A New Hypergraph Measure for #SAT

Florent Capelli
CRIL, Université d'Artois

Dagstuhl Seminar 24421

October 17, 2024

Loosely based on Direct Access for Conjunctive Queries with Negation with Oliver Irwin, ICDT 24

Structural Tractability of #SAT

The #SAT problem

Given CNF F, return $\#F$, the number of satisfying assignments.

- #P-hard to solve.
- Even for **very restricted** classes: #Mon-2-SAT, #Horn-SAT etc.
- NP-hard to (even badly) approximate (see ApproxMC for practical work in this direction).

The #SAT problem

Given CNF F, return $\#F$, the number of satisfying assignments.

- #P-hard to solve.
- Even for **very restricted** classes: #Mon-2-SAT, #Horn-SAT etc.
- NP-hard to (even badly) approximate (see ApproxMC for practical work in this direction).

Same story as SAT: hard problem but useful in practice.

- Reasoning on propositionnal knowledge basis.
- Solve other counting problems using parcimonious reductions.

When can we solve #SAT more efficiently than bruteforce?

Exploiting clauses-variables interactions

Tractability of #SAT arises from restricted **clauses-variables interactions**.

If $X \cap Y = \emptyset$ and $F(X, z_1, z_2, Y) \equiv G(X, z_1, z_2) \wedge H(Y, z_1, z_2)$:

Exploiting clauses-variables interactions

Tractability of #SAT arises from restricted **clauses-variables interactions**.

If $X \cap Y = \emptyset$ and $F(X, z_1, z_2, Y) \equiv G(X, z_1, z_2) \wedge H(Y, z_1, z_2)$:

$$\#F = \sum_{a,b \in \{0,1\}^2} \#G[z_1 = a, z_2 = b] \cdot \#H[z_1 = a, z_2 = b]$$

Exploiting clauses-variables interactions

Tractability of #SAT arises from restricted **clauses-variables interactions**.

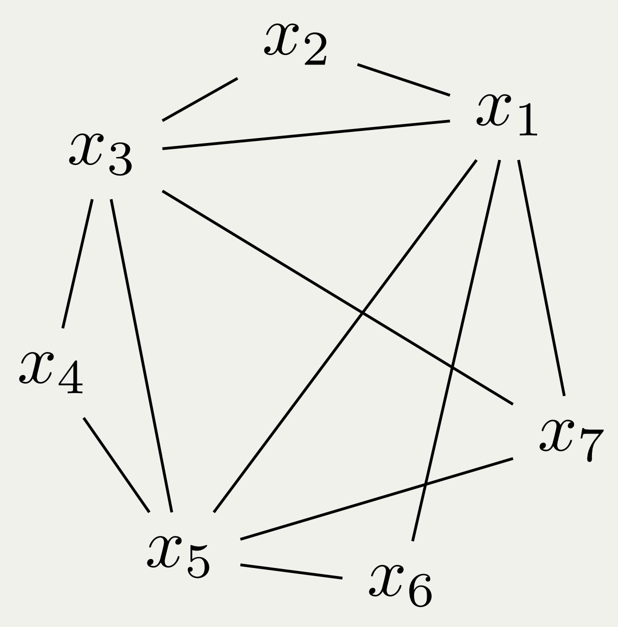
If $X \cap Y = \emptyset$ and $F(X, z_1, z_2, Y) \equiv G(X, z_1, z_2) \wedge H(Y, z_1, z_2)$:

$$\#F = \sum_{a, b \in \{0,1\}^2} \#G[z_1 = a, z_2 = b] \cdot \#H[z_1 = a, z_2 = b]$$

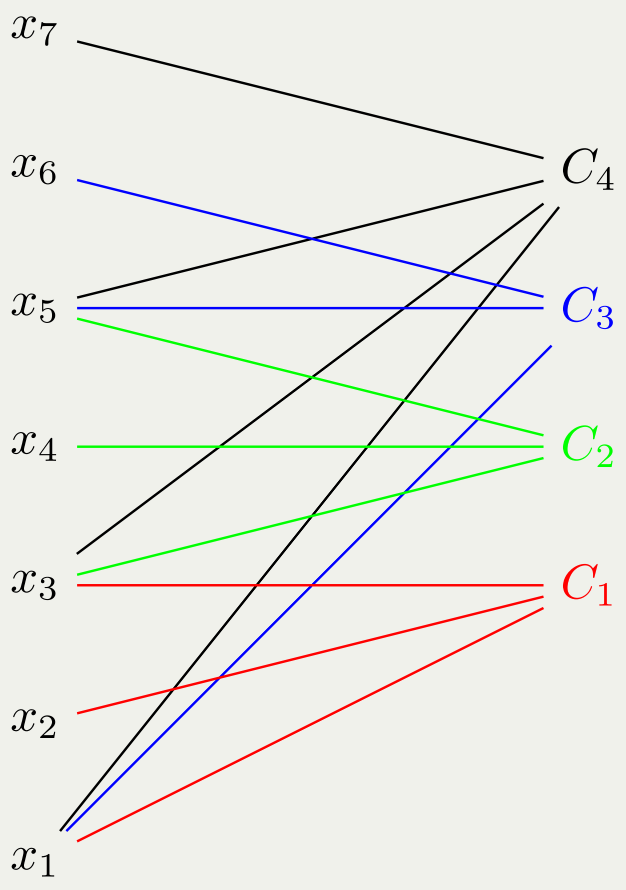
If we can recursively decompose the formula this way, we can count efficiently.

Structure of CNF formulas

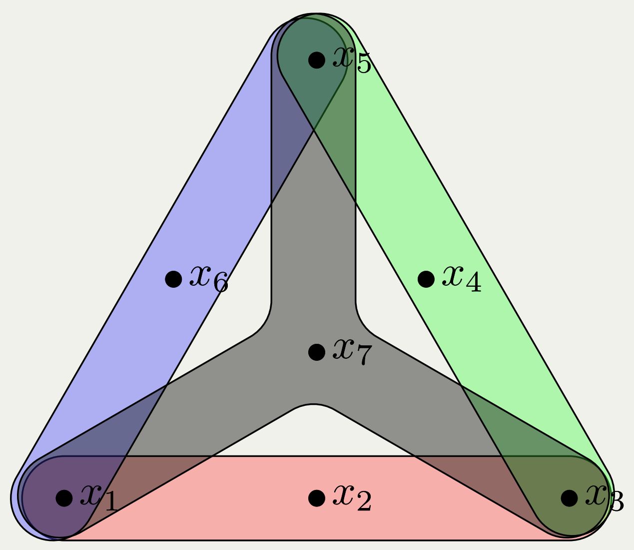
$$F = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee \neg x_5 \vee \neg x_6) \wedge (x_1 \vee x_3 \vee x_5 \vee x_7)$$



Primal graph



Incidence graph



Hypergraph

Structural Tractability

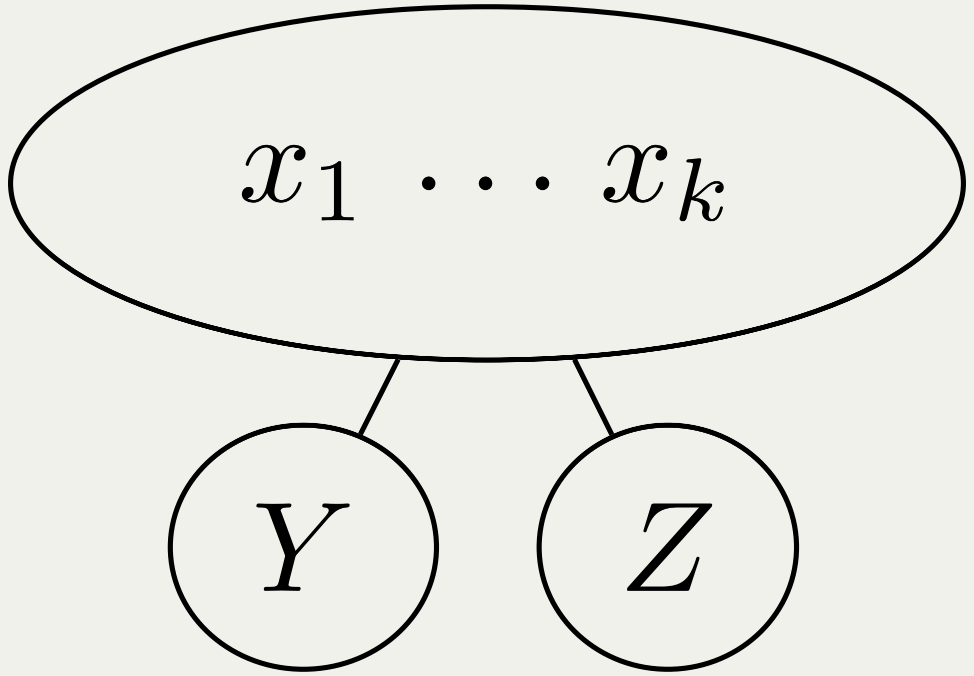
Theorem

If (primal / incidence) graph of F of size n has **treewidth** k then $\#F$ can be computed in time $2^{O(k)}n$. [1]

