Hypergraphs FTW

Galen Ballew, gballew@uw.edu

CSEP590D, Spring 2022

Agenda 🎯

- What are they; how are they related; why they matter:
    - Hypergraphs
    - Worst-case optimal join algorithms
    - Hypertree decompositions

Disclaimer → 10 hours in 10 minutes

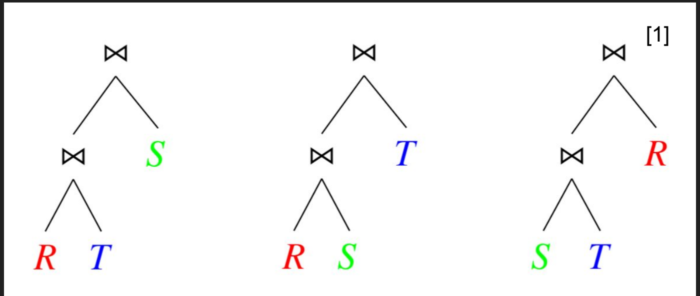
- If you know about this already:
  - Forgive me for glossing over so much
- If this is all new:
  - Forgive me for glossing over so much

Much ado about the triangle 🔺

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

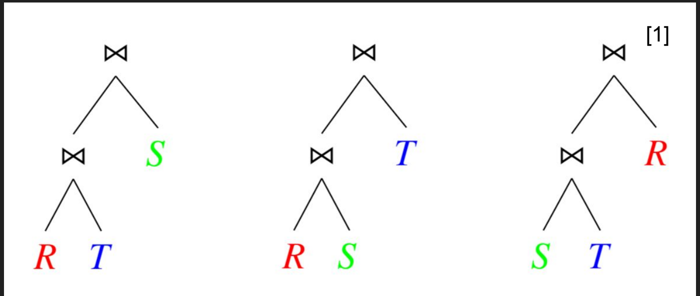
Much ado about the triangle 🔺

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$



Much ado about the triangle 🔺

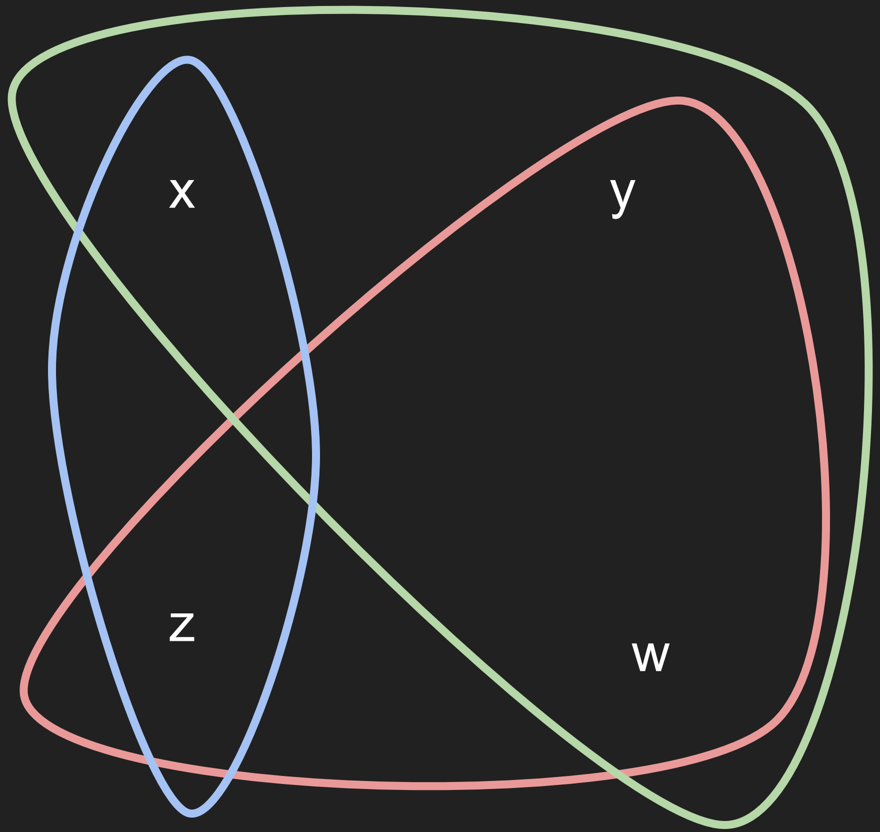
$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$



$$|Q_{\triangle}| \leq N^2$$

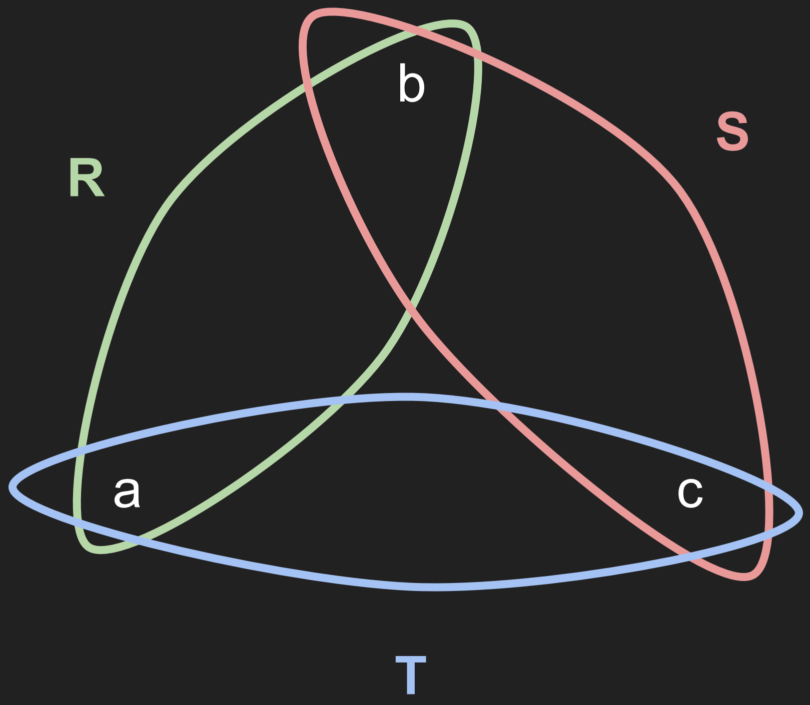
Hypergraphs 📊

- Vertices == Attributes
- Edges == Relations
- Great data representation for
  - Queries
  - Constraint Satisfaction Problems
  - Graphical Models
  - Nash Equilibria
  - And more ?!



