Compressed Representations of Conjunctive Query Results

Paris Koutris

University of Wisconsin-Madison

joint work with **Shaleen Deep** (UW-Madison)

MOTIVATION

- In many scenarios a data processing pipeline repeatedly accesses the result of a join query using some access pattern
- But the result of the query over large data can lead to a large result, and be very expensive to store directly

Can we compress the query result so that we can still allow these accesses to be performed efficiently?

EXAMPLE: GRAPH DATA

Consider the author relation from DBLP:

$$R(author, paper)$$

We want to run a data analysis over the co-author graph, which can be expressed as the following view:

$$V(x, z) \leftarrow R(x, p), R(y, p)$$

EXAMPLE: GRAPH DATA

Graph analytics algorithms access a graph through an API that asks for *the set of neighbors of a given vertex*, expressed by an adorned view:

$$V^{bf}(x, y) \leftarrow R(x, p), R(y, p)$$

- x is a bound (**b**) variable
- z is a free (**f**) variable

[Xirogiannopoulos & Deshpande, '17]

EXAMPLE: GRAPH DATA

$$V^{bf}(x, y) \leftarrow R(x, p), R(y, p)$$

How can we solve this problem?

1. run each access request from scratch
- no extra space needed
- answer time can be $\Omega(N)$

2. create index on materialized V
- space can be $\Omega(N^2)$
- answer in constant time

what exists between the two extremes?

TALK OUTLINE

1. **Problem Setting**
2. Main Result #1
3. Main Result #2
4. Future Work

ADORNED VIEWS

We consider the class of conjunctive queries:

$$V(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$$

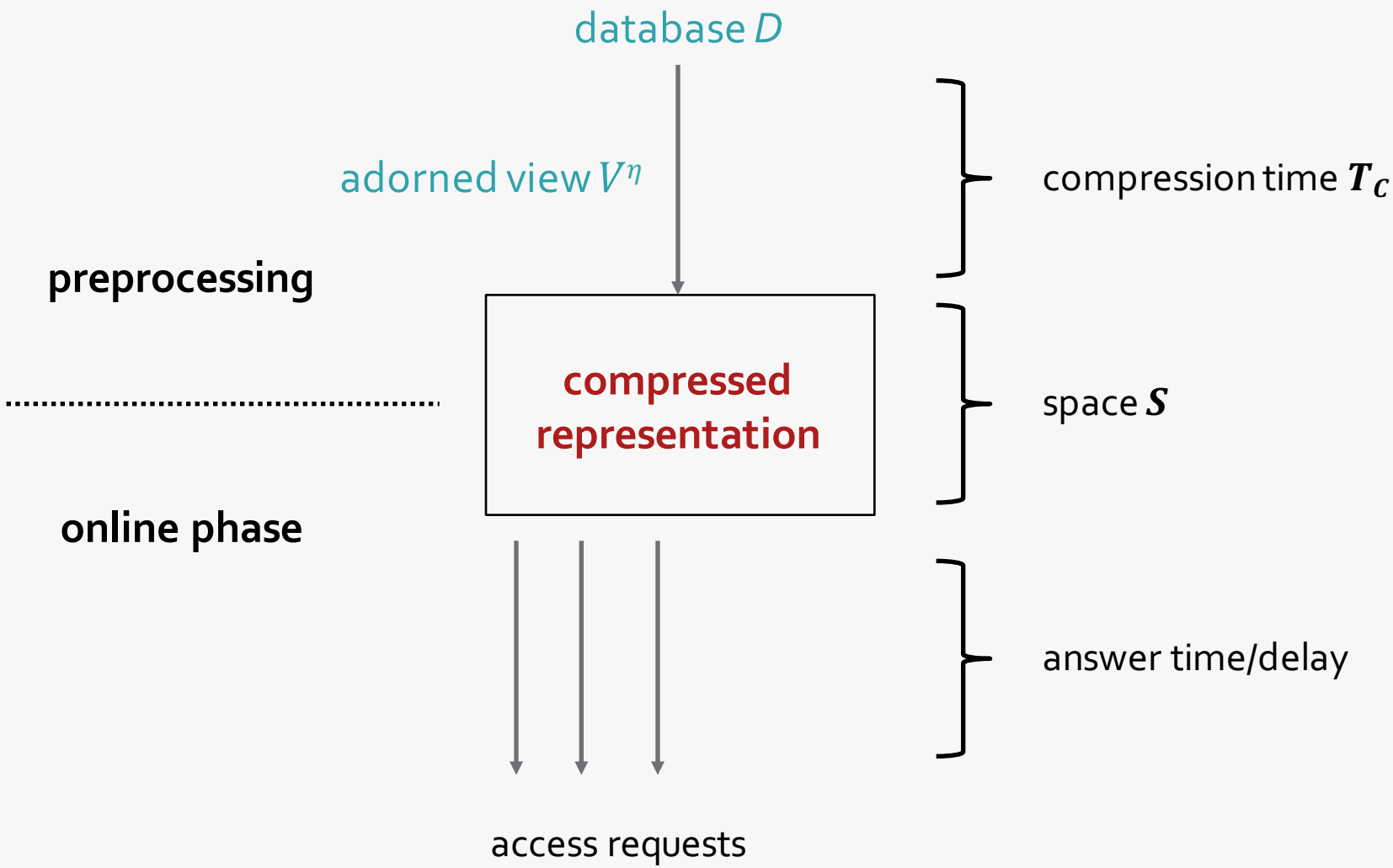
An adorned view [Ullman '85] describes an access pattern where some variables are bound (**b**) and others free (**f**)

$$V^{bbf}(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$$

$$V^{fff}(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$$

An adorned view is full is every variable appears in the head

COMPRESSED REPRESENTATIONS



PARAMETERS

Goal: construct a space-efficient representation to answer access requests that originate from a given adorned view

Parameters:

- compression time T_C
- space: S
- answering time
  - total answer time T_A (*time to enumerate all results*)
  - delay δ (*maximum time between outputting two consecutive tuples*)

FACTORIZED DATABASES

[Olteanu & Závodný 15] Suppose that the adorned view has only free variables and the query is full:

$$V^{f \cdots f}(x_1, \ldots, x_k) \leftarrow \ldots$$

In compression time

$$T_A = O(|D|^{\mathsf{fhw}(Q)})$$

we can construct a compressed representation with space

$$S = O(|D|^{\mathsf{fhw}(Q)})$$

such that we can answer any access request over *D* with constant delay.