Structural Tractability

Theorem

If (primal / incidence) graph of F of size n has **treewidth** k then $\#F$ can be computed in time $2^{O(k)}n$. [1]



Tree decomposition of F

Exhaustive DPLL:

$$\#F[X \leftarrow \tau] = \#G[X \leftarrow \tau] \cdot \#H[X \leftarrow \tau]$$

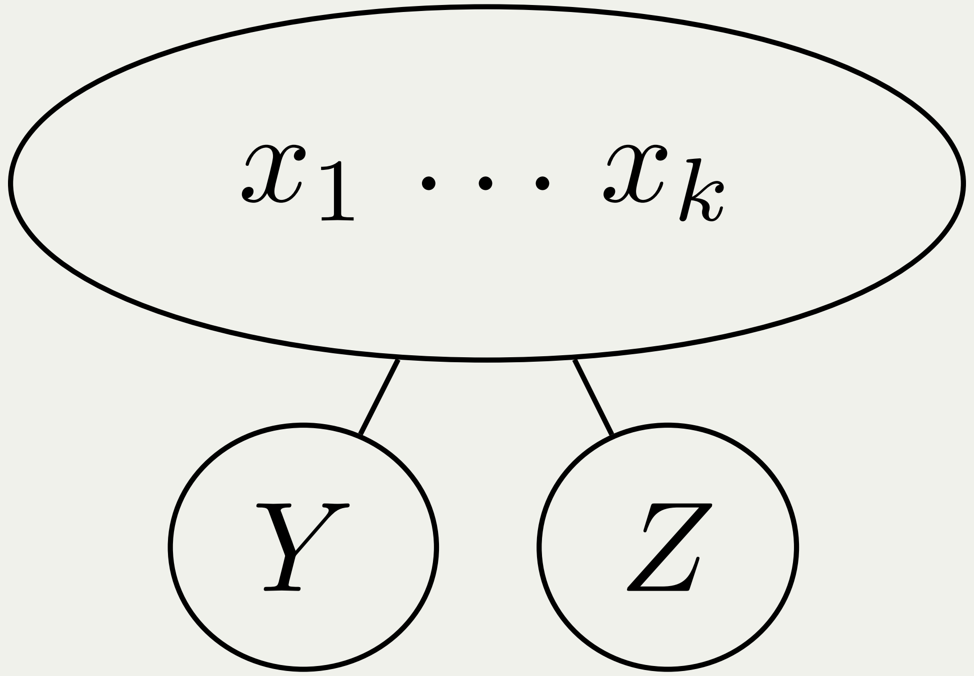
since $Y \cap Z \subseteq X$

[1] Samer, Marko, and Stefan Szeider. “Algorithms for propositional model counting.” Journal of Discrete Algorithms 8.1 (2010): 50-64.

Structural Tractability

Theorem

If (primal / incidence) graph of F of size n has **treewidth** k then $\#F$ can be computed in time $2^{O(k)}n$. [1]



Tree decomposition of F

Exhaustive DPLL:

$$\#F[X \leftarrow \tau] = \#G[X \leftarrow \tau] \cdot \#H[X \leftarrow \tau]$$

since $Y \cap Z \subseteq X$

- Branch on 2^k values $x_1, \ldots, x_k$
- Recursive calls on H and G
- Cache subformulas already solved

[1] Samer, Marko, and Stefan Szeider. "Algorithms for propositional model counting." Journal of Discrete Algorithms 8.1 (2010): 50-64.

Hypergraph Acyclicities

α-acyclicity

Generalize acyclicity to hypergraphs:

- Used in databases/CSP (tractable conjunctive queries / CSPs).
- Usually defined in terms of **tree decompositions** of hypergraphs… **Not today!**

Definition

A graph is acyclic if and only if we can obtained the empty graph by iteratively **removing leaves**.

α-acyclicity

Generalize acyclicity to hypergraphs:

- Used in databases/CSP (tractable conjunctive queries / CSPs).
- Usually defined in terms of **tree decompositions** of hypergraphs… **Not today!**

Definition

A *hyper*graph is **α-acyclic** if and only if we can obtained the empty graph by iteratively **removing α- leaves**.

α-acyclicity

Generalize acyclicity to hypergraphs:

- Used in databases/CSP (tractable conjunctive queries / CSPs).
- Usually defined in terms of **tree decompositions** of hypergraphs… **Not today!**

Definition

A *hyper*graph is **α-acyclic** if and only if we can obtained the empty graph by iteratively **removing α- leaves**.

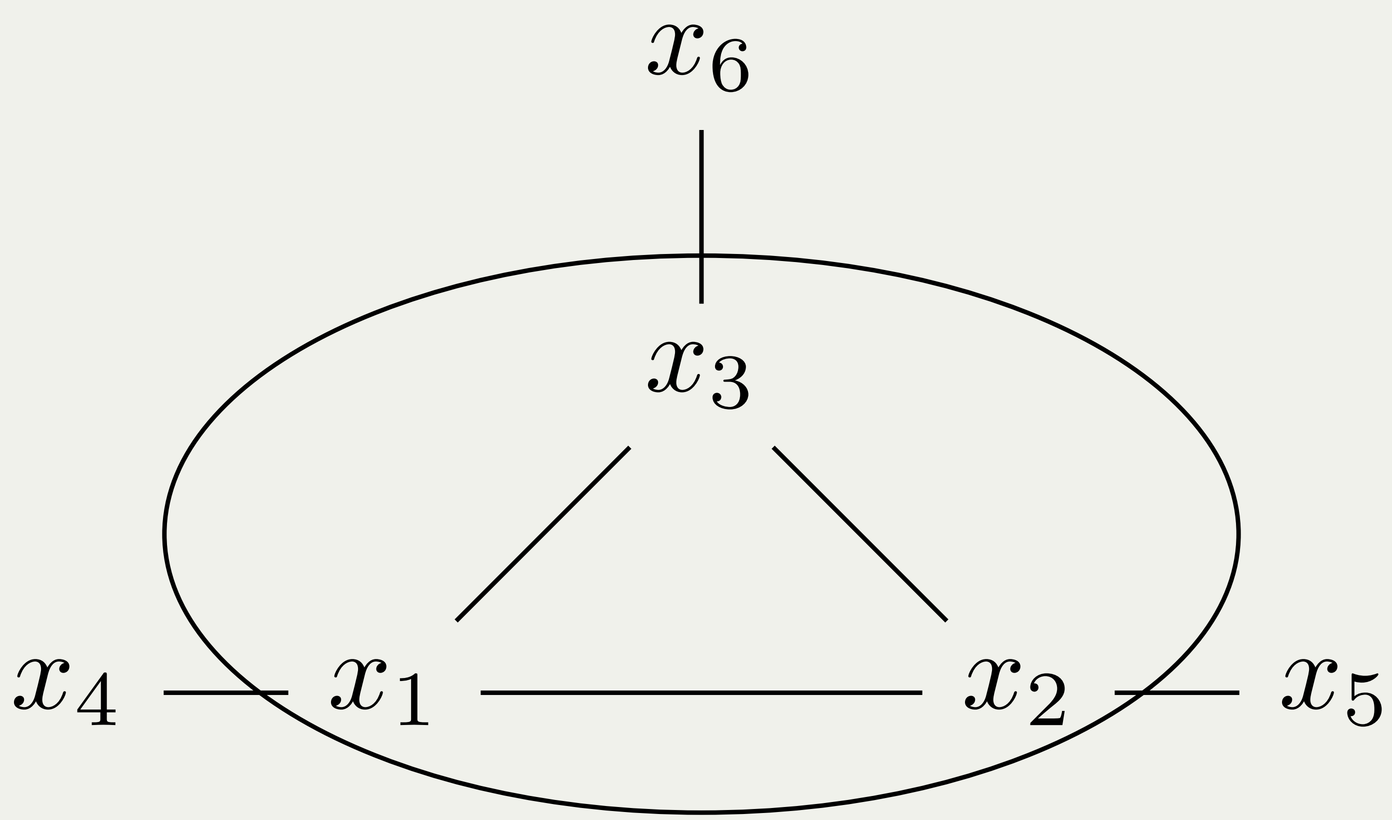
We call such vertex ordering: **α-elimination order.**

α-leaves

- $H = (V, E)$ a hypergraph.
- $N(v)$: neighborhood of v

Definition

A vertex v in a hypergraph is an α-leave if $N(v) \subseteq e$ for some edge e of H



- $x_6, \ldots, x_1$ is an α-elimination order.
- Subgraphs may no be α-acyclic (look, a triangle!)