Hypergraphs 📊

- Edge Cover == subset of edges that contain all vertices



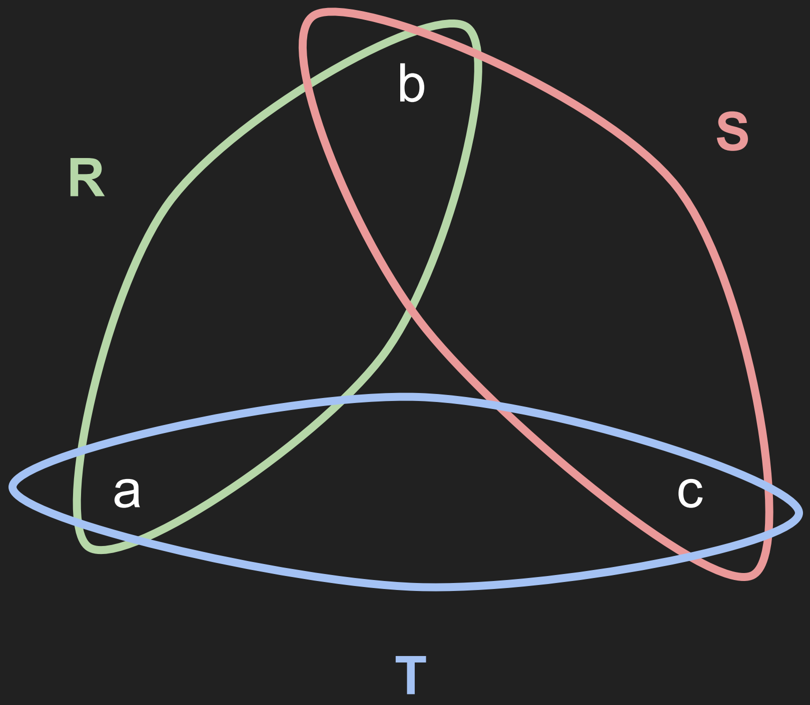
Hypergraphs

- Edge Cover == subset of edges that contain all vertices

- Fractional Edge Cover ==

  Numbers, one for each edge e, s.t for every vertex v

$$\sum_{e:v\in e} u_e \geq 1$$



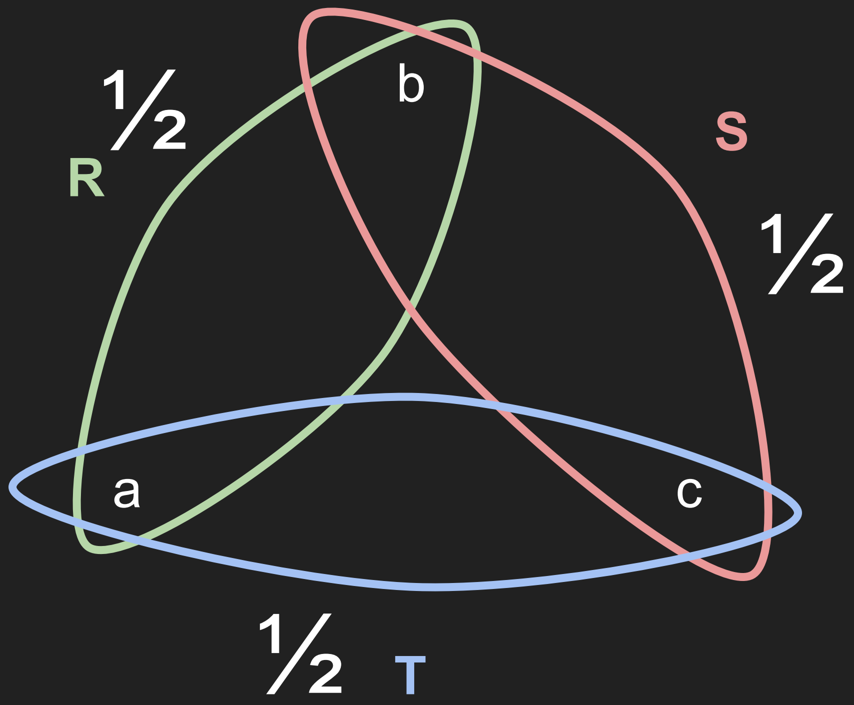
Hypergraphs 📊

- Edge Cover == subset of edges that contain all vertices

- Fractional Edge Cover ==

  Numbers, one for each edge e, s.t for every vertex v

$$\sum_{e:v\in e} u_e \geq 1$$



AGM Bound 📐

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

AGM Bound 📐

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

|R|, |S|, |T| ≤ N

$$|Q_{\triangle}| \leq N^2$$

AGM Bound 📐

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

|R|, |S|, |T| ≤ N

$$|Q_{\triangle}| \leq N^2$$

With AGM Bound

$$|Q_{\triangle}| \leq |R_1|^{u_1} \times \ldots \times |R_m|^{u_m}$$

AGM Bound 📐

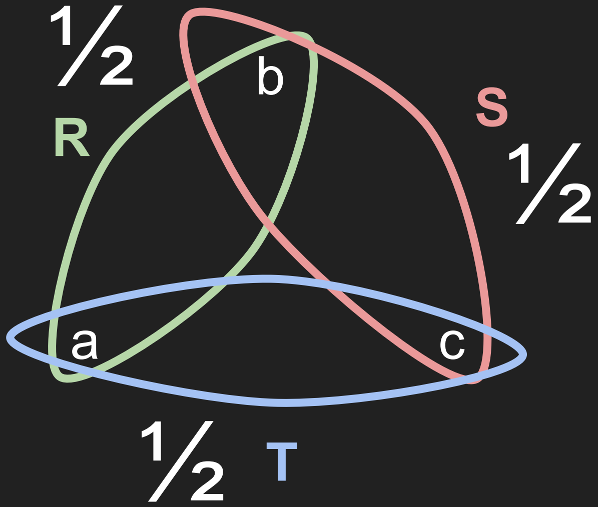
$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

