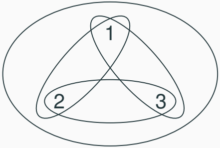
node 1 removed



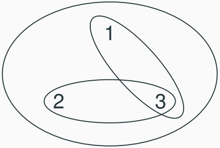
edge $\{2, 3\}$ removed

The GYO Algorithm: Example 3/3

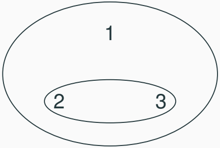
initial hypergraph $\mathcal{H}$



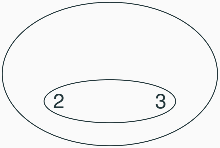
edge $\{1, 2\}$ removed



edge $\{1, 3\}$ removed



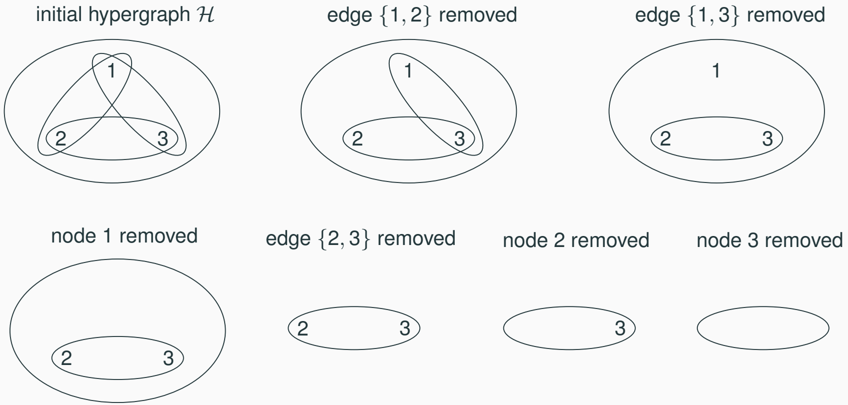
node 1 removed



edge $\{2, 3\}$ removed

node 2 removed

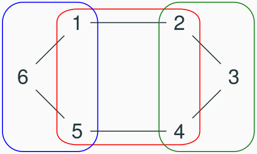
The GYO Algorithm: Example 3/3



$\Longrightarrow$ GYO($\mathcal{H}$) $= (\emptyset, \{\emptyset\}) \Longrightarrow \mathcal{H}$ is α-acyclic

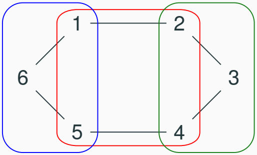
Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$

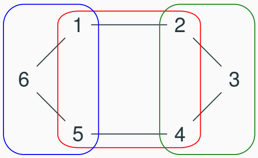


Algorithm

1. Compute *weighted graph* for $\mathcal{H}$

Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$

Weighted graph for $\mathcal{H}$:

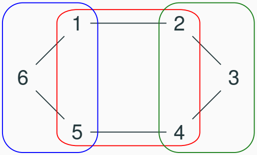
$\{1, 6\}$ $\{1, 2\}$

$\{1, 5, 6\}$ $\{5, 6\}$ $\{1, 2, 4, 5\}$ $\{2, 3\}$ $\{2, 3, 4\}$

$\{4, 5\}$ $\{3, 4\}$

Computing a Join Tree for an α-Acyclic Hypergraph

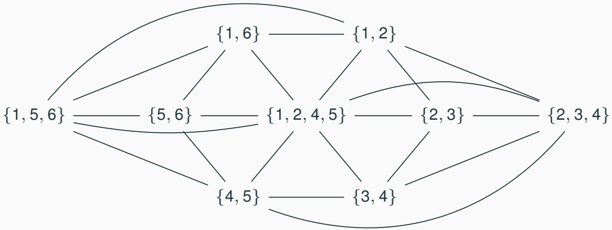
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

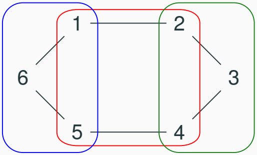
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

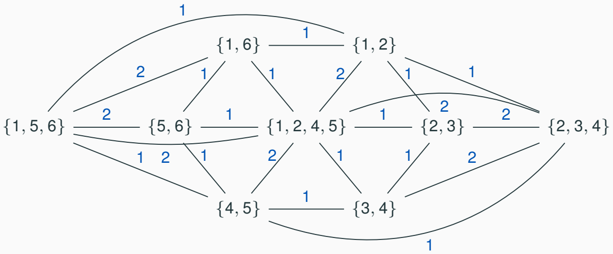
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