SAT is hard on α-acyclic hypergraphs

Not a good variable-clause restriction for tractability:

- F a CNF formula
- $F' = F \wedge (x_1 \vee \ldots \vee x_n \vee y)$ is α-acyclic
- F' SAT iff F is SAT.

Hard subformulas make the formula hard (*this does not happen with conjunctive queries*).

Enters the rest of greek alphabet

Definition

A hypergraph H is β-acyclic if and only if every $H' \subseteq H$ is α-acyclic.

How can we use it algorithmically?

Enters the rest of greek alphabet

Definition

A hypergraph H is β-acyclic if and only if every $H' \subseteq H$ is α-acyclic.

How can we use it algorithmically?

Theorem

A hypergraph H is β-acyclic if and only if there exists an order on V that is an α-elimination order for **every** $H' \subseteq H$.

We call such ordering a **β-elimination order.**

Side note: this is **not** how β-elimination order is usually defined.

SAT and β-acyclicity

SAT is easy on β-acyclic instances, with a classical algorithm [2]:

Theorem

Davis-Putnam resolution following a β-elimination order terminates in polynomial time!

[2] Ordyniak, Sebastian, Daniël Paulusma, and Stefan Szeider. "Satisfiability of acyclic and almost acyclic CNF formulas." Theoretical Computer Science, 2013.

#SAT and β-acyclicity

#SAT is easy on β-acyclic instances, with classical algorithm [3]:

Theorem

Exhaustive DPLL following a reversed β-elimination order terminates in polynomial time!

#SAT and β-acyclicity

#SAT is easy on β-acyclic instances, with classical algorithm [3]:

Theorem

Exhaustive DPLL following a reversed β-elimination order terminates in polynomial time!

- Tractable case not captured by **bounded treewidth** or other existing graph measures
- Only works for a very restricted set of instances.

[3] Florent Capelli, Understanding the complexity of #SAT using knowledge compilation, LICS, 2017.

Hyperorder widths

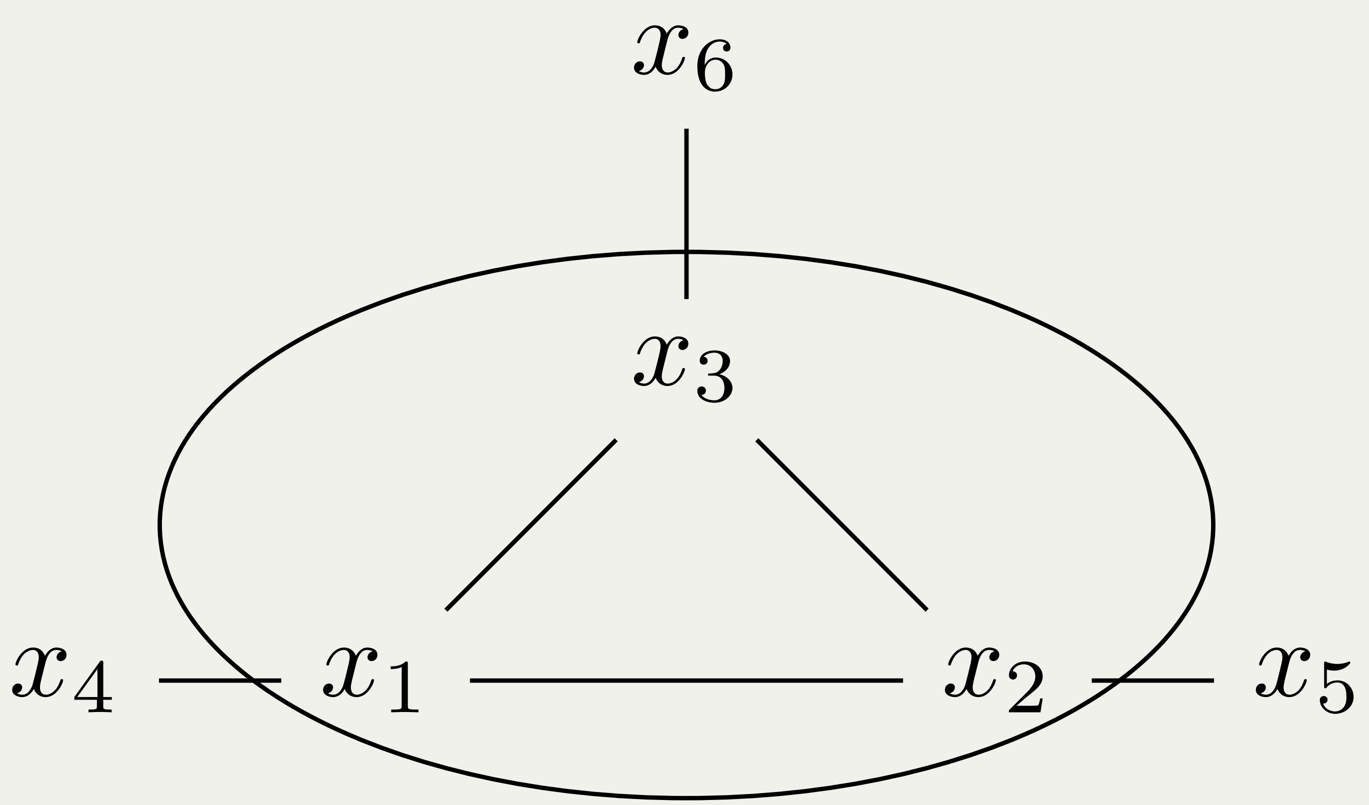
Non acyclic hypergraphs

How do we measure how far we are from acyclicity?

- α-acyclicity naturally generalizes to **hypertree width**: $htw(H) \in \mathbb{N}$.
- Usually defined via tree decomposition.
- We give an **order based definition**.

Width of an elimination order

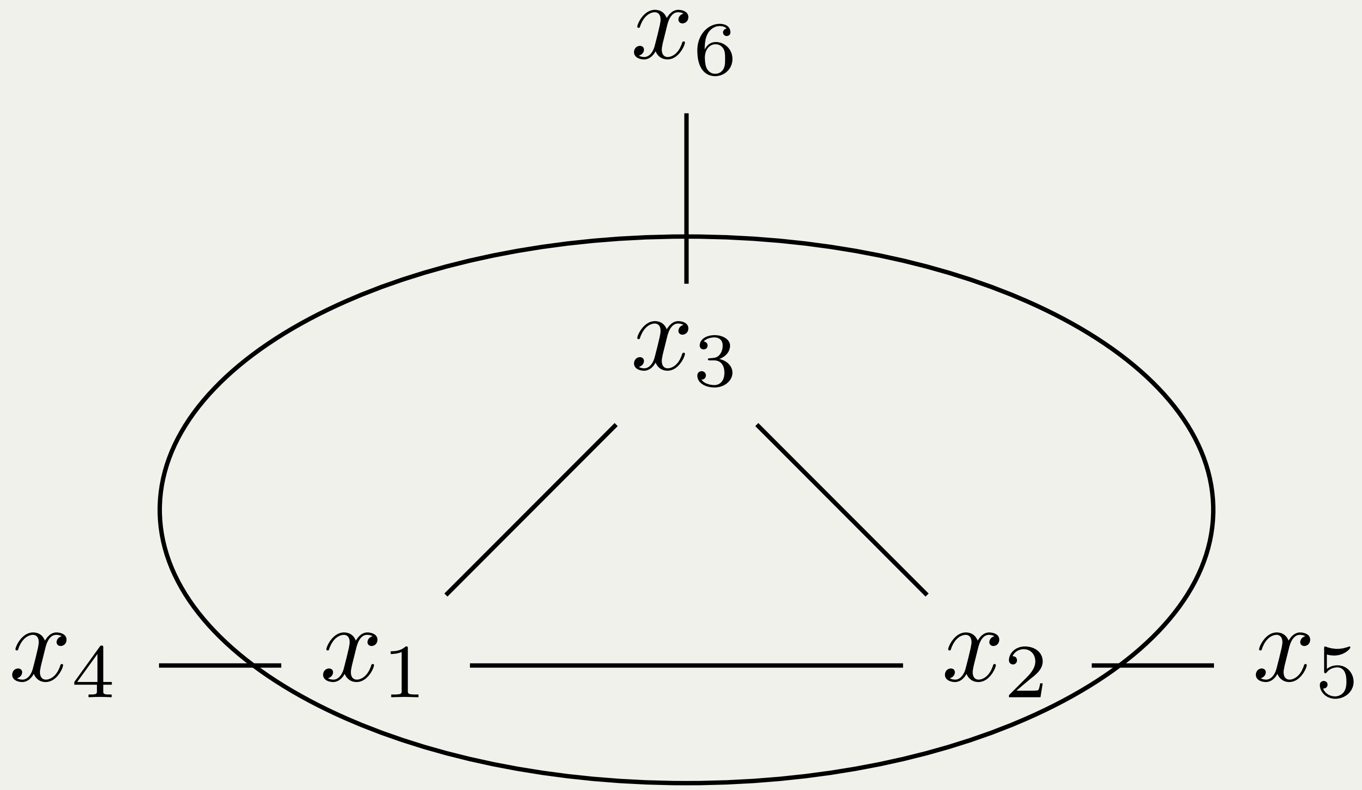
- $H = (V, E)$, $\pi = (v_1, ..., v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, ..., v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