|R|, |S|, |T| ≤ N

$$|Q_{\triangle}| \leq N^2$$

With AGM Bound

$$|Q_{\triangle}| \leq |R_1|^{u_1} \times \ldots \times |R_m|^{u_m}$$



AGM Bound 📐

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

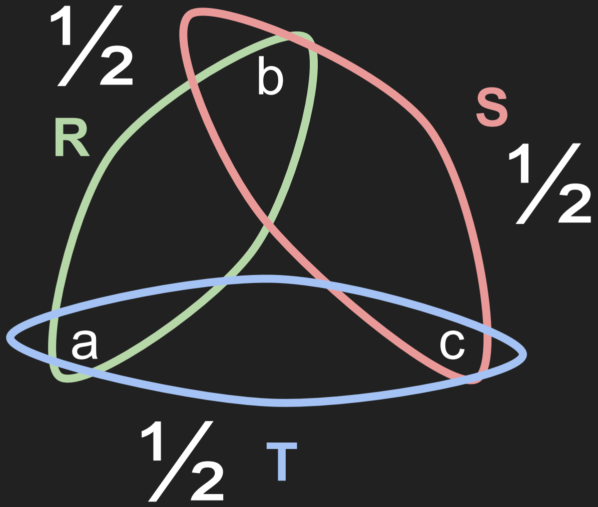
|R|, |S|, |T| ≤ N

$$|Q_{\triangle}| \leq N^2$$

With AGM Bound

$$|Q_{\triangle}| \leq |R_1|^{u_1} \times \ldots \times |R_m|^{u_m}$$

$$|Q_{\triangle}| \leq |R|^{1/2} \times |S|^{1/2} \times |T|^{1/2}$$



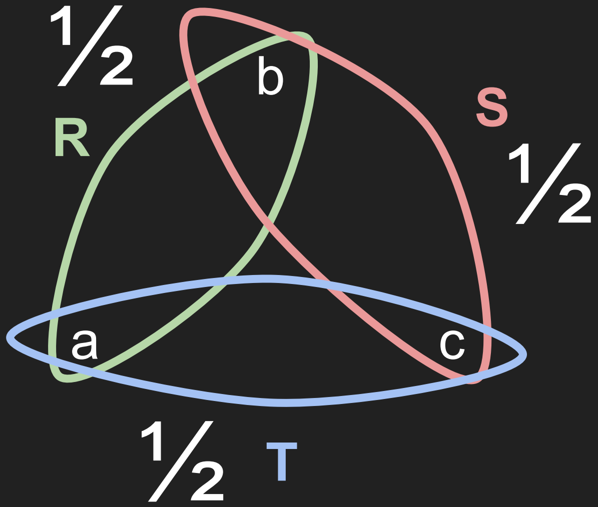
AGM Bound 📐

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

|R|, |S|, |T| ≤ N

$$|Q_{\triangle}| \leq N^2$$

With AGM Bound



$$|Q_{\triangle}| \leq |R_1|^{u_1} \times \ldots \times |R_m|^{u_m}$$

$$|Q_{\triangle}| \leq |R|^{1/2} \times |S|^{1/2} \times |T|^{1/2}$$

$$|Q_{\triangle}| \leq N^{3/2}$$

So what? 🤷

So what? 🤷

- AGM Bound is 100% math magic.

So what? 🤷

- AGM Bound is 100% math magic.
- Can we design algorithms with a matching runtime?

So what? 🤷

- AGM Bound is 100% math magic.

- Can we design algorithms with a matching runtime?

Worst-case Optimal Join Algorithms 🆒

Worst-case Optimal Join Algorithms 🆒

[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

// Input: Hypergraph $H = (V, E)$*, and a tuple* t*.*
Generic−Join(V, E, t):
  if $|V| = 1$ then **return** $\cap_{e \in E} R_e[t]$.
  Let $I = \{v_1\}$ *// the first attribute.*
  Q $\leftarrow \emptyset$ *// the return value*
  // Intersect all relations that contain v_1
  // Only those tuples that agree with t*.*
  for every $t_v \in \cap_{e \in E : e \ni v_1} \pi_I(R_e[t])$ **do**
    $Q_t \leftarrow$ Generic−Join($V - I$, E, $t :: t_v$)
    Q $\leftarrow Q \cup \{t_v\} \times Q_t$
  return Q

Worst-case Optimal Join Algorithms 🆒

- Runs on hypergraph
  - Conceptually, performs a recursive backtracking search
  - Focused on intersections

[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

// *Input: Hypergraph* $H = (V, E)$*, and a tuple* t*.*
Generic−Join(V, E, t):
  if $|V| = 1$ then **return** $\cap_{e \in E} R_e[t]$.
  Let $I = \{v_1\}$ // *the first attribute.*
  Q $\leftarrow \emptyset$ // *the return value*
  // *Intersect all relations that contain* v_1
  // *Only those tuples that agree with* t*.*
  for every $t_v \in \cap_{e \in E : e \ni v_1} \pi_I(R_e[t])$ **do**
    $Q_t \leftarrow$ Generic−Join($V - I$, E, $t :: t_v$)
    Q $\leftarrow Q \cup \{t_v\} \times Q_t$
  return Q

Worst-case Optimal Join Algorithms 🆒

- Runs on hypergraph
    - Conceptually, performs a recursive backtracking search
    - Focused on intersections
- Runs in $\mathcal{O}(AGM(Q_\triangle)\log(N))$

[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

// *Input: Hypergraph* $H = (V, E)$, *and a tuple* t.
Generic−Join(V, E, t):
  if $|V| = 1$ then **return** $\cap_{e \in E} R_e[t]$.
  Let $I = \{v_1\}$ // *the first attribute.*
  Q $\leftarrow \emptyset$ // *the return value*
  // *Intersect all relations that contain* v_1
  // *Only those tuples that agree with* t.
  for every $t_v \in \cap_{e \in E : e \ni v_1} \pi_I(R_e[t])$ **do**
    $Q_t \leftarrow$ Generic−Join($V - I$, E, $t :: t_v$)
    Q $\leftarrow Q \cup \{t_v\} \times Q_t$
  return Q

Worst-case Optimal Join Algorithms 🆒

- Runs on hypergraph
  - Conceptually, performs a recursive backtracking search
  - Focused on intersections
- Runs in $\mathcal{O}(AGM(Q_{\triangle})\log{(N)})$
- Multi-way join
  - Variable ordering doesn't matter (as much)
  - Will always be worst-case optimal

[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

```
//Input: Hypergraph H = (V,E), and a tuple t.
Generic-Join(V,E,t):
  if |V| = 1 then return ∩_{e∈E} R_e[t].
  Let I = {v_1} // the first attribute.
  Q ← ∅ // the return value
  // Intersect all relations that contain v_1
  // Only those tuples that agree with t.
  for every t_v ∈ ∩_{e∈E:e∋v_1} π_I(R_e[t]) do
    Q_t ← Generic-Join(V - I, E, t :: t_v)
    Q ← Q ∪ {t_v} × Q_t
  return Q
```

Worst-case Optimal Join Algorithms

- Runs on hypergraph
  - Conceptually, performs a recursive backtracking search
  - Focused on intersections
- Runs in $\mathcal{O}(AGM(Q_{\triangle})\log(N))$
- Multi-way join
  - Variable ordering doesn't matter (as much)
  - Will always be worst-case optimal
- Commonly slower than binary joins if the output cardinality is less than either input cardinality [5][6]

[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

// Input: Hypergraph $H = (V, E)$*, and a tuple t.*
Generic−Join(V, E, t):
  if $|V| = 1$ then **return** $\cap_{e \in E} R_e[t]$.
  Let $I = \{v_1\}$ *// the first attribute.*
  $Q \leftarrow \emptyset$ *// the return value*
  // Intersect all relations that contain v_1
  // Only those tuples that agree with t.
  for every $t_v \in \cap_{e \in E: e \ni v_1} \pi_I(R_e[t])$ **do**
    $Q_t \leftarrow$ Generic−Join$(V - I, E, t :: t_v)$
    $Q \leftarrow Q \cup \{t_v\} \times Q_t$
  return Q

Worst-case Optimal Join Algorithms 🆒

- Runs on hypergraph
  - Conceptually, performs a recursive backtracking search
  - Focused on intersections
- Runs in $\mathcal{O}(AGM(Q_\triangle)\log{(N)})$
- Multi-way join
  - Variable ordering doesn't matter (as much)
  - Will always be worst-case optimal
- Commonly slower than binary joins if the output cardinality is less than either input cardinality [5][6]
- Amazing for cyclic queries
  - À la $Q_\triangle$

[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

```
//Input: Hypergraph H = (V,E), and a tuple t.
Generic-Join(V,E,t):
  if |V| = 1 then return ∩_{e∈E} R_e[t].
  Let I = {v_1} // the first attribute.
  Q ← ∅ // the return value
  // Intersect all relations that contain v_1
  // Only those tuples that agree with t.
  for every t_v ∈ ∩_{e∈E:e∋v_1} π_I(R_e[t]) do
    Q_t ← Generic-Join(V − I, E, t :: t_v)
    Q ← Q ∪ {t_v} × Q_t
  return Q
```