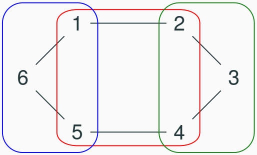
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

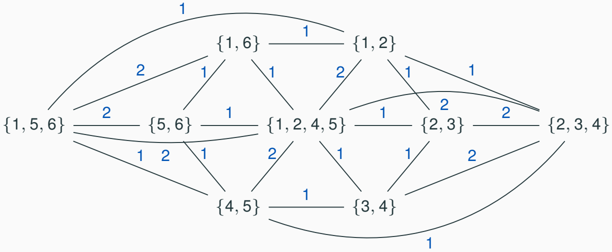
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

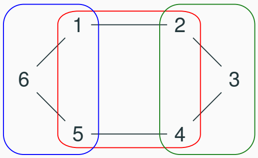
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

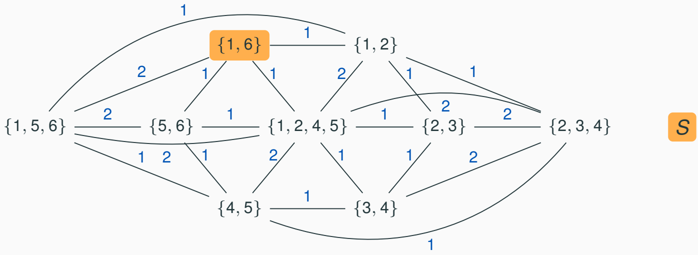
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

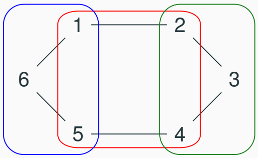
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

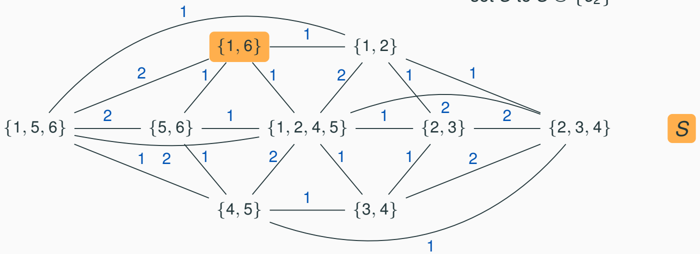
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

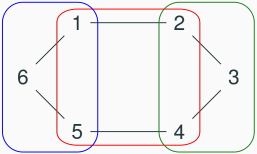
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

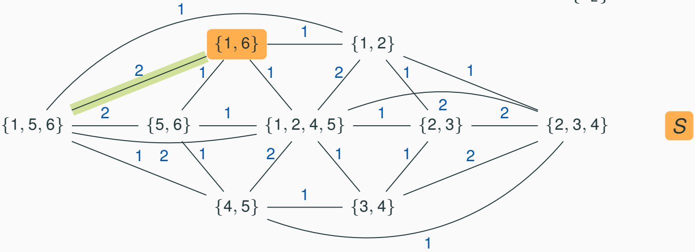
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

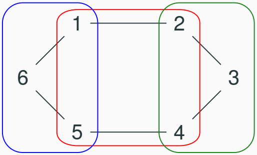
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

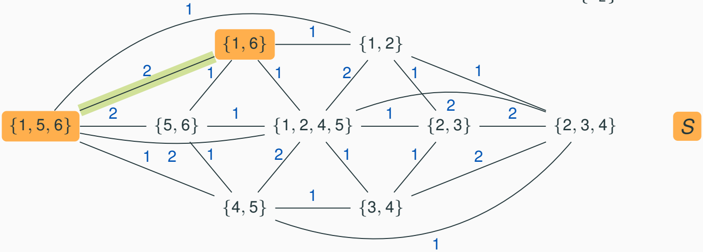
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

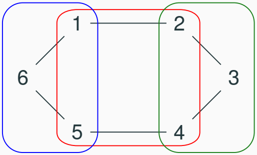
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