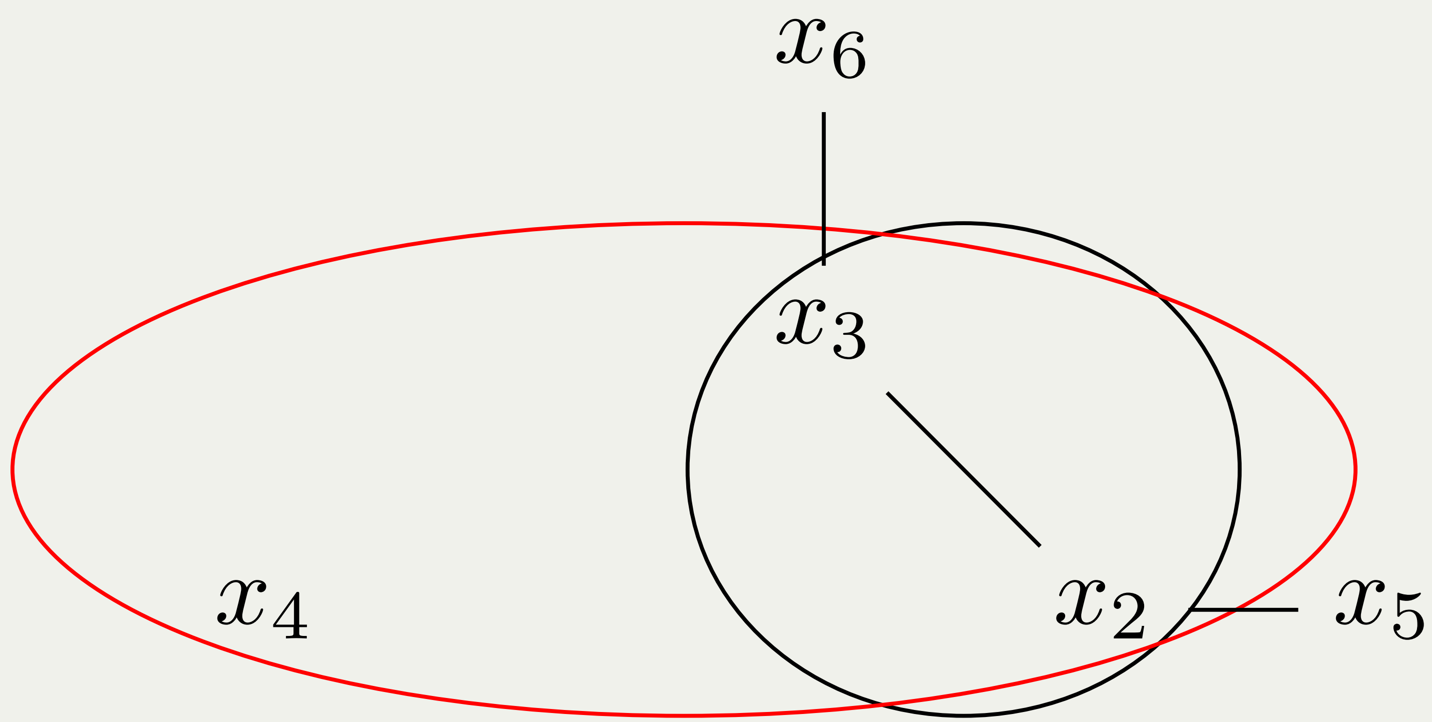
Original hypergraph

Width of an elimination order

- $H = (V, E)$, $\pi = (v_1, ..., v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, ..., v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