Worst-case Optimal Join Algorithms 🆒

- Runs on hypergraph
  - Conceptually, performs a recursive backtracking search
  - Focused on intersections
- Runs in $\mathcal{O}(AGM(Q_\triangle)\log(N))$
- Multi-way join
  - Variable ordering doesn't matter (as much)
  - Will always be worst-case optimal
- Commonly slower than binary joins if the output cardinality is less than either input cardinality [5][6]
- Amazing for cyclic queries
  - À la $Q_\triangle$
- Assumes all relations are pre-sorted or indexed

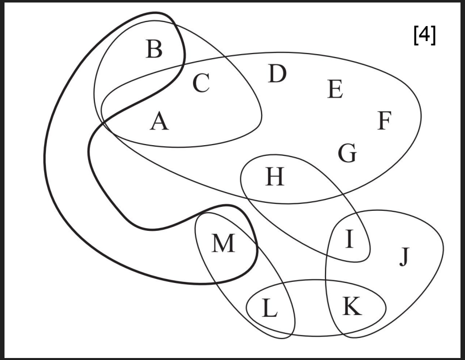
[3]

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

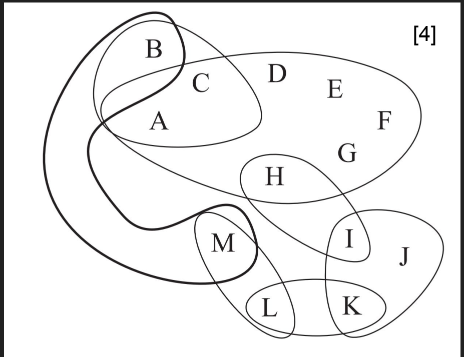
// *Input: Hypergraph* $H = (V, E)$*, and a tuple* t*.*
Generic−Join(V, E, t):
  if $|V| = 1$ then **return** $\cap_{e \in E} R_e[t]$.
  Let $I = \{v_1\}$ // *the first attribute.*
  $Q \leftarrow \emptyset$ // *the return value*
  // *Intersect all relations that contain* v_1
  // *Only those tuples that agree with* t*.*
  for every $t_v \in \cap_{e \in E : e \ni v_1} \pi_I(R_e[t])$ **do**
    $Q_t \leftarrow$ Generic−Join$(V - I, E, t :: t_v)$
    $Q \leftarrow Q \cup \{t_v\} \times Q_t$
  return Q

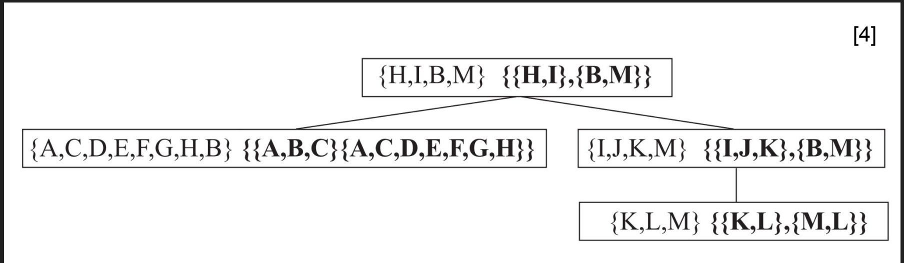
⚠️ (Generalized) Hypertree Decompositions 🌲

(Generalized) Hypertree Decompositions 🌲



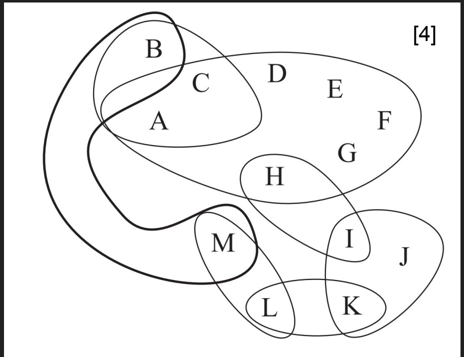
(Generalized) Hypertree Decompositions 🌲

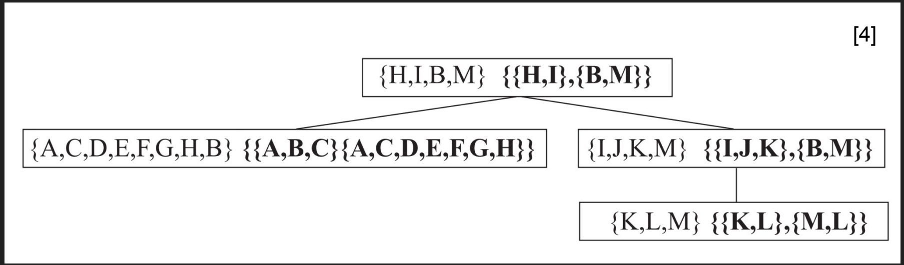




(Generalized) Hypertree Decompositions 🌲

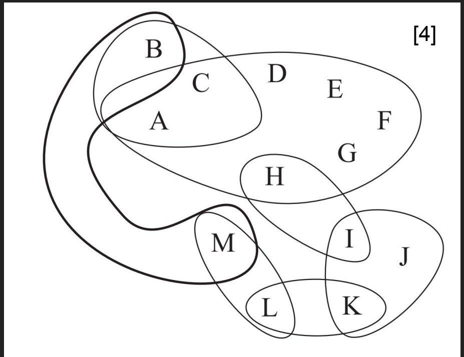
- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.



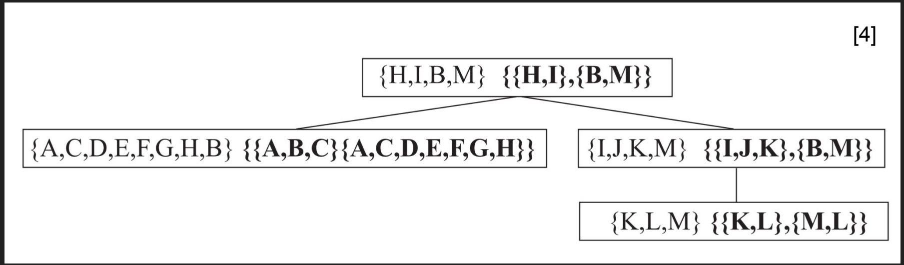


(Generalized) Hypertree Decompositions 🌲

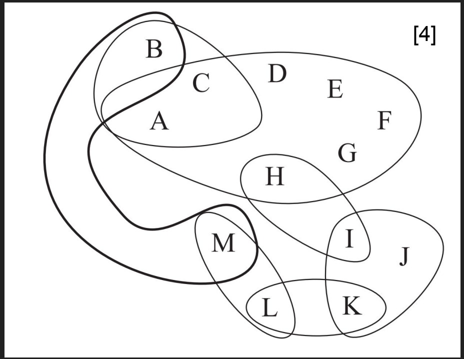
- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.