* fhw(Q) = *fractional hypertree width* of Q

EXAMPLE

$$|R| = |S| = |T| = N$$

$$V^{bbf}(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$$

	compression time	**space**	**total answer time**	**delay**
do nothing	-	$O(N)$	$O(N)$	$O(N)$
materialize + create index	$O(N^{3/2})$	$O(N^{3/2})$	$O(\|OUT\|)$ *	$O(1)$
our results	$O(N^{3/2})$	$O\left(\frac{N^{3/2}}{\tau}\right)$	$O(\tau\|OUT\|)$	$O(\tau)$

* *OUT* is the output of an access request

ALL BOUND VARIABLES

Suppose that the adorned view has only bound variables:

$$V^{b\cdots b}(x_1, \ldots, x_k) \leftarrow \ldots$$

Then, in time linear to the database size, we can construct a compressed representation with linear space that can answer any access request over *D* with constant delay.

IDEA: simply create a hash index for every relation

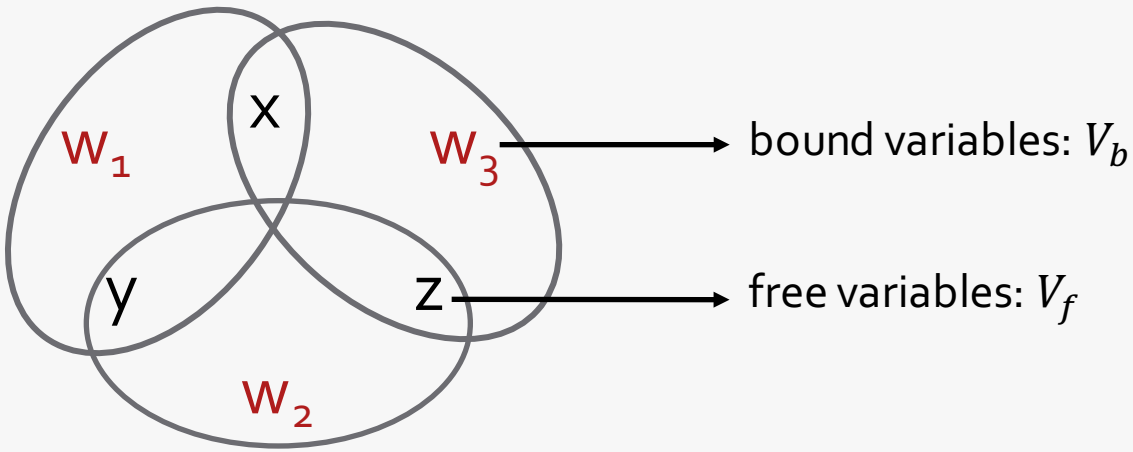
TALK OUTLINE

1. Problem Setting
2. **Main Result #1**
3. Main Result #2
4. Future Work

QUERY AS A HYPERGRAPH

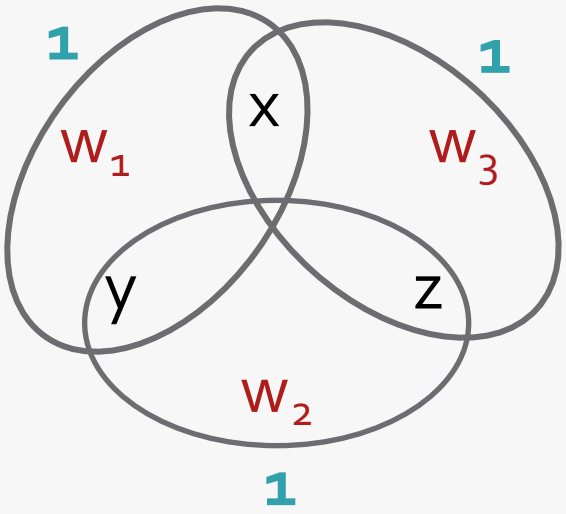
Given an adorned view Q^{η}, it will be convenient to view it as a hypergraph $H = (V, E)$

$$Q^{fffbbb}(x, y, z, w_1, w_2, w_3) \leftarrow R_1(w_1, x, y), R_2(w_2, y, z), R_3(w_3, x, z)$$



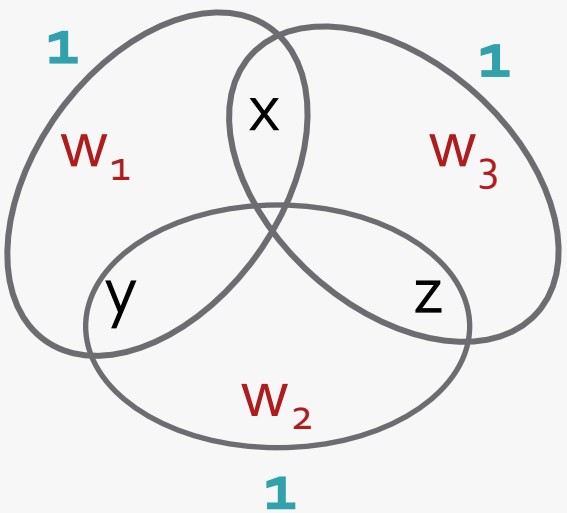
FRACTIONAL EDGE COVER

fractional edge cover: assign a weight to each hyperedge such that for every variable, the sum of the weights that include it is at least 1