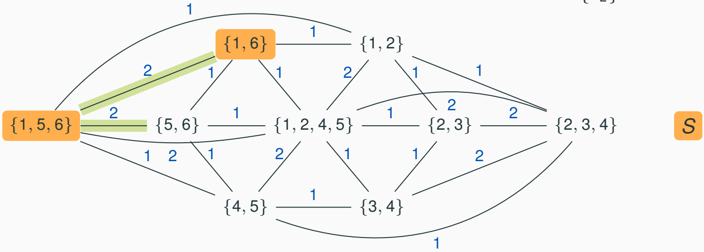
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

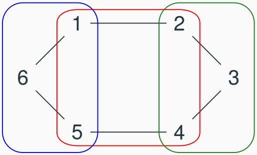
1. Compute *weighted graph* for $\mathcal{H}$
    - vertex set: $\mathcal{E}$
    - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
    - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
    - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
    - While $S \neq \mathcal{E}$:
        - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
        - (e_1, e_2) becomes spanning tree edge
        - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:

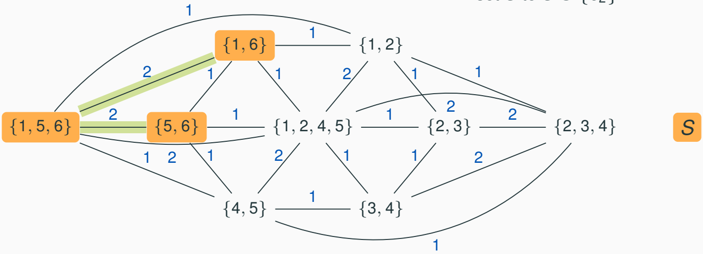


Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Weighted graph for $\mathcal{H}$:

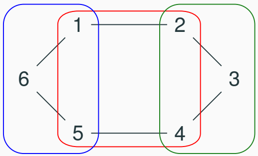


Algorithm

1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Computing a Join Tree for an α-Acyclic Hypergraph

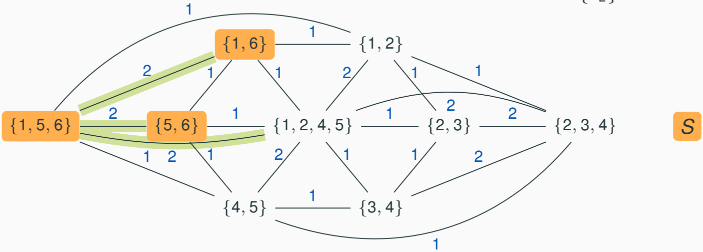
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

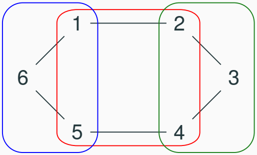
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

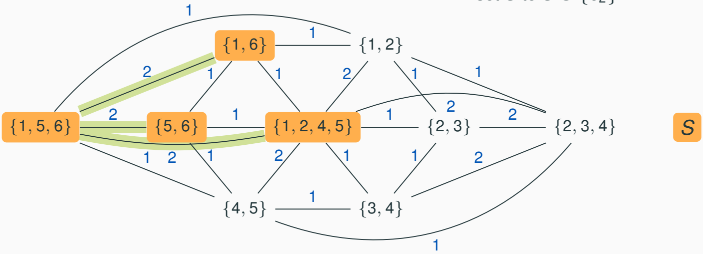
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

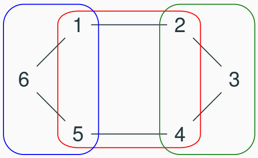
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

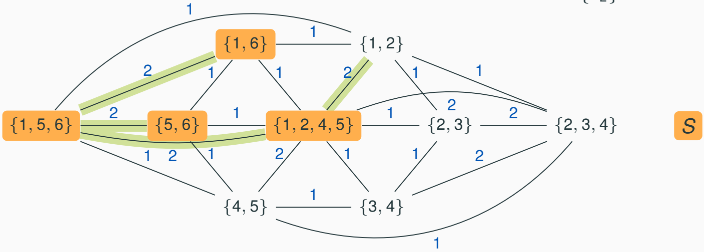
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

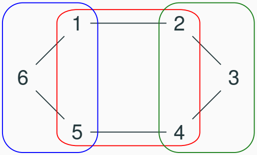
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