$$HD = \langle T, \chi, \lambda \rangle$$

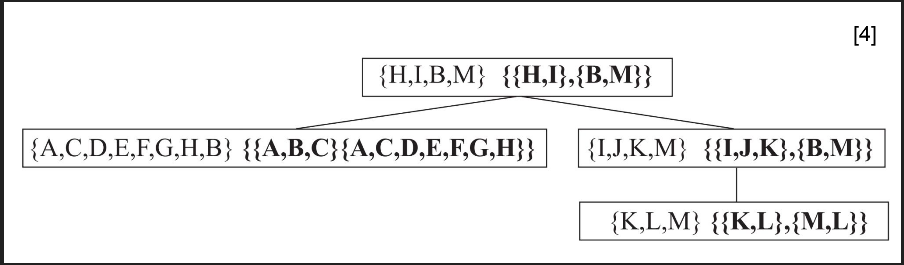


(Generalized) Hypertree Decompositions 🌲



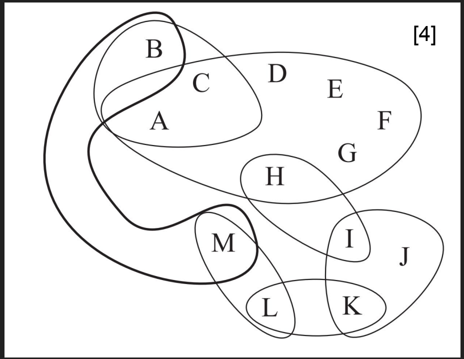
- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.
- There are potentially many GHD for a given hypergraph—some are better than others because they have a lower *width*

$$HD = \langle T, \chi, \lambda \rangle$$

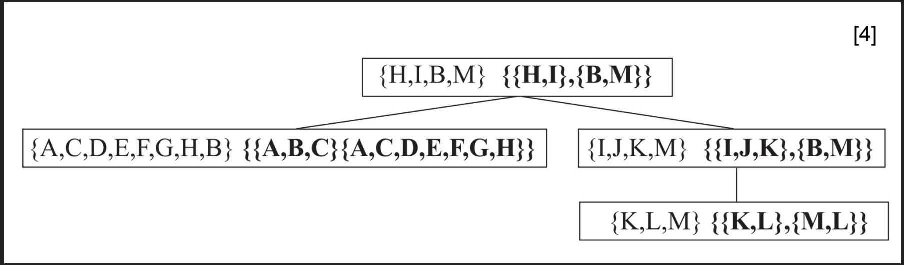


(Generalized) Hypertree Decompositions 🌲

- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.
- There are potentially many GHD for a given hypergraph—some are better than others because they have a lower *width*
- Width of GHD: $\max_{p \in V}(|\lambda(p)|)$



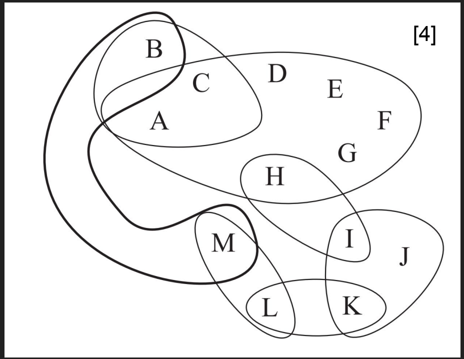
$$HD = \langle T, \chi, \lambda \rangle$$

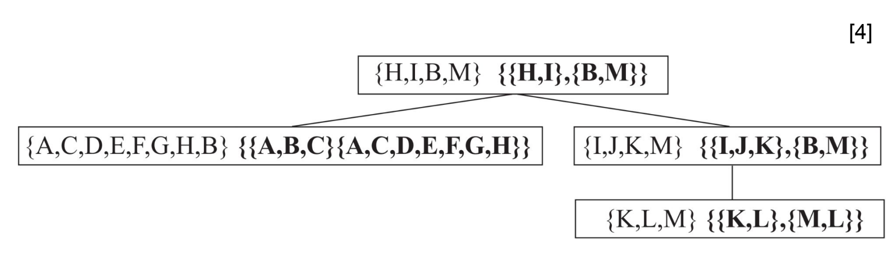


(Generalized) Hypertree Decompositions 🌲

- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.
- There are potentially many GHD for a given hypergraph—some are better than others because they have a lower *width*
- Width of GHD: $\max_{p \in V}(|\lambda(p)|)$
- Generalized hypertree width of a hypergraph: $ghw(\mathcal{H})$

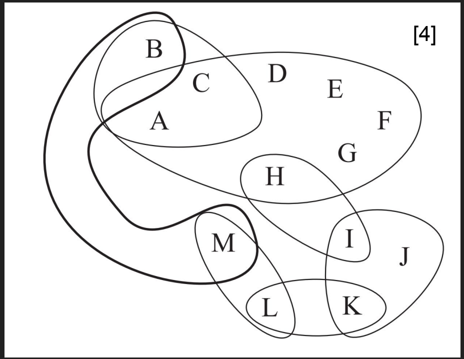
$$HD = \langle T, \chi, \lambda \rangle$$



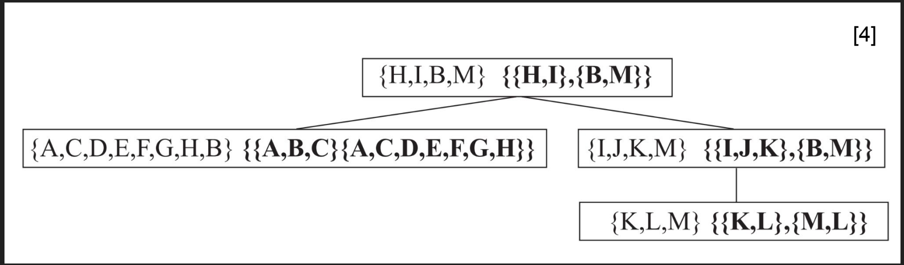


(Generalized) Hypertree Decompositions 🌲

- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.
- There are potentially many GHD for a given hypergraph—some are better than others because they have a lower *width*
- Width of GHD: $\max_{p \in V}(|\lambda(p)|)$
- Generalized hypertree width of a hypergraph: $ghw(\mathcal{H})$
  - The minimum width over all its generalized hypertree decompositions

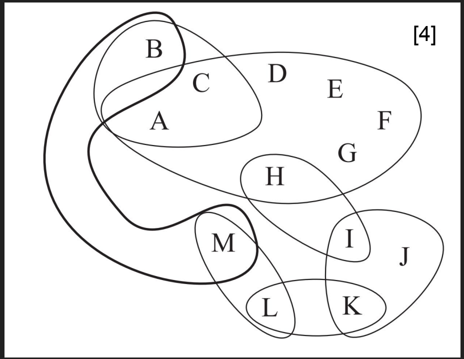


$$HD = \langle T, \chi, \lambda \rangle$$

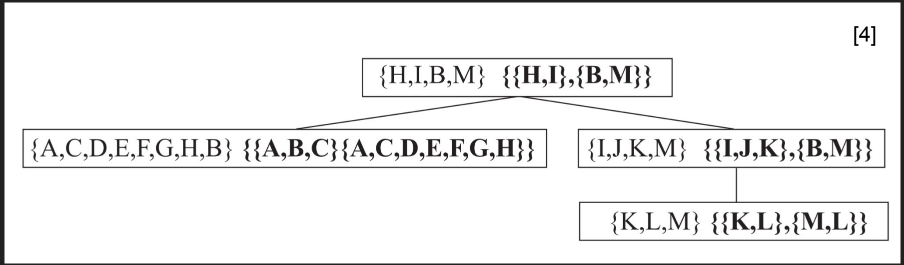


(Generalized) Hypertree Decompositions 🌲

- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.
- There are potentially many GHD for a given hypergraph—some are better than others because they have a lower *width*
- Width of GHD: $\max_{p \in V}(|\lambda(p)|)$
- Generalized hypertree width of a hypergraph: $ghw(\mathcal{H})$
  - The minimum width over all its generalized hypertree decompositions
  - Want $ghw(\mathcal{H})$ to be *bounded*