SLACK

slack: given a fractional edge cover $\boldsymbol{u}$, and a subset S of the variables, the slack $\alpha(S)$ is the maximum quantity such that $\boldsymbol{u}/\alpha(S)$ is still a fractional cover of S



$$V_f = \{x, y, z\}$$
$$\alpha(V_f) = 2$$

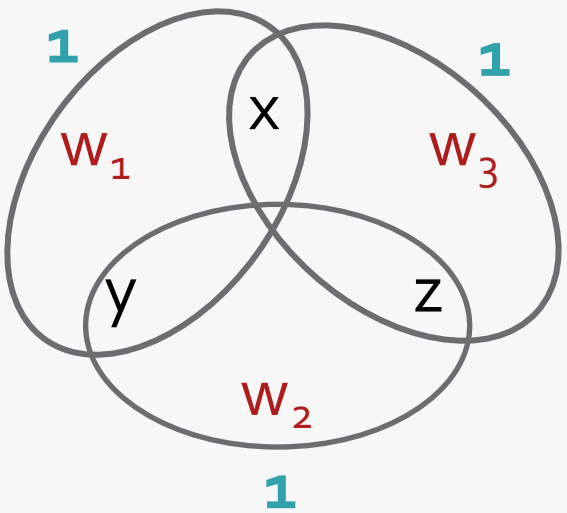
the slack is always at least one!

AGM BOUND

Let $H = (V, E)$ be a hypergraph.

For every fractional edge cover $\boldsymbol{u}$ of V, the output size of the corresponding join query is upper bounded by

$$\prod_{F \in E} |R_F|^{u_F}$$



In our example, if all relations have size N, we obtain a bound of N^3

MAIN THEOREM #1

Q^η : full adorned view with hypergraph $H = (V, E)$

$\boldsymbol{u}$: any fractional edge cover of V

For any database D and parameter $\tau > 0$, we can construct a compressed representation with:

$$\text{compression time } T_C = \tilde{O}(|D| + \prod_{F \in E} |R_F|^{u_F})$$

$$\text{space } S = \tilde{O}(|D| + \prod_{F \in E} |R_F|^{u_F} / \tau^{\alpha(V_f)})$$

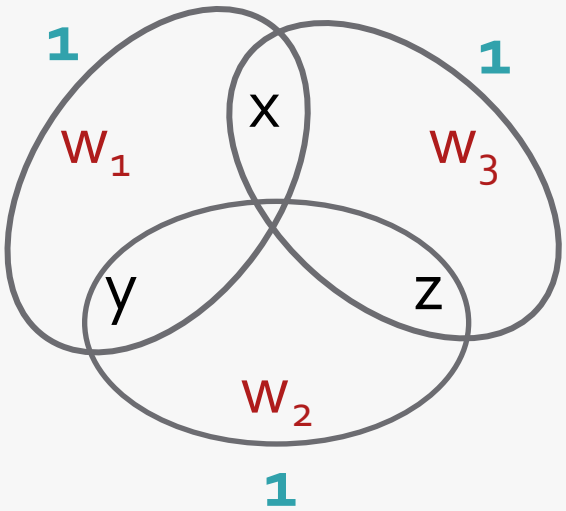
$$\text{delay } \delta = \tilde{O}(\tau)$$

$$\text{answer time } T_A = \tilde{O}(|q(D)| + \tau \cdot |q(D)|^{1/\alpha(V_f))})$$

EXAMPLE

$$\mathbf{u} = (1, 1, 1)$$

$$\alpha(V_f) = 2, \quad V_f = \{x, y, z\}$$



$$\text{compression time } T_C = \tilde{O}(N^3)$$

$$\text{space } S = \tilde{O}(N^3/\tau^2)$$

$$\text{delay } \delta = \tilde{O}(\tau)$$

$$\text{answer time } T_A = \tilde{O}(|q(D)| + \tau \cdot |q(D)|^{1/2})$$

THE DATA STRUCTURE (1)

- Consider an ordering of the free variables V_f

  e.g. $x \leq y \leq z$

- This induces a lexicographic ordering for all the valuations over V_f
- Using this ordering, we can define intervals:

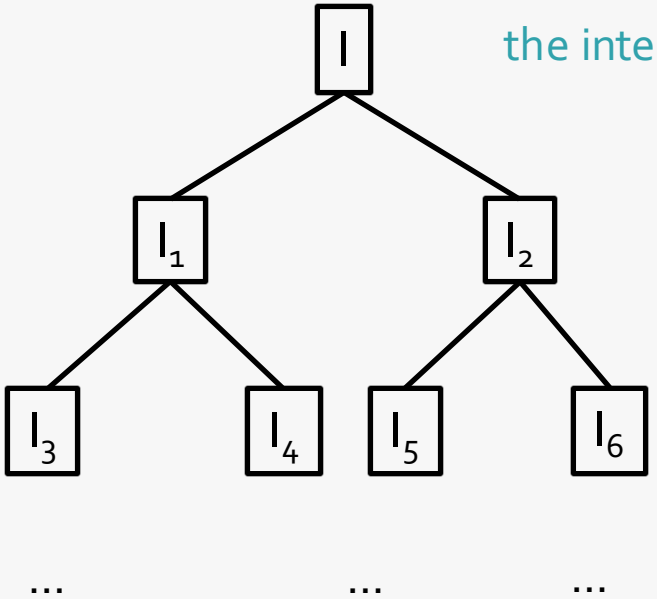
$$I_1 = [(0,0,10),(0,10,20)]$$
$$I_2 = [(3,1,0),(4,5,0))$$