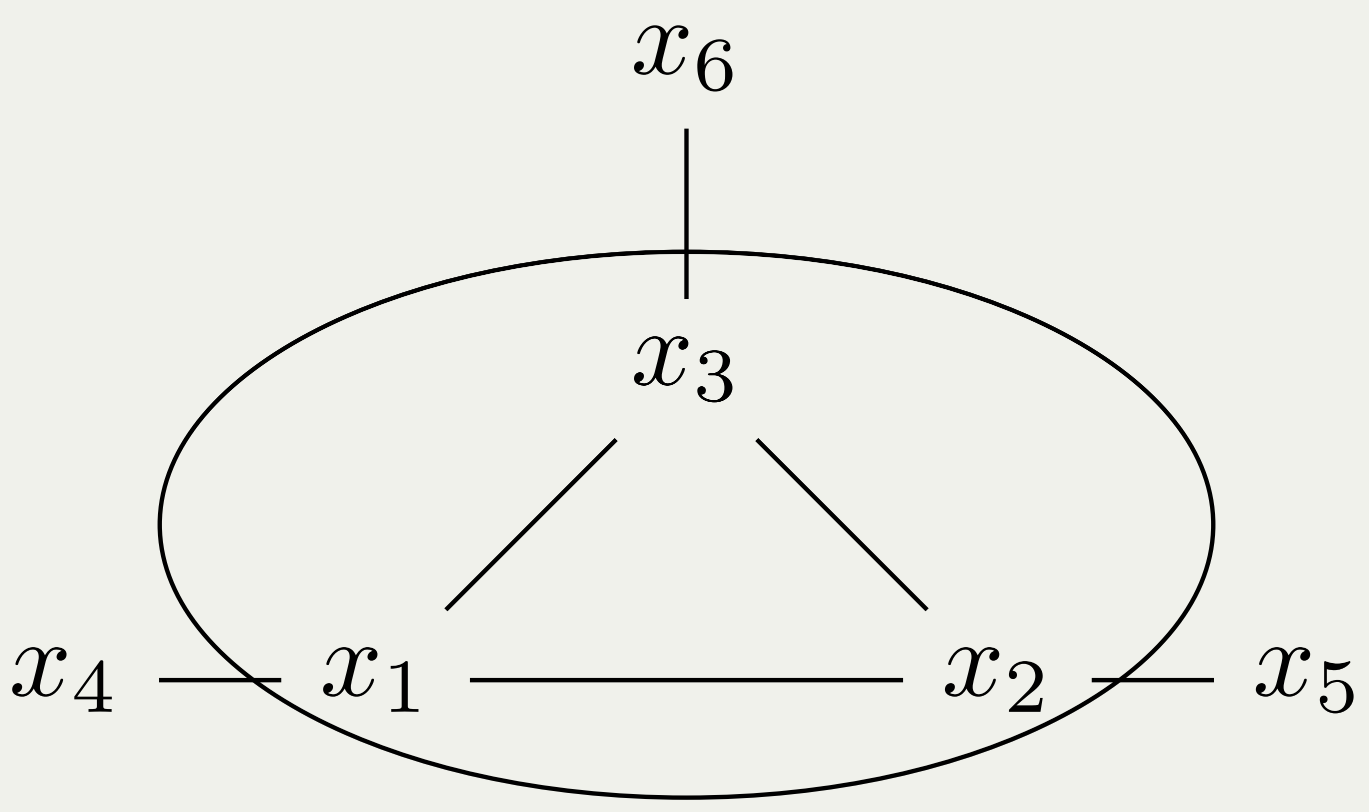
Original hypergraph



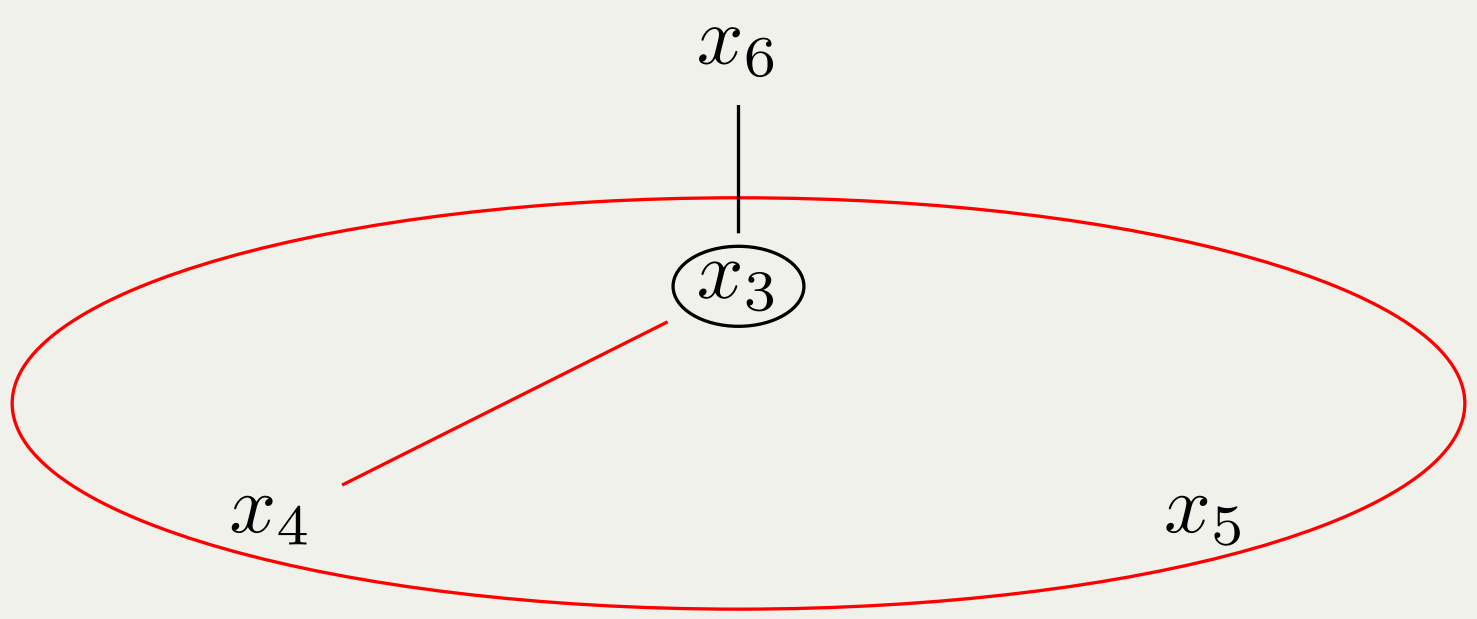
$N(x_1)$ covered by 2 edges of H

Width of an elimination order

- $H = (V, E)$, $\pi = (v_1, ..., v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, ..., v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