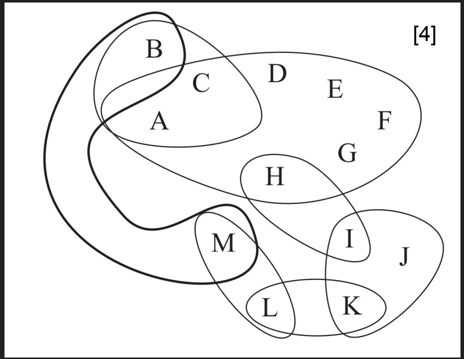


$$HD = \langle T, \chi, \lambda \rangle$$

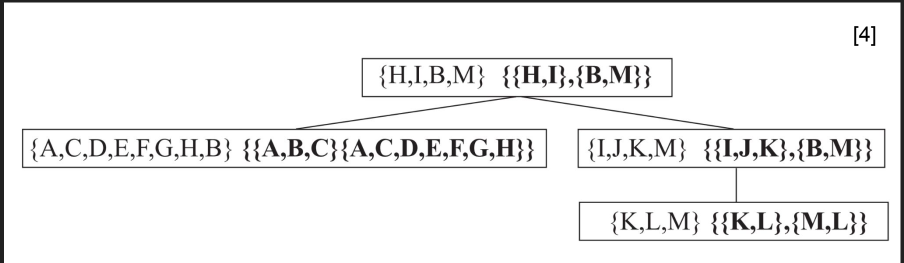


(Generalized) Hypertree Decompositions 🌲

- Layman's definition of GHD:
  - Tree decomposition of hypergraph where each node contains hypergraph vertices and the edges that cover those vertices.
- There are potentially many GHD for a given hypergraph—some are better than others because they have a lower *width*
- Width of GHD: $\max_{p \in V}(|\lambda(p)|)$
- Generalized hypertree width of a hypergraph: $ghw(\mathcal{H})$
  - The minimum width over all its generalized hypertree decompositions
  - Want $ghw(\mathcal{H})$ to be *bounded*
  - Very good measure of cyclicity



$$HD = \langle T, \chi, \lambda \rangle$$



(Generalized) Hypertree Decompositions 🌲

- We can check if a conjunctive query is of bounded hypertree width in polynomial time.

(Generalized) Hypertree Decompositions 🌲

- We can check if a conjunctive query is of bounded hypertree width in polynomial time.

- If it is, we can compute a hypertree decomposition in polynomial time
  - The best upper bound currently known for this is $\mathcal{O}\left(m^{2k}v^2\right)$ time, where *m* and *v* are the number of atoms and the number of variables in Q, respectively [4].

(Generalized) Hypertree Decompositions 🌲

- We can check if a conjunctive query is of bounded hypertree width in polynomial time.
- If it is, we can compute a hypertree decomposition in polynomial time
  - The best upper bound currently known for this is $\mathcal{O}\left(m^{2k}v^2\right)$ time, where *m* and *v* are the number of atoms and the number of variables in Q, respectively [4].
- k bound of 2 or 3 usually suffices for queries

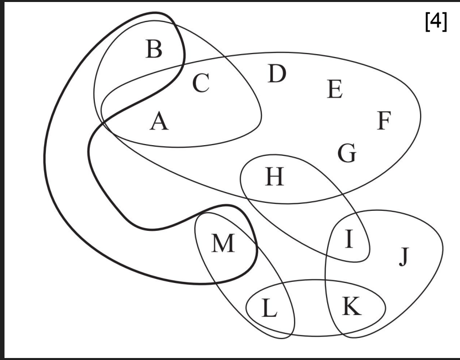
(Generalized) Hypertree Decompositions 🌲

- We can check if a conjunctive query is of bounded hypertree width in polynomial time.
- If it is, we can compute a hypertree decomposition in polynomial time
  - The best upper bound currently known for this is $\mathcal{O}(m^{2k}v^2)$ time, where *m* and *v* are the number of atoms and the number of variables in Q, respectively [4].
- k bound of 2 or 3 usually suffices for queries
- Even if *m* were larger, it’s common to run the same query over changing underlying data.
  - Pay upfront for computing the HD, but make it up by running a highly optimized query over and over again.

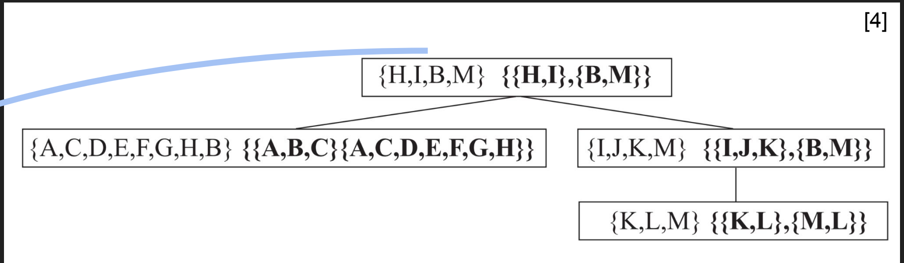
YES, BUT HOW DOES IT WORK?

$$ans() \leftarrow r_1(A, B, C) \wedge r_2(A, C, D, E, F, G, H) \wedge r_3(H, I) \wedge r_4(I, J, K) \wedge r_5(K, L) \wedge r_6(L, M) \wedge r_7(B, M)$$

$$ans() \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H) \wedge r_3(H,I) \wedge r_4(I,J,K) \wedge r_5(K,L) \wedge r_6(L,M) \wedge r_7(B,M)$$

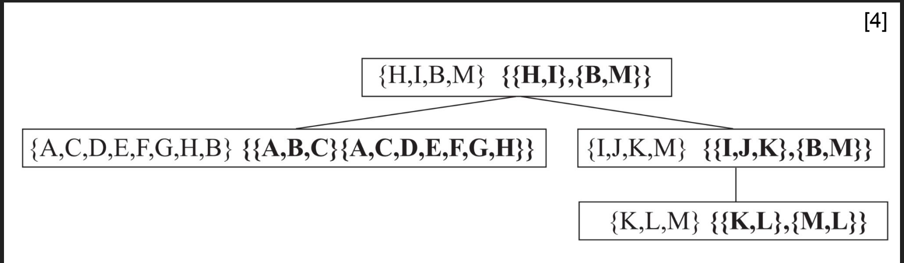


$$ans() \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H) \wedge r_3(H,I) \wedge r_4(I,J,K) \wedge r_5(K,L) \wedge r_6(L,M) \wedge r_7(B,M)$$