- Given a valuation v_b over V_b and an interval I, we can estimate an upper bound $\boldsymbol{T(v_b, I)}$ on the cost of computing the query restricted on v_b, I using the AGM bound

THE DATA STRUCTURE (2)

- The data structure is a binary tree parameterized by a threshold $\boldsymbol{\tau}$
- Each node is labeled by an interval I
- Each node stores a bit for every valuation over v_b over V_b with cost $T(v_b, I) > \tau$:
  - **0**: the query over v_b, I is empty
  - **1**: the query over v_b, I is not empty

THE DATA STRUCTURE (3)



the interval in the root node includes all valuations

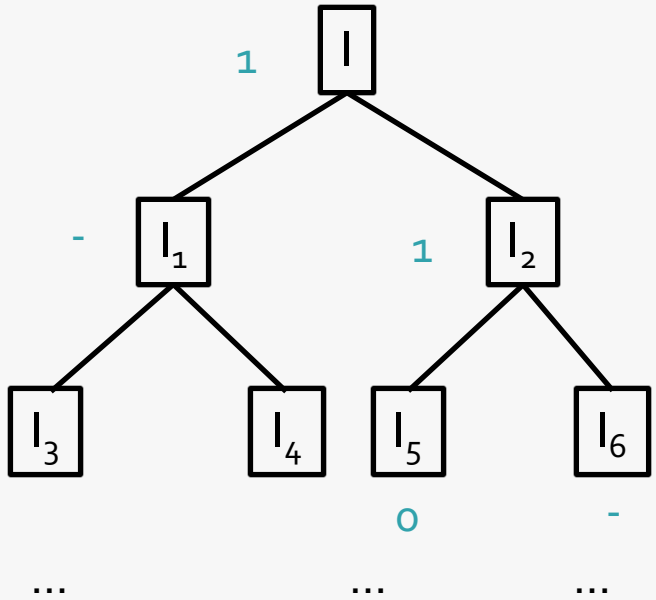
at the next level, the interval of the parent is split into two smaller intervals

we stop at $\log|D|$ levels

We split the intervals such that the bits we need to store at the two sub-intervals is balanced

USING THE DATA STRUCTURE

We are given a valuation v_b over V_b



starting from the root:

- if there is not a bit set, we run the query on the interval
- if bit =0, we exit the node
- if bit = 1, we visit the left and then the right child

The delay is bounded by the threshold τ

COROLLARY OF THEOREM #1

Q^η : full adorned view with hypergraph $H = (V, E)$

$\rho(H)$: minimum fractional edge cover

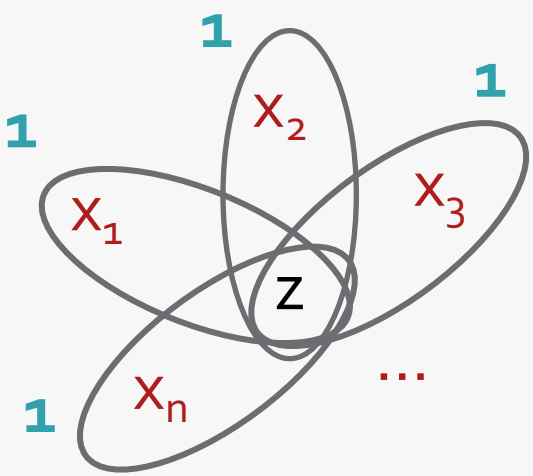
For any input database D and parameter $\tau > 0$, we can construct a compressed representation with:

$$\text{space } S = \tilde{O}(|D| + |D|^{\rho(H)}/\tau)$$

$$\text{delay } \delta = \tilde{O}(\tau)$$

For $\tau = 1$, the space matches the AGM bound

BETTER BOUNDS USING SLACK



Star join query

- fractional edge cover assigns weight 1
- the slack for $\{z\}$ in this case is n

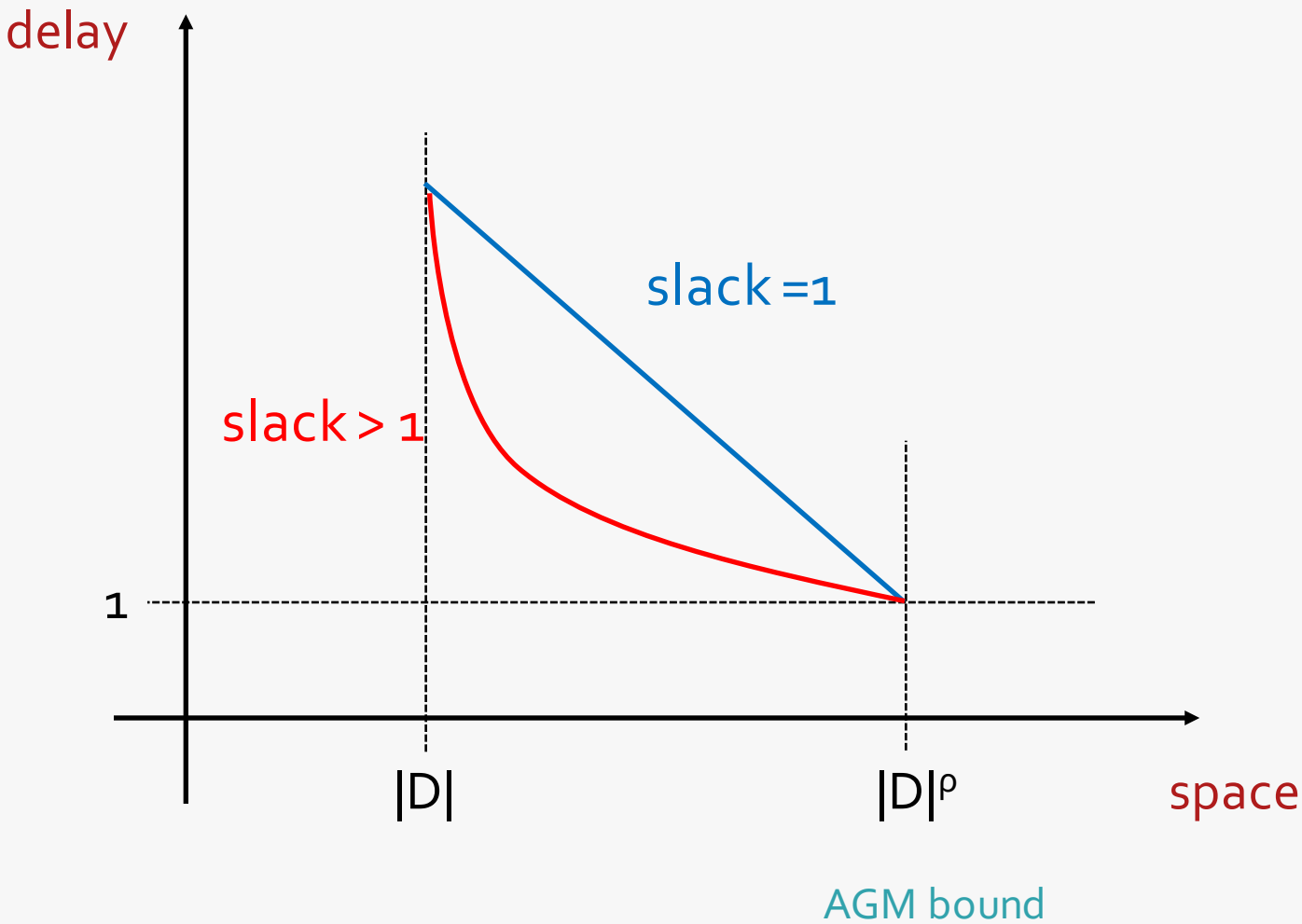
$$\text{space } S = \tilde{O}(|D| + |D|^n/\tau^n)$$