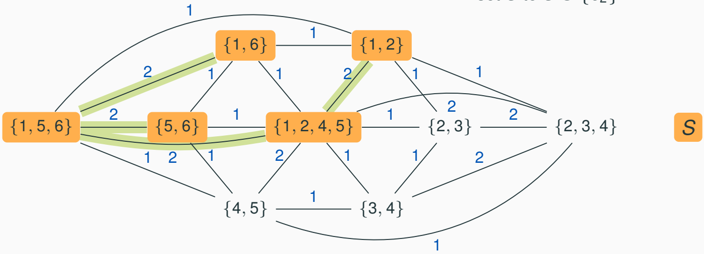
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

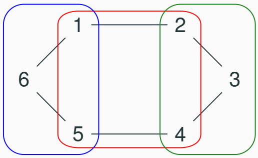
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:

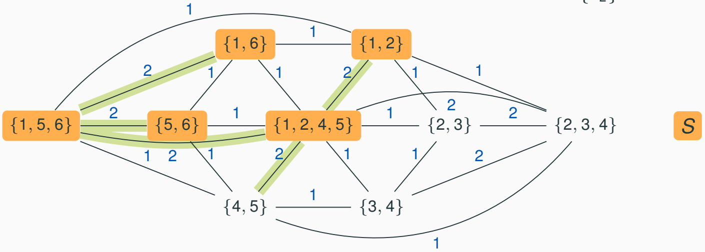


Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Weighted graph for $\mathcal{H}$:

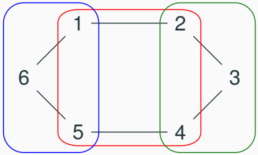


Algorithm

1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Computing a Join Tree for an α-Acyclic Hypergraph

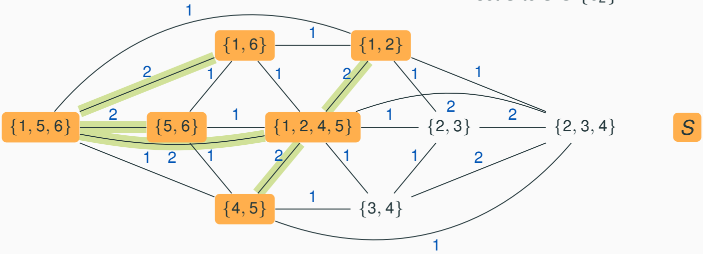
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

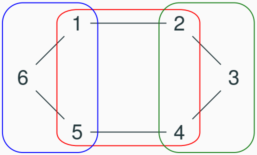
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

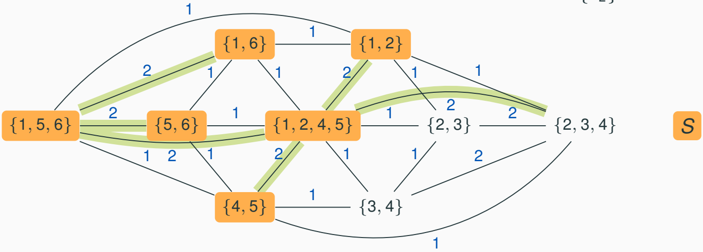
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

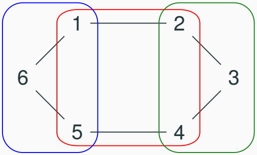
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

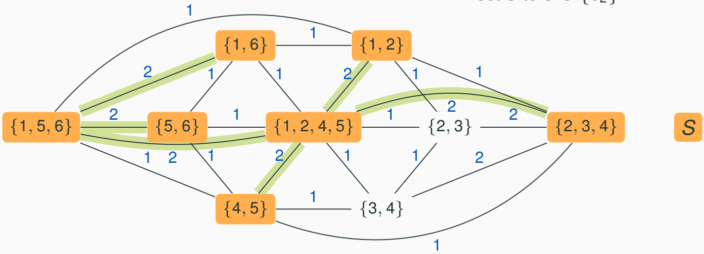
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

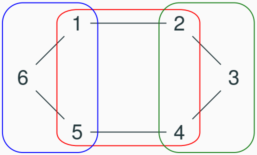
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

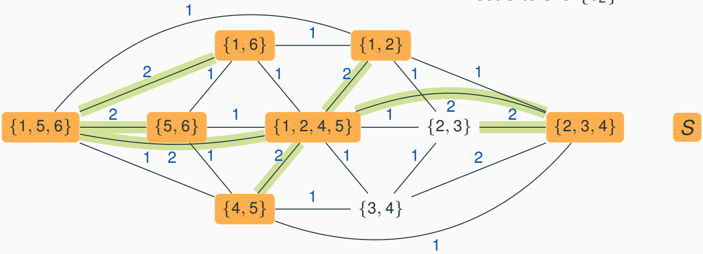
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