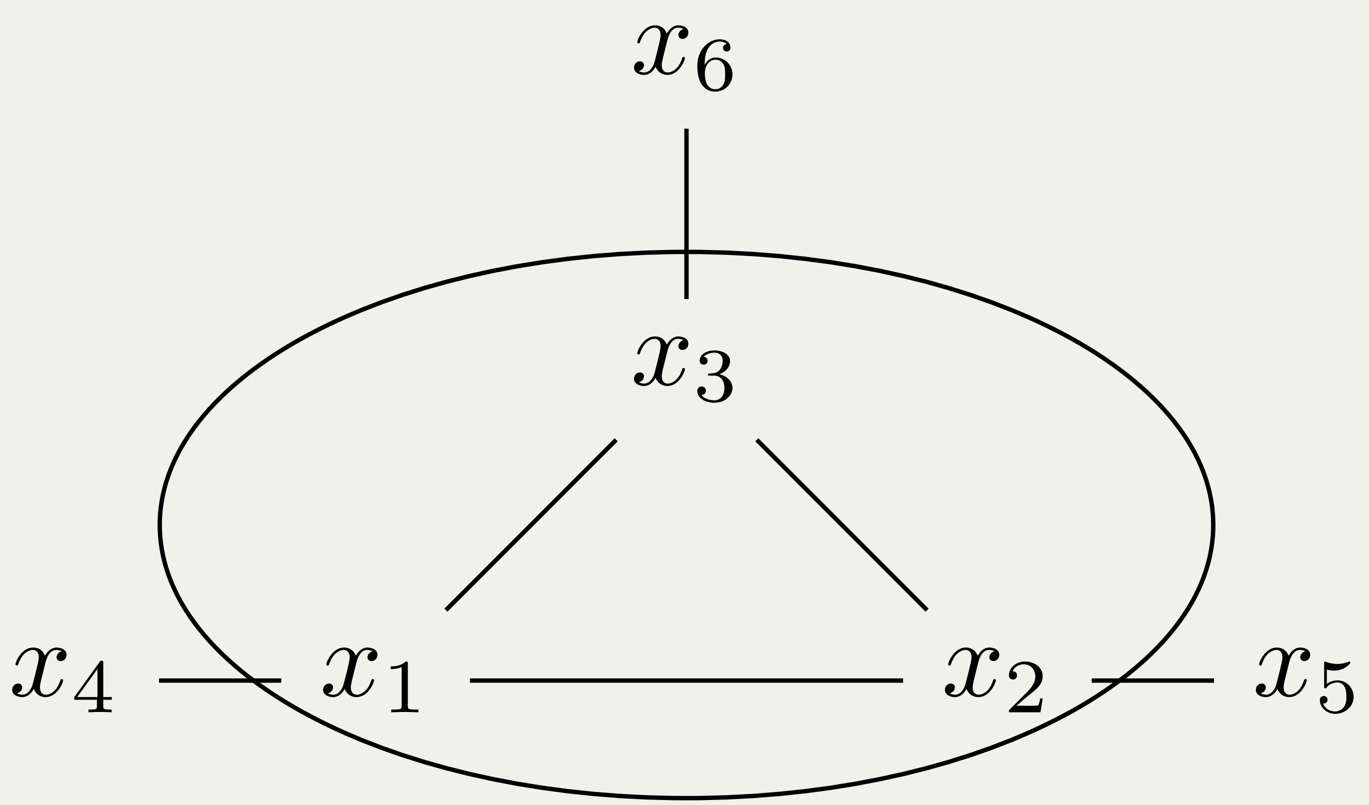
Original hypergraph



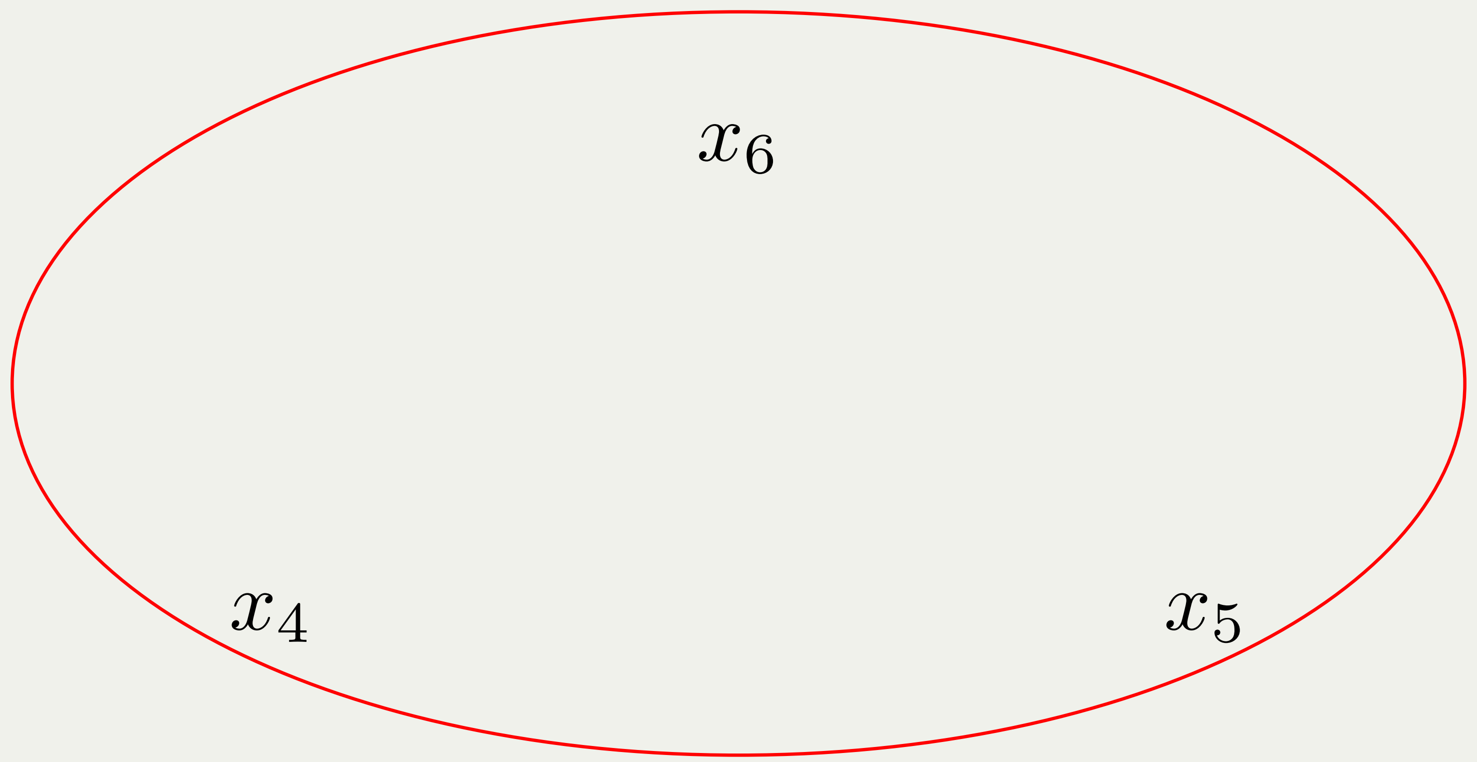
$N(x_2)$ covered by 3 edges of H

Width of an elimination order

- $H = (V, E)$, $\pi = (v_1, \ldots, v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, \ldots, v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



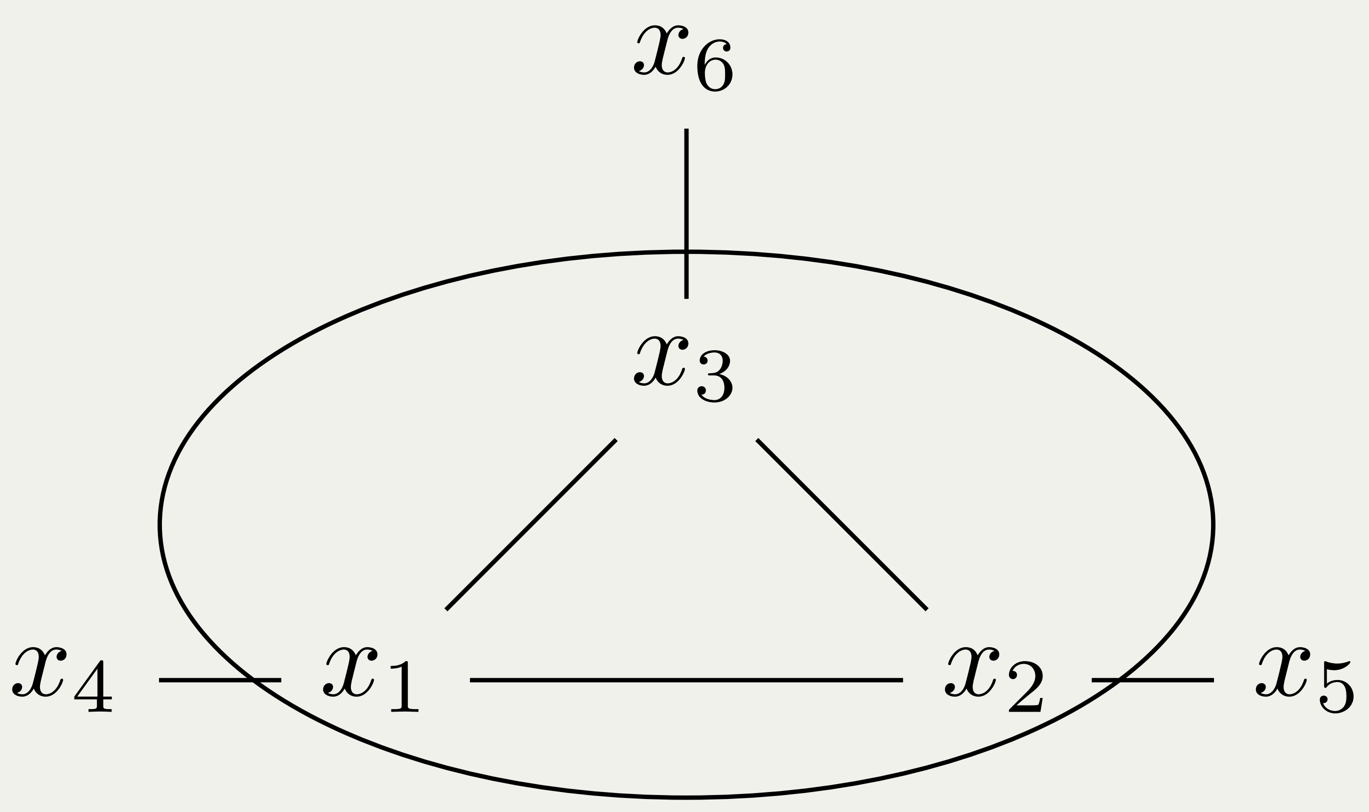
Original hypergraph



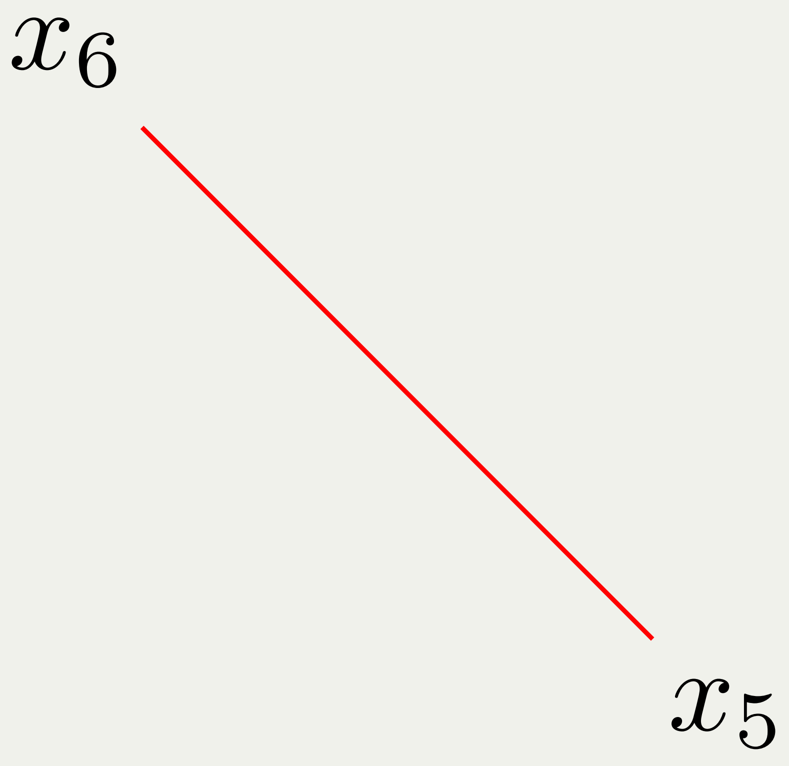
$N(x_3)$ covered by 3 edges of H

Width of an elimination order

- $H = (V, E)$, $\pi = (v_1, ..., v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, ..., v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