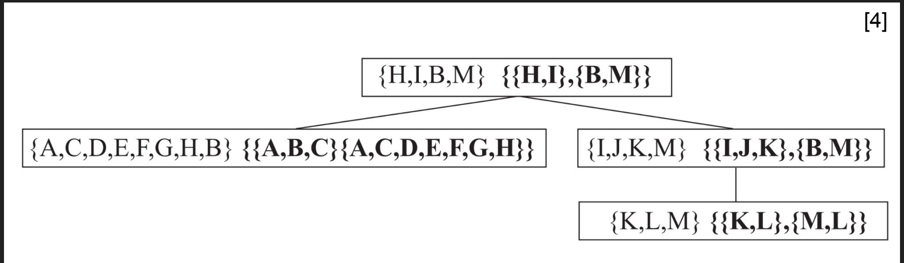
$$ans_1(H,I,B,M) \leftarrow r_3(H,I) \wedge r_7(B,M)$$

$$ans() \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H) \wedge r_3(H,I) \wedge r_4(I,J,K) \wedge r_5(K,L) \wedge r_6(L,M) \wedge r_7(B,M)$$



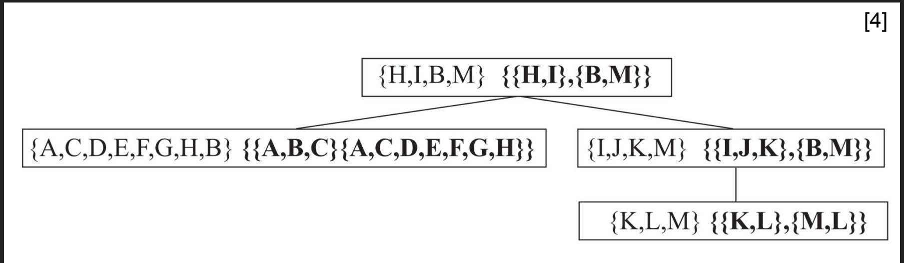
$$ans_1(H,I,M,B) \leftarrow r_3(H,I) \wedge r_7(B,M)$$
$$ans_2(A,B,C,D,E,F,G) \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H)$$
$$ans_3(I,J,K,M) \leftarrow r_4(I,J,K) \wedge r_7(B,M)$$
$$ans_4(K,L,M) \leftarrow r_5(K,L) \wedge r_6(L,M)$$

$$ans() \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H) \wedge r_3(H,I) \wedge r_4(I,J,K) \wedge r_5(K,L) \wedge r_6(L,M) \wedge r_7(B,M)$$



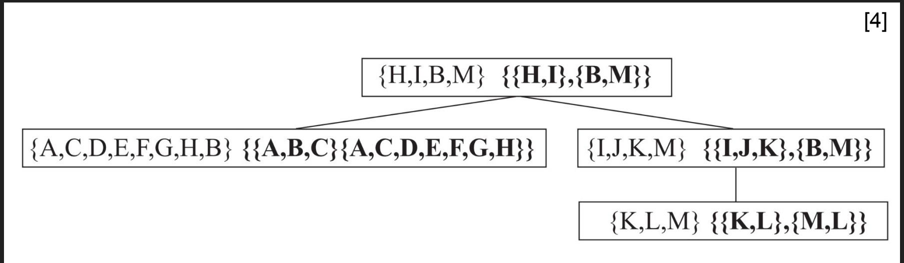
$$ans() \leftarrow ans_1(H,I,M,B) \wedge ans_2(A,B,C,D,E,F,G) \wedge ans_3(I,J,K,M) \wedge ans_4(K,L,M)$$

Cyclic :($ans() \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H) \wedge r_3(H,I) \wedge r_4(I,J,K) \wedge r_5(K,L) \wedge r_6(L,M) \wedge r_7(B,M)$



Acyclic! :D $ans() \leftarrow ans_1(H,I,M,B) \wedge ans_2(A,B,C,D,E,F,G) \wedge ans_3(I,J,K,M) \wedge ans_4(K,L,M)$

Cyclic :($ans() \leftarrow r_1(A,B,C) \wedge r_2(A,C,D,E,F,G,H) \wedge r_3(H,I) \wedge r_4(I,J,K) \wedge r_5(K,L) \wedge r_6(L,M) \wedge r_7(B,M)$



Acyclic! :D $ans() \leftarrow ans_1(H,I,M,B) \wedge ans_2(A,B,C,D,E,F,G) \wedge ans_3(I,J,K,M) \wedge ans_4(K,L,M)$

Yannakakis here we come!

Yannakakis Phase 0 → “bottom-up”

{K,L,M} **{{K,L},{M,L}}**

Join these relations.

Yannakakis Phase 0 → “bottom-up”

{K,L,M} **{{K,L},{M,L}}**

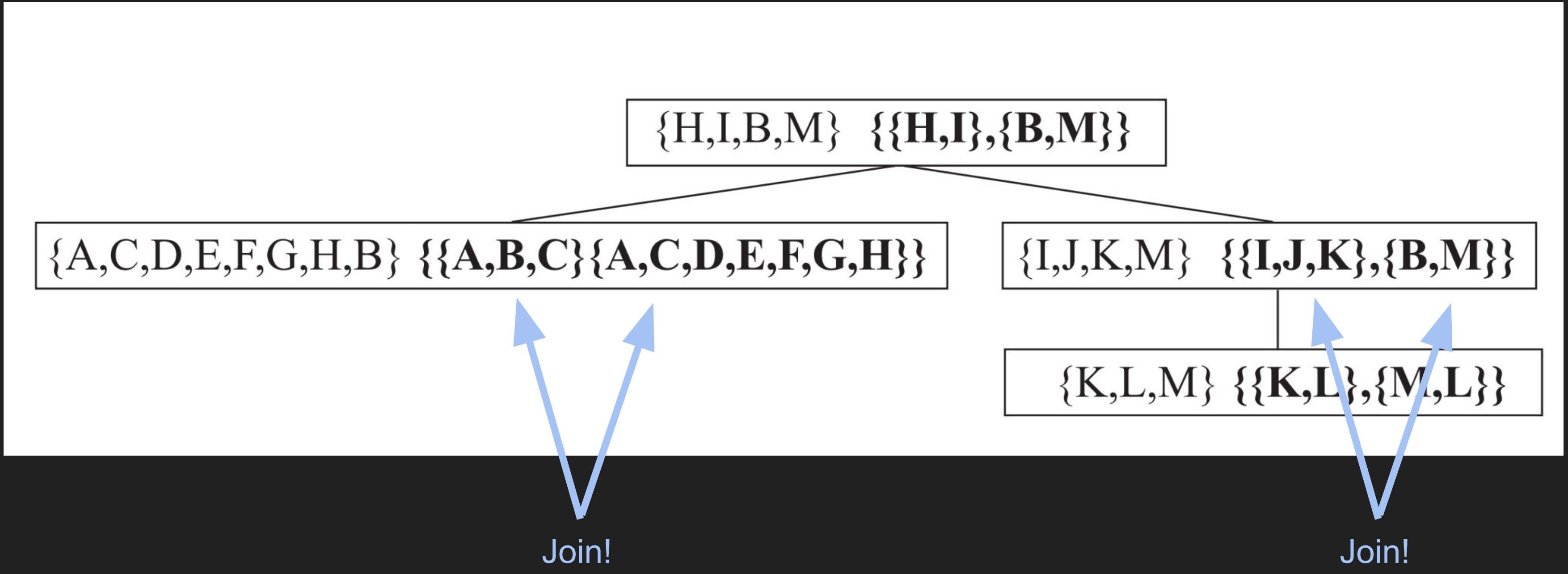
Consider WCOJ if cardinality estimate for the output is greater than the maximum of the input cardinalities! [5][6]

Join these relations.