$$\text{delay } \delta = \tilde{O}(\tau)$$

$$\text{answer time } S = \tilde{O}(|q(D)| + \tau|q(D)|^{1/n})$$

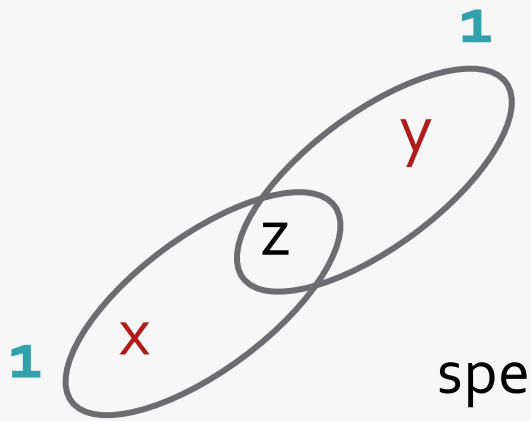
If we ignored slack, the space would be $|D|^n/\tau$

DELAY-SPACE TRADEOFF



FAST SET INTERSECTION

Given a family of sets $\{ S_1, \ldots, S_n \}$ with total size m, construct a space-efficient data structure such that given any i, j we can compute $S_i \cap S_j$ as fast as possible [Cohen & Porat '10]



$$Q^{bbf}(x, y, z) \leftarrow R(x, z), R(y, z)$$

special case of the Theorem #1:
$R(s, b)$ encodes that set s contains element b

LIMITATIONS OF THEOREM #1

Consider the following adorned view:

$$Q^{fff}(x, y, z) \leftarrow R(x, z), S(y, z)$$

- Theorem #1 implies that for constant delay we need space $O(N^2)$
- But we know that we can achieve the same delay with only linear space (because of acyclicity)
- Why is there a mismatch in the space bounds?

We must take the query structure into account as well

TALK OUTLINE

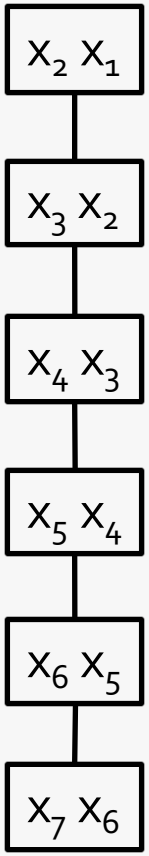
1. Problem Setting
2. Main Result #1
3. **Main Result #2**
4. Future Work

TREE DECOMPOSITION

$$Q(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$

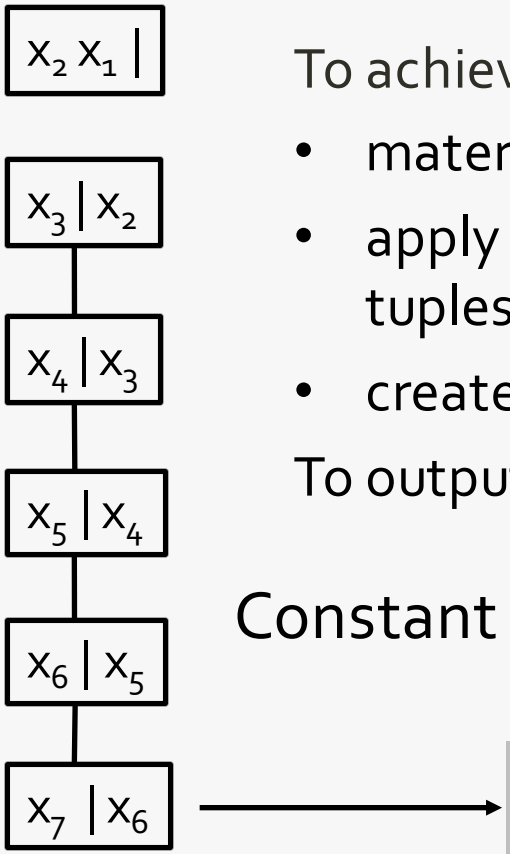
Given a hypergraph $H = (V, E)$, a tree decomposition of H is a tuple $(T, (B_t))$ where T is a tree, and each bag B_t is a subset of V such that:

- each edge is contained in some bag
- for each variable x, the set of tree nodes that contain x in their bag is connected



FULL ENUMERATION WITH CONSTANT DELAY

$$Q^{f\cdots f}(x_1,\ldots,x_7) \leftarrow R_1(x_1,x_2), R_2(x_2,x_3),\ldots,R_6(x_6,x_7)$$



To achieve constant delay:

- materialize the output of every bag
- apply a bottom up semi-join reduction to remove tuples that do not contribute to the result
- create a hash index for each bag

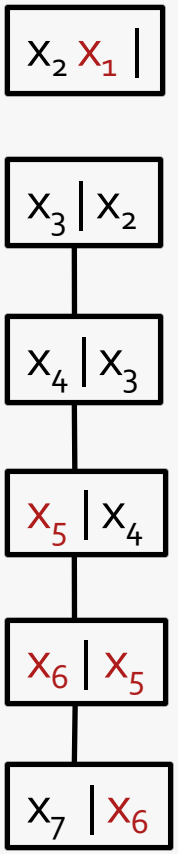
To output the result, traverse top down

Constant delay with space $S = O(|D|^{\text{fhw}(Q)})$

this creates an index that given a value of x_6 returns all matching values of x_7

CONSTANT DELAY WITH BOUND VARIABLES

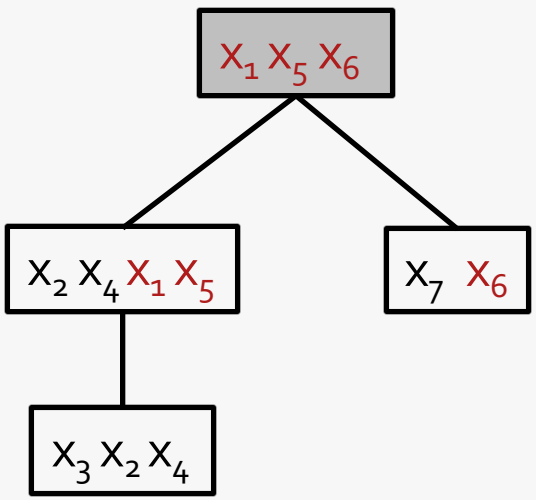
$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



- The previous construction fails when we have bound variables
- To overcome this issue, we need to use a more restricted type of tree decomposition!

CONNEX TREE DECOMPOSITION

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



Hypergraph $H = (V, E)$ and $C \subseteq V$.

A C-connex tree decomposition is a tuple (T, A) such that:

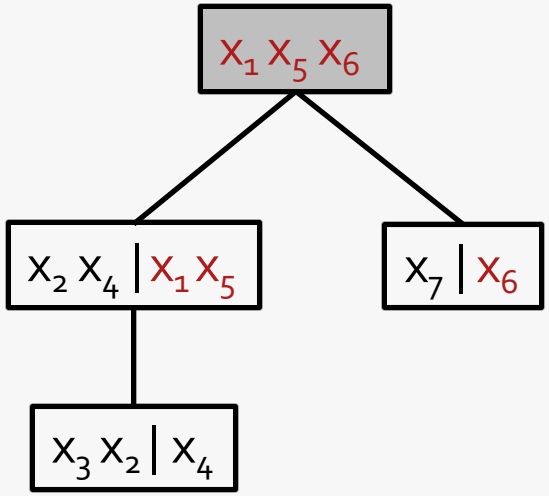
- T is a tree decomposition of H
- A is a connected subset of nodes such that their bags contain exactly C

$$C = V_b = \{x_1, x_5, x_6\}$$

CONSTANT DELAY WITH BOUND VARIABLES

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$

A is marked with gray



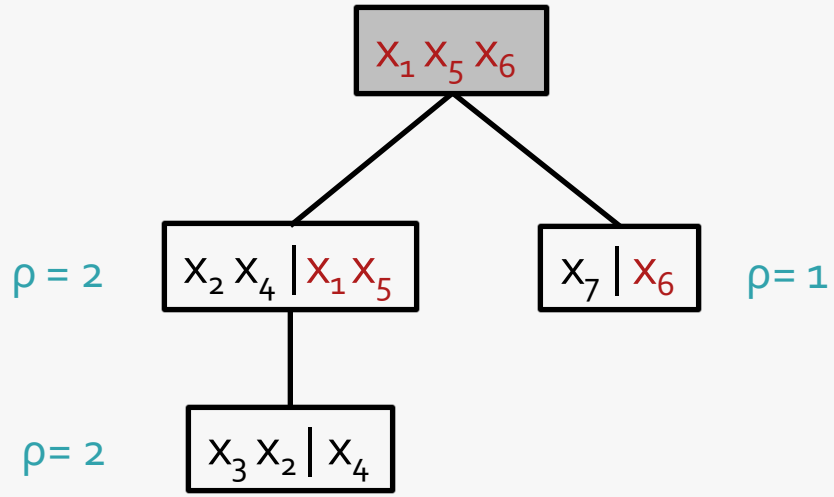
To achieve constant delay:

- root the tree from *A*
- materialize the output of every bag **not in** *A*
- apply a bottom up semi-join reduction (ignoring the bags in *A*)
- create a hash index for each bag (**not in** *A*)
- for every relation contained in some node of *A*, create an index that checks existence