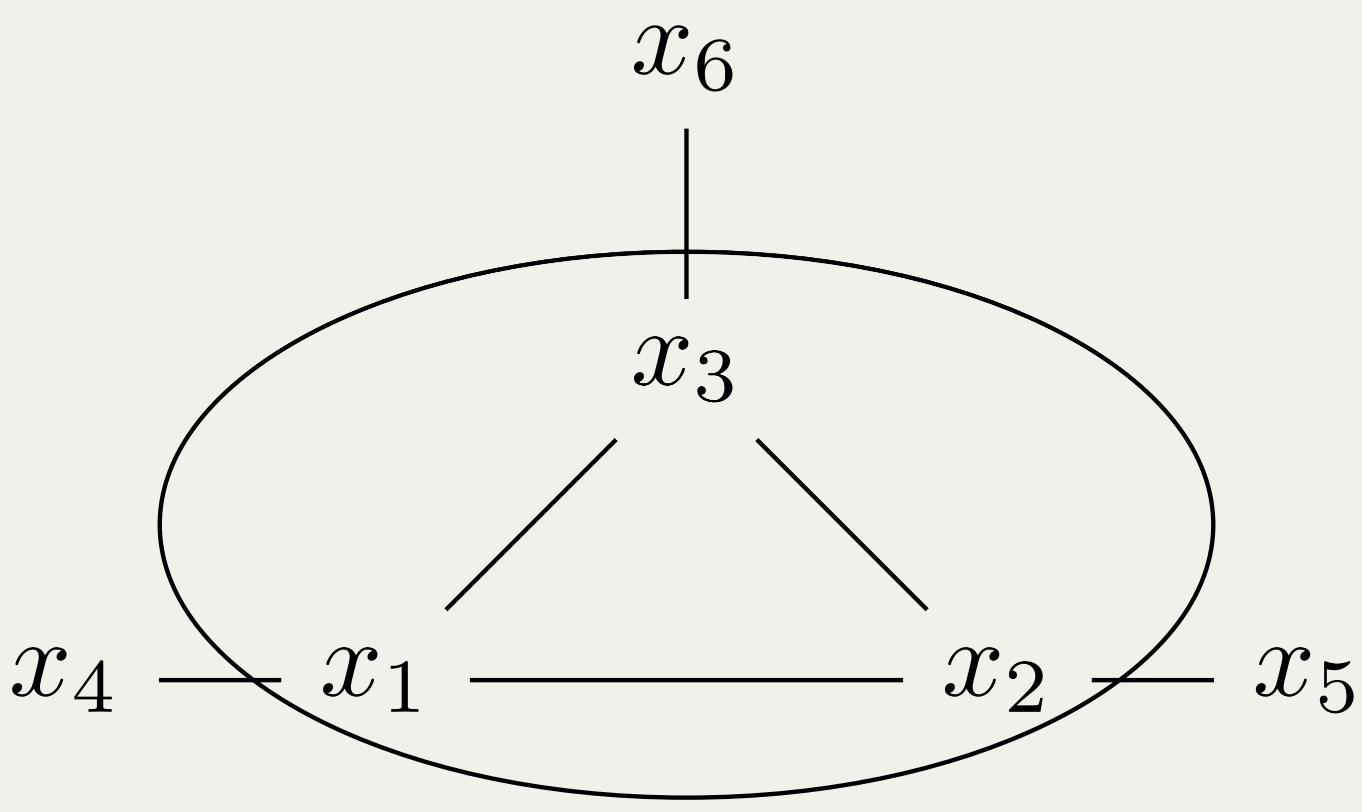
Original hypergraph



$N(x_4)$ covered by 3 edges of H

Width of an elimination order

- $H = (V, E)$, $\pi = (v_1, ..., v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, ..., v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



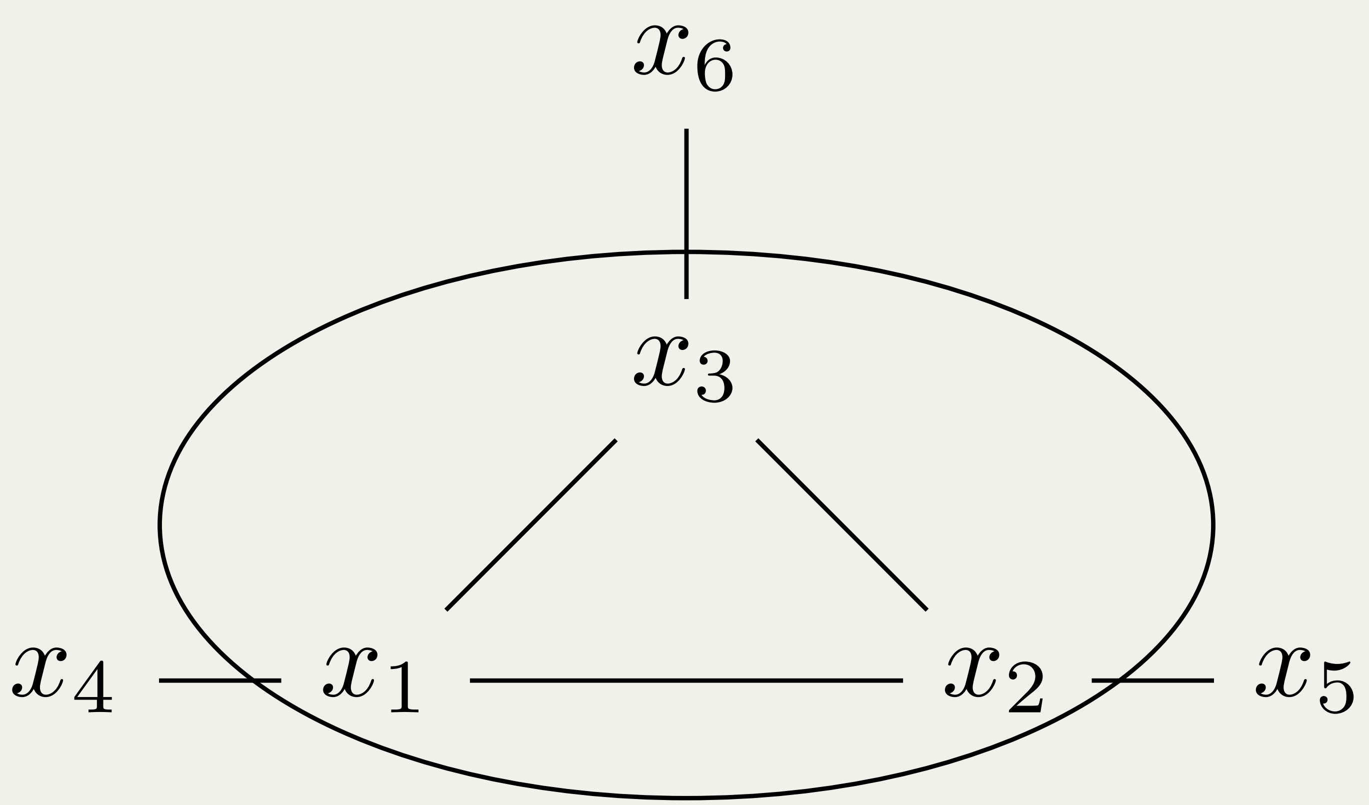
Original hypergraph

x_6

$N(x_5)$ covered by 2 edges of H

Width of an elimination order

- $H = (V, E)$, $\pi = (v_1, ..., v_n)$ order on V.
- Iteratively add edge $N(v_i)$ and remove v_i
- *Hyperorder width* $how(H, \pi)$ of $\pi = (v_1, ..., v_n)$: maximum number of edges from H to cover the neighborhood of v_i in H_i.



Original hypergraph

$$how(H, (x_1, ..., x_6)) = 3$$
$$how(H, (x_6, ..., x_1)) = 1$$

Hyperorder width and Hypertree width

Hypertree width of H : $htw(H) = \min_T htw(H, T)$ where T is **a tree decomposition**
Hyperorder width of H : $how(H) = \min_\pi how(H, \pi)$ where π is **an elimination order**.

Theorem

$$how(H) = htw(H).$$

- $how(H) = 1$ iff H is α-acyclic
- For *how(·)*, the **order is the decomposition**.

β-Hypertree Width

Sometimes, there is $H' \subseteq H$ st $htw(H') > htw(H)$.
Same trick as before:

$$\beta htw(H) = \max_{H' \subseteq H} htw(H')$$

How can we use it algorithmically?

β-Hypertree Width

Sometimes, there is $H' \subseteq H$ st $htw(H') > htw(H)$.

Same trick as before:

$$\beta htw(H) = \max_{H' \subseteq H} htw(H')$$

How can we use it algorithmically?