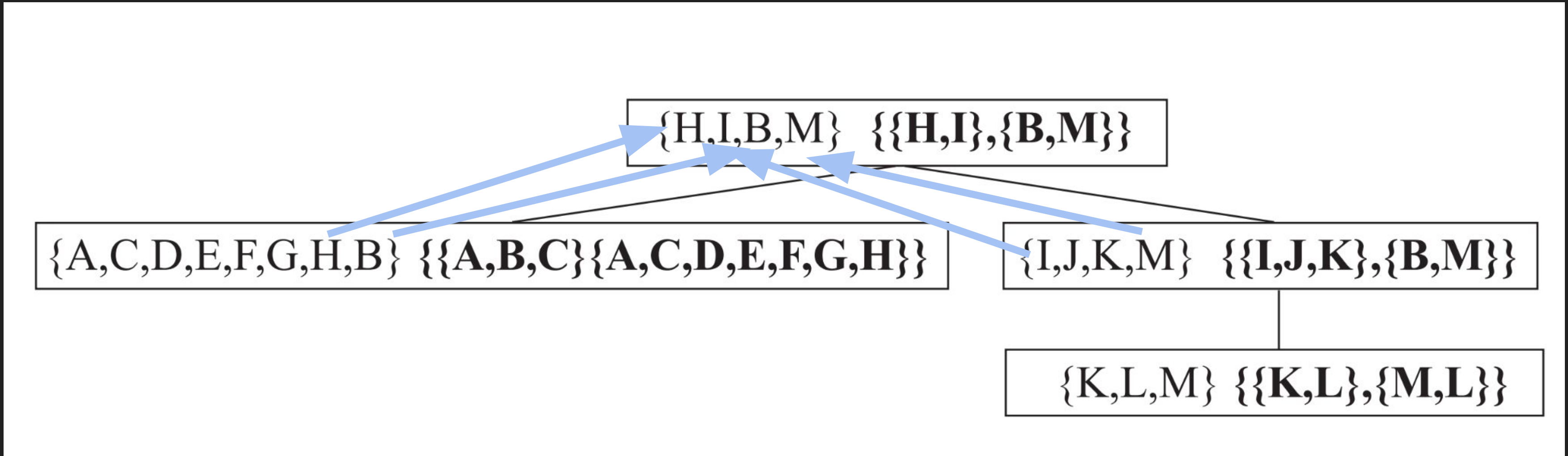
Yannakakis Phase 0 → “bottom-up”

Pass the relations projected onto shared attributes “up”.
K, M

Yannakakis Phase 0 → “bottom-up”

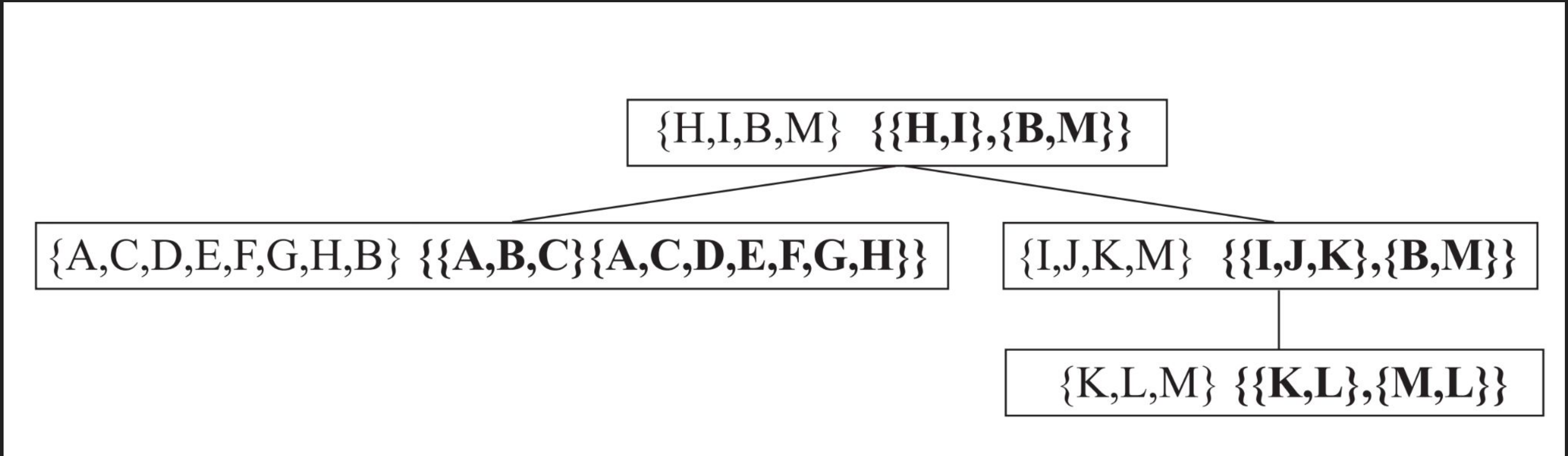


Yannakakis Phase 0 → “bottom-up”



Pass *H,B,I,M “up”.*

Yannakakis Phase 1 → “top-down”



Construct the final result by walking the tree “top-down” and appending values from other attributes.
A,C,D,E,F,G,I

El Fin 👏

- Hypergraphs
  - Extremely useful for variety of domains, including conjunctive queries
  - Fractional edge cover → AGM Bound

- Worst-case optimal join algorithms
  - AGM Bound → WCOJ
  - Best when |output| > max(|input_1, input_2|)
  - Multi-way join—great for cyclic queries!
  - Assumes sorted/indexed relations

- Hypertree decompositions
  - Convert cyclic queries into acyclic queries → ??? → Profit
  - Can use WCOJ within nodes
  - Use Yannakakis between nodes

References

[1] Ngo, H. Q., Ré, C., and Rudra, A. Skew strikes back: New developments in the theory of join algorithms. SIGMOD Rec. 42, 4 (feb 2014), 5–16.

[2] Suciu, D. Cse544 data management lectures 9-10 advanced query processing. https://courses.cs.washington.edu/courses/cse544/21wi/lectures/lecture9-10-agm-wcoj.pdf,2021. Accessed: 2022–05-15.

[3] Aberger, C. R., Lamb, A., Tu, S., Nötzli, A., Olukotun, K., and Ré, C. Emptyheaded: A relational engine for graph processing. ACM Trans. Database Syst. 42, 4 (oct 2017).

[4] Gottlob, G., Greco, G., Leone, N., and Scarcello, F. Hypertree decompositions: Questions and answers. In Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (New York, NY, USA, 2016), PODS '16, Association for Computing Machinery, p. 57–74.

[5] Aberger, C. R., Lamb, A., Olukotun, K., and Ré, C. Levelheaded: Making worst-case optimal joins work in the common case. CoRR abs/1708.07859 (2017).

[6] Freitag, M., Bandle, M., Schmidt, T., Kemper, A., and Neumann, T. Adopting worst-case optimal joins in relational database systems. Proc. VLDB Endow. 13, 12 (jul 2020),1891–1904.