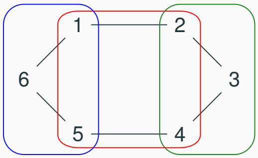
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

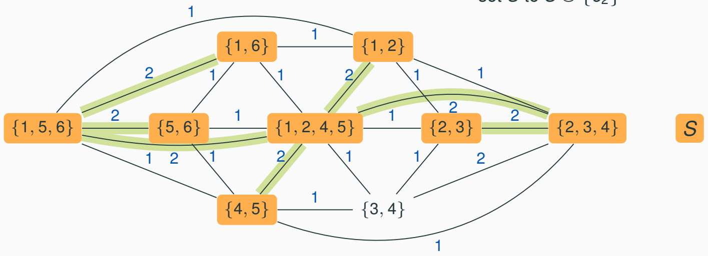
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

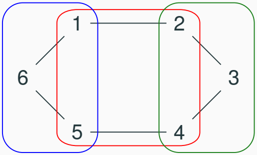
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

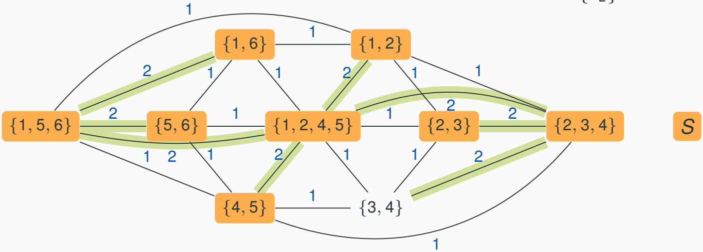
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

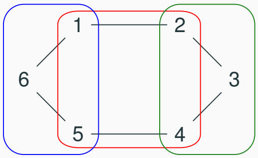
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



Computing a Join Tree for an α-Acyclic Hypergraph

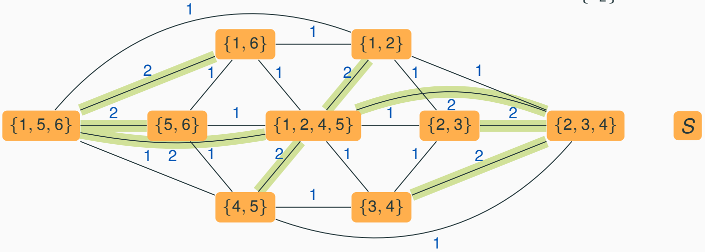
Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Algorithm

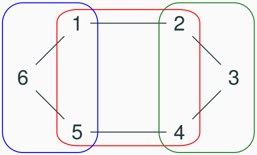
1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

Weighted graph for $\mathcal{H}$:



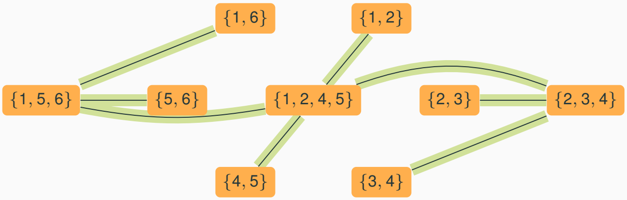
Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



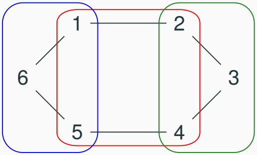
Algorithm

1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

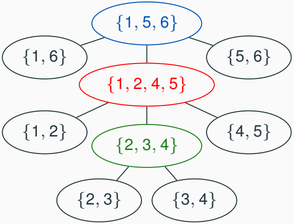


Computing a Join Tree for an α-Acyclic Hypergraph

Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$



Join tree for $\mathcal{H}$:



Algorithm

1. Compute *weighted graph* for $\mathcal{H}$
   - vertex set: $\mathcal{E}$
   - edge set: $\{(e_1, e_2) \in \mathcal{E}^2 \mid e_1 \cap e_2 \neq \emptyset\}$
   - weight of an edge (e_1, e_2): $|e_1 \cap e_2|$
2. Compute a maximum-weight spanning tree, e.g. using Prim's Algorithm
   - Start with $S = \{e\}$ for an arbitrary $e \in \mathcal{E}$
   - While $S \neq \mathcal{E}$:
     - choose edge (e_1, e_2) with $e_1 \in S$, $e_2 \notin S$ with maximal weight
     - (e_1, e_2) becomes spanning tree edge
     - set S to $S \cup \{e_2\}$