We do not know…

Problem with β-Hypertree Width

Expanding the definition:

$$\beta htw(H) = \max_{H' \subseteq H} \min_{T} htw(H', T)$$

Problem: a different decomposition can be used for different subhypergraphs…

Problem with β-Hypertree Width

Expanding the definition:

$$\beta htw(H) = \max_{H' \subseteq H} \min_{T} htw(H', T)$$

Problem: a different decomposition can be used for different subhypergraphs…

Swap quantifiers!

$$\beta' htw(H) = \min_{T} \max_{H' \subseteq H} htw(H', T)$$

Problem with β-Hypertree Width

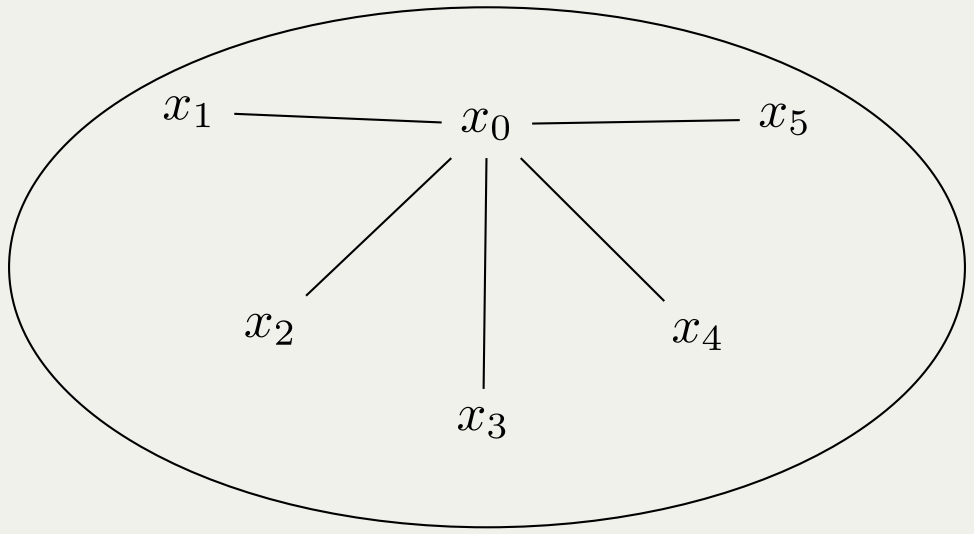
Expanding the definition:

$$\beta htw(H) = \max_{H' \subseteq H} \min_{T} htw(H', T)$$

Problem: a different decomposition can be used for different subhypergraphs…

Swap quantifiers!

$$\beta' htw(H) = \min_{T} \max_{H' \subseteq H} htw(H', T)$$



S_4

Problem: $\beta' htw(S_n) = n\ldots$

Bringing Order

For H β-acyclic:

- $H_1, H_2 \subseteq H$ may have very different tree decompositions.
- **Tree decomposition is not the right tool here**.

Theorem

A hypergraph H is β-acyclic if and only if there exists an order on V that is an α-elimination order for **every** $H' \subseteq H$.

Bringing Order

For H β-acyclic:

- $H_1, H_2 \subseteq H$ may have very different tree decompositions.
- **Tree decomposition is not the right tool here**.

Theorem

A hypergraph H is β-acyclic if and only if there exists an order on V that is an α-elimination order for **every** $H' \subseteq H$.

$$\begin{aligned} \beta htw(H) &= \max_{H' \subseteq H} \min_{T} htw(H', T) \\ &= \max_{H' \subseteq H} \min_{\pi} how(H', \pi) \end{aligned}$$

Swap quantifier in the second equality:

$$\beta how(H) = \min_{\pi} \max_{H' \subseteq H} how(H', \pi)$$

#SAT and $\beta how(H)$

Theorem

#SAT can be solved in time $n^{O(k)}$ for a formula F of size n and $\beta how(F) = k$.

#SAT and $\beta how(H)$

Theorem

#SAT can be solved in time $n^{O(k)}$ for a formula F of size n and $\beta how(F) = k$.

- **Algorithm**: exhaustive DPLL following a reversed optimal elimination order.

#SAT and $\beta how(H)$

Theorem

#SAT can be solved in time $n^{O(k)}$ for a formula F of size n and $\beta how(F) = k$.

- **Algorithm**: exhaustive DPLL following a reversed optimal elimination order.
- Generalizes tractability of β-acyclic formulas and bounded **nest set width** [4]

[4] Lanzinger, M.. Tractability beyond β-acyclicity for conjunctive queries with negation and SAT. Theoretical Computer Science, 2023.

#SAT and $\beta how(H)$

Theorem

#SAT can be solved in time $n^{O(k)}$ for a formula F of size n and $\beta how(F) = k$.

- **Algorithm**: exhaustive DPLL following a reversed optimal elimination order.
- Generalizes tractability of β-acyclic formulas and bounded **nest set width** [4]
- Algorithm implicitly constructs decision-DNNF for F:

[4] Lanzinger, M.. Tractability beyond β-acyclicity for conjunctive queries with negation and SAT. Theoretical Computer Science, 2023.

#SAT and $\beta how(H)$

Theorem

#SAT can be solved in time $n^{O(k)}$ for a formula F of size n and $\beta how(F) = k$.

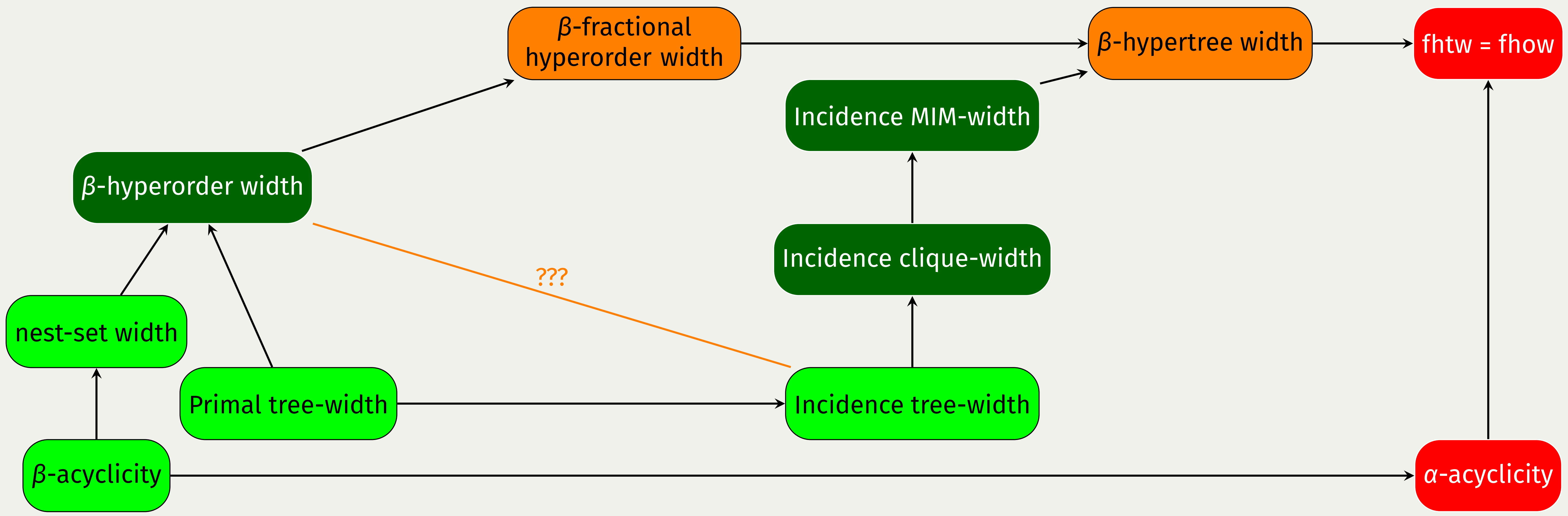
- **Algorithm**: exhaustive DPLL following a reversed optimal elimination order.
- Generalizes tractability of β-acyclic formulas and bounded **nest set width** [4]
- Algorithm implicitly constructs decision-DNNF for F:
  - gives tractable *weighted model counting*
  - tractable *direct access*
  - …

[4] Lanzinger, M.. Tractability beyond β-acyclicity for conjunctive queries with negation and SAT. Theoretical Computer Science, 2023.

Wrapping up

Structural complexity of #SAT:

FPT | XP | open | #P-hard



- Where does β-how sit in this diagram?
- Where is the frontier for SAT?

Ad

Postdoc position open at CRIL, Lens!

References

[1] Samer, Marko, and Stefan Szeider. "Algorithms for propositional model counting." Journal of Discrete Algorithms 8.1 (2010): 50-64.

[2] Ordyniak, Sebastian, Daniël Paulusma, and Stefan Szeider. "Satisfiability of acyclic and almost acyclic CNF formulas." Theoretical Computer Science, 2013.

[3] Florent Capelli, "Understanding the complexity of #SAT using knowledge compilation", LICS, 2017.

[4] Lanzinger Matthias. "Tractability beyond β-acyclicity for conjunctive queries with negation and SAT". Theoretical Computer Science, 2023.