To output the result, traverse top down

FRACTIONAL HYPERTREE WIDTH

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



$$\text{fhw}(Q) = 1$$

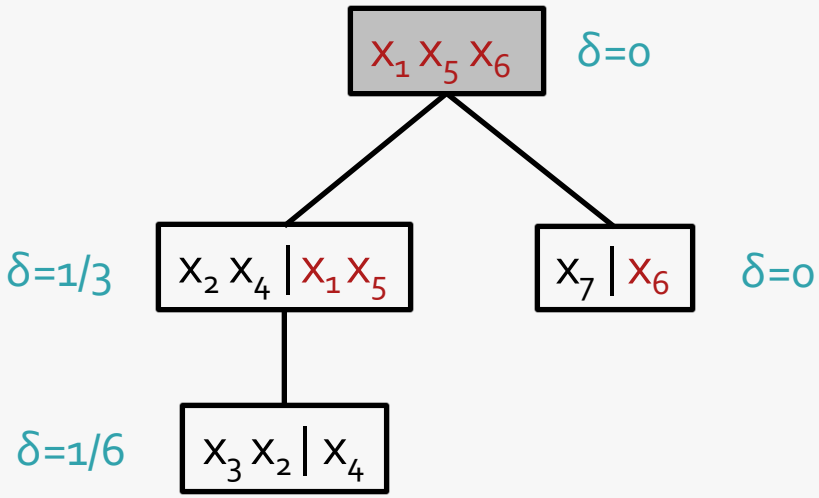
$$\text{fhw}(Q \mid V_b) = 2$$

fhw(decomposition) = maximum ρ over all bags
fhw(Q) = minimum fhw over all tree decompositions of Q
fhw(Q | C) = minimum fhw over all C-connex tree decompositions of Q

Constant delay with space $S = O(|D|^{\text{fhw}(H|V_b)})$

BEYOND CONSTANT DELAY

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



assign a number $\delta(t)$ to each bag such that we achieve delay $|D|^{\delta(t)}$ when we traverse the bag

Using Theorem #1, to achieve delay $|D|^{\delta(t)}$ when traversing bag t with edge cover **u** we need space

$$|D|^{\sum_F u_F - \delta(t)\alpha(V_b^t)}$$

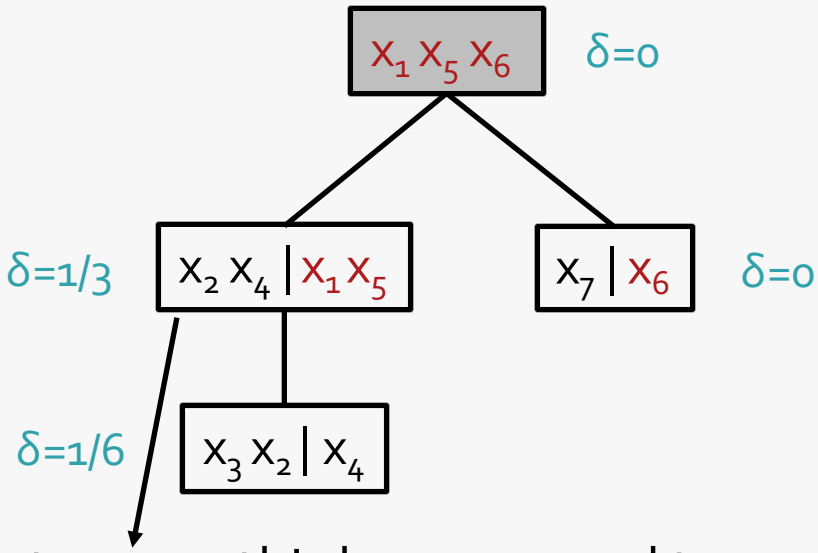
Hence, we must find **u** that minimizes

$$\sum_F u_F - \delta(t)\alpha(V_b^t)$$

$\rho^+(t)$ = minimum quantity

BEYOND CONSTANT DELAY

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



To traverse this bag, we need to answer the following adorned query:

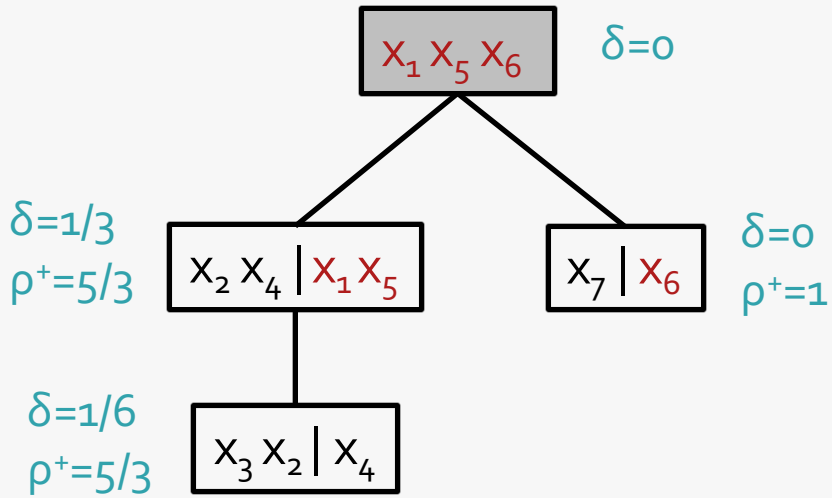
$$Q^{ffbb}(x_2, x_4, x_1, x_5) \leftarrow R_1(x_1, x_2), R_2'(x_2), R_3'(x_4), R_4(x_4, x_5), R_5'(x_5)$$

The optimal edge cover has $u_1 = u_4 = 1$. Then

$$\rho^+(t) = (1 + 1) - 1/3 \cdot 1 = 5/3$$

BEYOND CONSTANT DELAY

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



$$\text{fh } \delta\text{-width} = 5/3$$

$$\delta\text{-height} = 1/2$$

fh δ-width(decomposition) = maximum ρ^+ over all bags

- captures space

δ-height(decomposition) = maximum weight root-to-leaf path (using the delay as the weight)