β-Acyclicity: Every Subgraph is Acyclic

$\boxed{\text{$\beta$-acyclicity is closed under hyperedge removal, α-acyclicity is not closed}}$

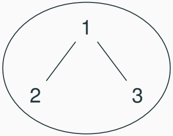
- α-acyclic hypergraphs may have cycles covered by hyperedges
  - The hypergraph without such hyperedges is not α-acyclic
- β-acyclic hypergraphs cannot have cycles
  - Removal of any hyperedges preserves β-acyclicity

β-Acyclicity: Every Subgraph is Acyclic

$\boxed{\beta\text{-acyclicity is closed under hyperedge removal, } \alpha\text{-acyclicity is not closed}}$

- α-acyclic hypergraphs may have cycles covered by hyperedges
  - The hypergraph without such hyperedges is not α-acyclic
- β-acyclic hypergraphs cannot have cycles
  - Removal of any hyperedges preserves β-acyclicity

Hypergraph

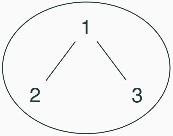


β-Acyclicity: Every Subgraph is Acyclic

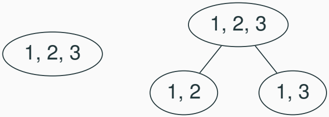
β-acyclicity is closed under hyperedge removal, α-acyclicity is not closed

- α-acyclic hypergraphs may have cycles covered by hyperedges
  - The hypergraph without such hyperedges is not α-acyclic
- β-acyclic hypergraphs cannot have cycles
  - Removal of any hyperedges preserves β-acyclicity

Hypergraph



Hypertree decompositions



Free-connex α-Acyclicity

- α-acyclicity is prerequisite to efficient computation
- FAQ compute time also depends on free variables $X_1, \ldots, X_f$

$$\Phi(\mathbf{x}_{[f]}) = \bigoplus_{x_{f+1}}^{(f+1)} \cdots \bigoplus_{x_n}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

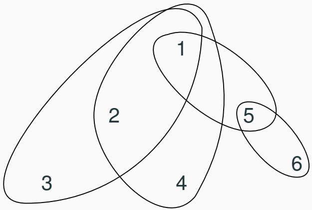
- Free-connex property: There is a join tree for the hypergraph of Φ, where
  - All free variables $[f]$ appear in nodes that form a connected subtree
  - Each bound variable either occurs in nodes without free variables or only in one node with free variables
  - $\to$ All bound variables can be aggregated away in linear time

When is an FAQ with hypergraph $\mathcal{H}$ and free variables $[f]$ free-connex α-acyclic?

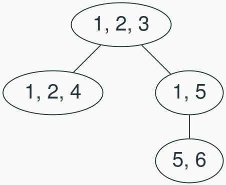
$\mathcal{H}$ remains α-acyclic even after adding the hyperedge $[f]$ over the free variables