- captures delay

MAIN THEOREM #2

Q^η : full adorned view with hypergraph $H = (V, E)$

V_b-connex tree decomposition with delay assignment δ:

- fh δ-width ***f***
- δ-height ***h***

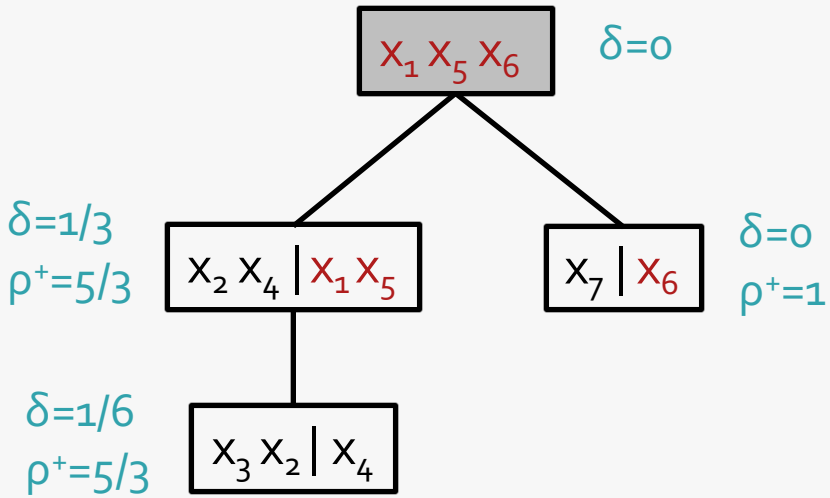
For any input database *D*, we can construct a compressed representation with:

$$\text{space } S = \tilde{O}(|D| + |D|^f)$$

$$\text{delay } \delta = \tilde{O}(|D|^h)$$

EXAMPLE

$$Q^{bfffbbf}(x_1, \ldots, x_7) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_6(x_6, x_7)$$



fh δ-width $= 5/3$

δ-height $= 1/2$

$$\text{space } S = \tilde{O}(|D| + |D|^{5/3})$$

$$\text{delay } \delta = \tilde{O}(|D|^{1/2})$$

A FEW COMMENTS

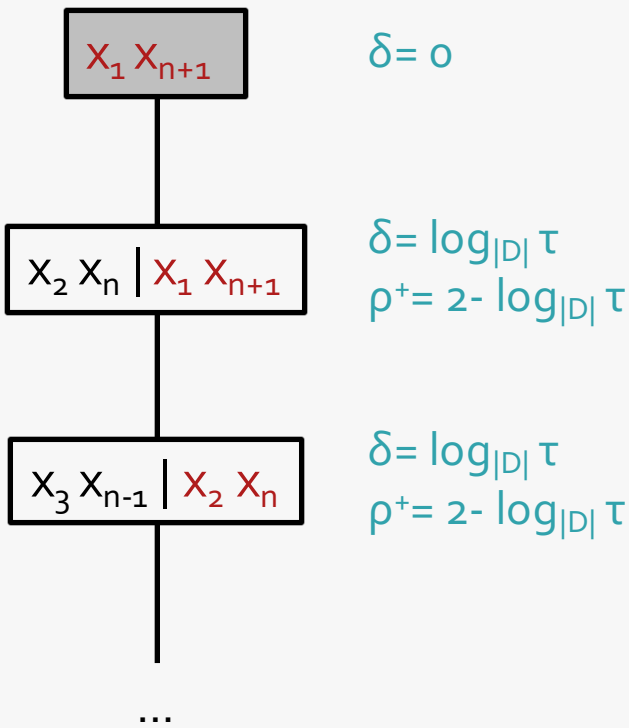
- The new notion of δ-width is parametrized by the delay assignment
- When the delay assignment is $\delta = 0$ everywhere, then we recover the standard notions of fractional hypertree width

Given an adorned view and a connex tree decomposition:

- Given a space constraint, we can compute in **PTIME** the delay assignment and width that minimizes delay
- Given a delay constraint, we can compute in **PTIME** the delay assignment and width that minimizes space

ADDITIONAL EXAMPLE

$$P_n^{bf\cdots fb}(x_1,\ldots,x_{n+1}) \leftarrow R_1(x_1,x_2), R_2(x_2,x_3),\ldots,R_n(x_n,x_{n+1}).$$



$$\text{fh } \delta\text{-width} = 2 - \log_{|D|}\tau$$

$$\delta\text{-height} = \lfloor n/2 \rfloor \cdot \log_{|D|}\tau$$

$$\text{space } S = \tilde{O}(|D| + |D|^2/\tau)$$

$$\text{delay } \delta = \tilde{O}(\tau^{\lfloor n/2 \rfloor})$$

TALK OUTLINE

1. Problem Setting
2. Main Result #1
3. Main Result #2
4. **Future Work**

LOWER BOUNDS

k-SetDisjointness: Given a family of sets $\{ S_1, \ldots, Sn \}$ with total size m, construct a space-efficient data structure such that given any k set indexes we can decide efficiently whether their intersection is empty.

Conjecture [Goldstein et al. '17] Any data structure that answers k-SetDisjointess queries in time *T* must use space $\Omega(m^k / T^k)$

Considered to be a hard open problem in data structures!

BEYOND FULL ADORNED VIEWS

- How do we handle the case where variables are projected out?

$$V^{bf}(x, y) \leftarrow R(x, p), R(y, p)$$

- How can we handle other access patterns?
  - aggregation
  - learning

MORE OPEN QUESTIONS

Is it possible to obtain guarantees that go beyond delay?

- average-case analysis
- guarantees w.r.t. to an access request workload
- guarantees that are data-specific

What about dynamic data structures?

How does our data structure behave in practice?

THANK YOU !