Free-connex α-Acyclicity Example 1/2

α-acyclic hypergraph $\mathcal{H}$

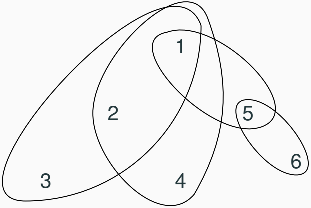


Possible join tree for $\mathcal{H}$

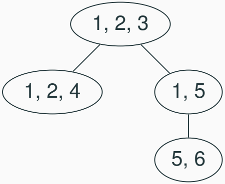


Free-connex α-Acyclicity Example 1/2

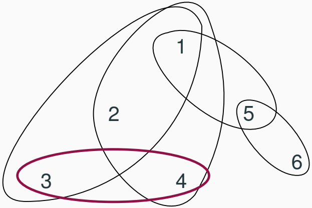
α-acyclic hypergraph $\mathcal{H}$



Possible join tree for $\mathcal{H}$

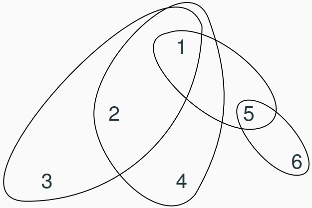


choose $\{3, 4\}$
as free variables

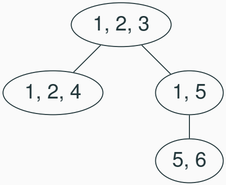


Free-connex α-Acyclicity Example 1/2

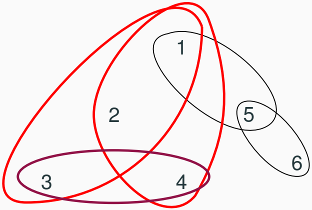
α-acyclic hypergraph $\mathcal{H}$



Possible join tree for $\mathcal{H}$



choose $\{3, 4\}$
as free variables



$\Longrightarrow$ Add the hyperedge $\{3, 4\}$ to the hypergraph

$\Longrightarrow$ There is a cycle not covered by a hyperedge

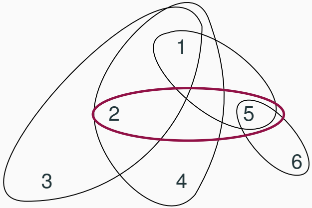
$\Longrightarrow$ Extended hypergraph not α-acyclic

$\Longrightarrow$ No join tree for extended hypergraph

$\Longrightarrow$ Original hypergraph not free-connex α-acyclic

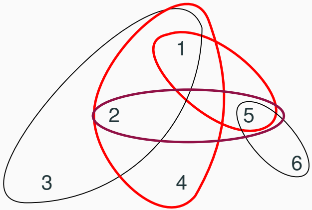
Free-connex α-Acyclicity Example 2/2

choose $\{2, 5\}$
as free variables



Free-connex α-Acyclicity Example 2/2

choose $\{2, 5\}$
as free variables



$\implies$ Add the hyperedge $\{2, 5\}$ to the hypergraph

$\implies$ There is a cycle not covered by a hyperedge

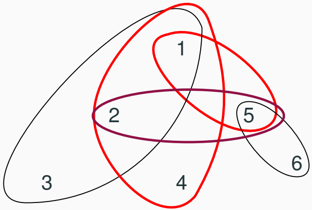
$\implies$ Extended hypergraph not α-acyclic

$\implies$ No join tree for extended hypergraph

$\implies$ Original hypergraph not free-connex α-acyclic

Free-connex α-Acyclicity Example 2/2

choose $\{2, 5\}$
as free variables



$\Longrightarrow$ Add the hyperedge $\{2, 5\}$ to the hypergraph

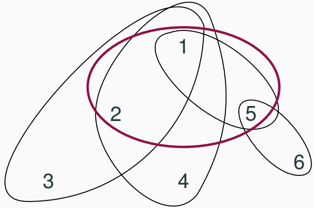
$\Longrightarrow$ There is a cycle not covered by a hyperedge

$\Longrightarrow$ Extended hypergraph not α-acyclic

$\Longrightarrow$ No join tree for extended hypergraph

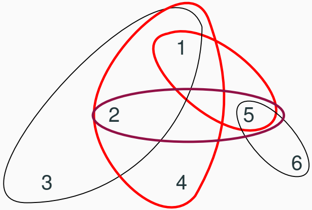
$\Longrightarrow$ Original hypergraph not free-connex α-acyclic

choose $\{1, 2, 5\}$
as free variables



Free-connex α-Acyclicity Example 2/2

choose $\{2, 5\}$
as free variables



$\Longrightarrow$ Add the hyperedge $\{2, 5\}$ to the hypergraph

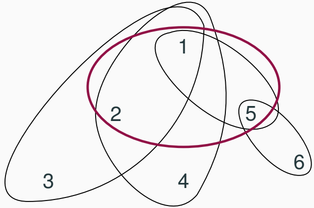
$\Longrightarrow$ There is a cycle not covered by a hyperedge

$\Longrightarrow$ Extended hypergraph not α-acyclic

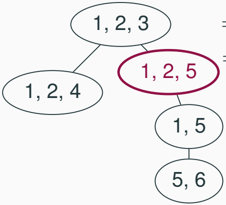
$\Longrightarrow$ No join tree for extended hypergraph

$\Longrightarrow$ Original hypergraph not free-connex α-acyclic

choose $\{1, 2, 5\}$
as free variables



Possible join tree for the extended hypergraph



$\Longrightarrow$ Extended hypergraph α-acyclic

$\Longrightarrow$ Original hypergraph
free-connex α-acyclic

Landscape of Hypergraph Types