Efficiently Answering Quantile Queries

Nofar Carmeli

Based on: PODS 21, TODS special issue?, PODS 22

Joint work with: Karl Bringmann, Wolfgang Gatterbauer,
Benny Kimelfeld, Stefan Mengel, Mirek Riedewald, Nikolaos Tziavelis

Factorized Databases Workshop, August 2022

Content

- **Task**
- Dichotomy for ideal time complexity:
  - Hardness
  - Algorithm
- Solutions for hard cases:
  - Functional dependencies
  - Selection problem
  - Extended preprocessing
- Concluding remarks

Talk focus:
Join queries, lex orders

Example

Employees

Name	Role	Address
Jack	Junior dev	Boston
Jill	Senior dev	Brookline
Joanna	Senior dev	Braintree

Remuneration

Period	Role	Salary
11/2020	Junior dev	4000
11/2020	Senior dev	4500
12/2020	Junior dev	7000
12/2020	Senior dev	7100

Travel

Address	Cost
Boston	50
Brookline	100
Braintree	200

- What is the median monthly cost of an employee?
  - Solution 1:
    join, sort, access the middle
  - Solution 2:
    count, ranked enumeration until the middle
  - Solution 3:
    count, ranked access to the middle

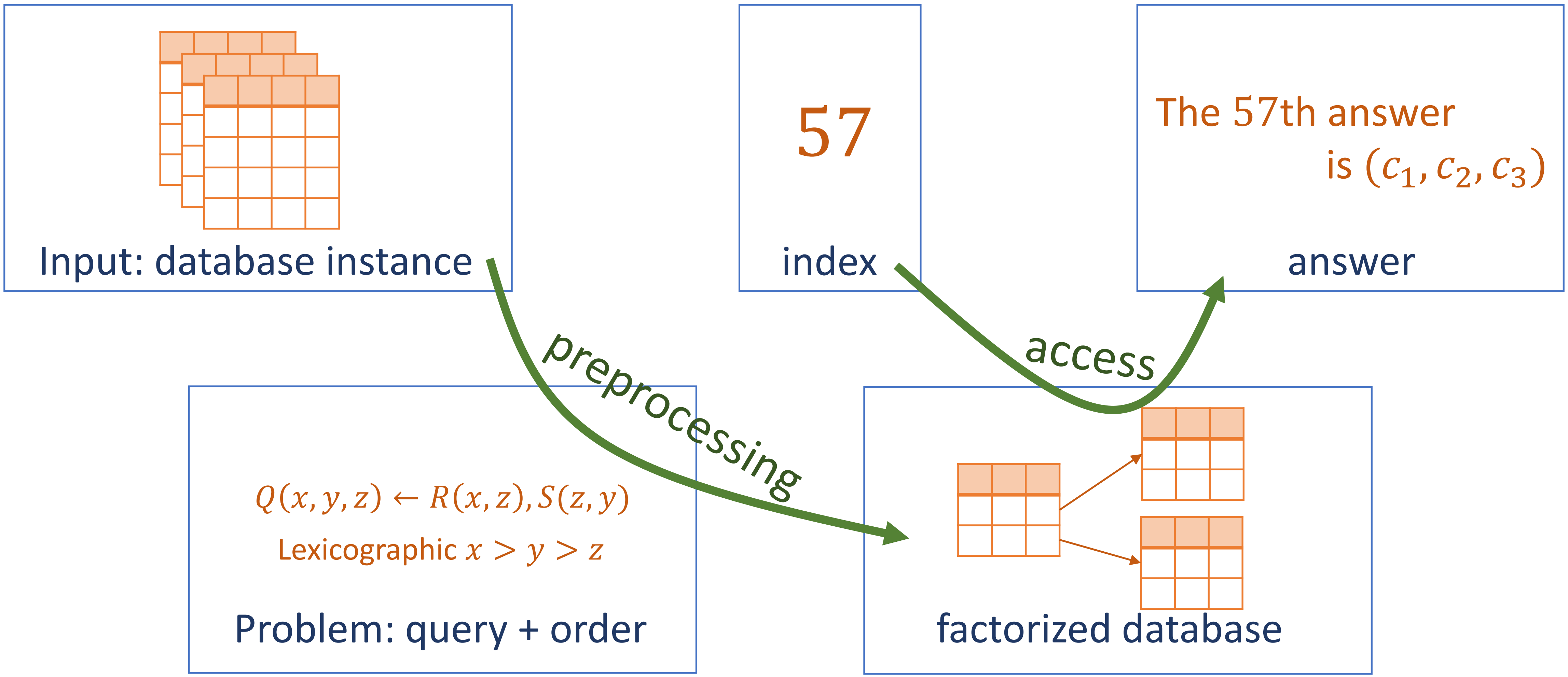
Join Results

Name	Role	Address	Period	Salary	Cost
Jack	Junior dev	Boston	11/2020	4000	50
Jill	Senior dev	Brookline	11/2020	4500	100
Joanna	Senior dev	Braintree	11/2020	4500	200
Jack	Junior dev	Boston	12/2020	7000	50
Jill	Senior dev	Brookline	12/2020	7100	100
Joanna	Senior dev	Braintree	12/2020	7100	200

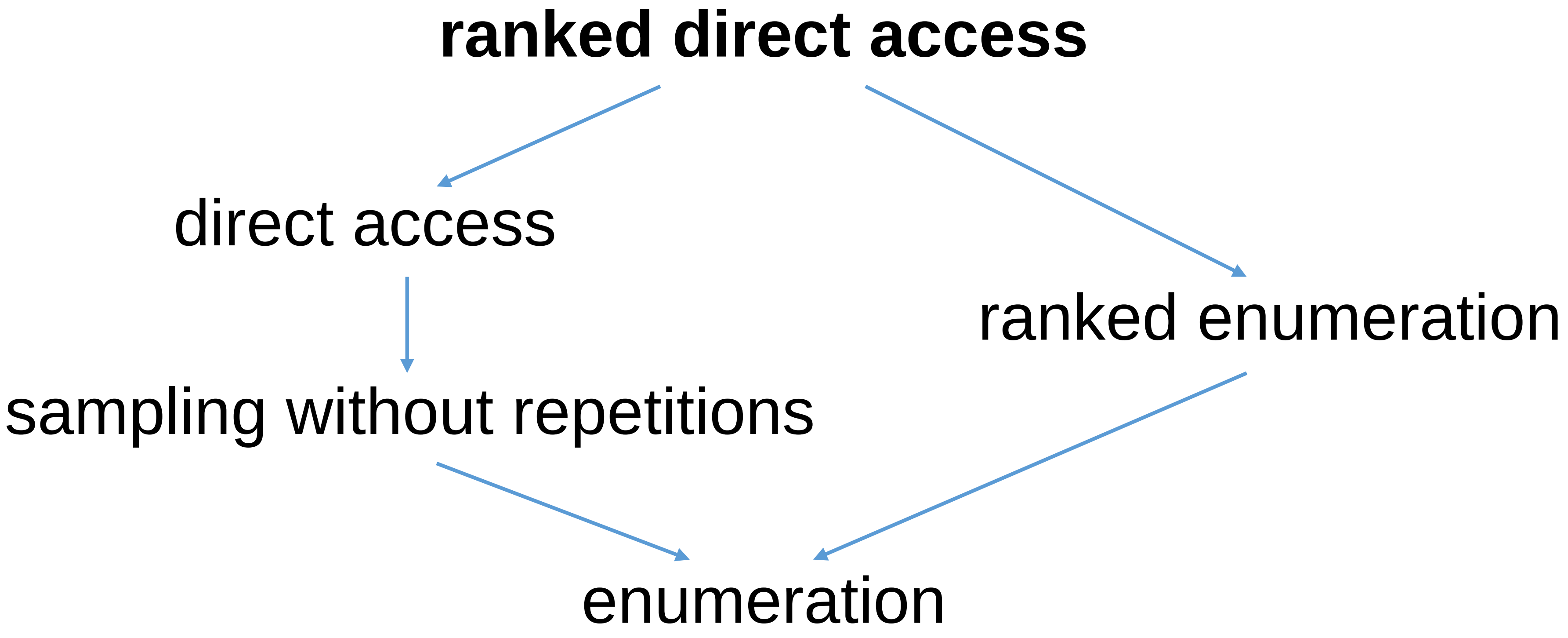
3rd

Count = 6

Goal: efficient ranked access



Connection to other problems



Content

- Task
- **Dichotomy for ideal time complexity:**
  - Hardness
  - Algorithm
- Solutions for hard cases:
  - Functional dependencies
  - Selection problem
  - Extended preprocessing
- Concluding remarks

Talk focus:
Join queries, lex orders

What is the best achievable performance?

<u>Ideal time guarantees:</u>

Preprocessing: linear (to read the input)

Access: constant

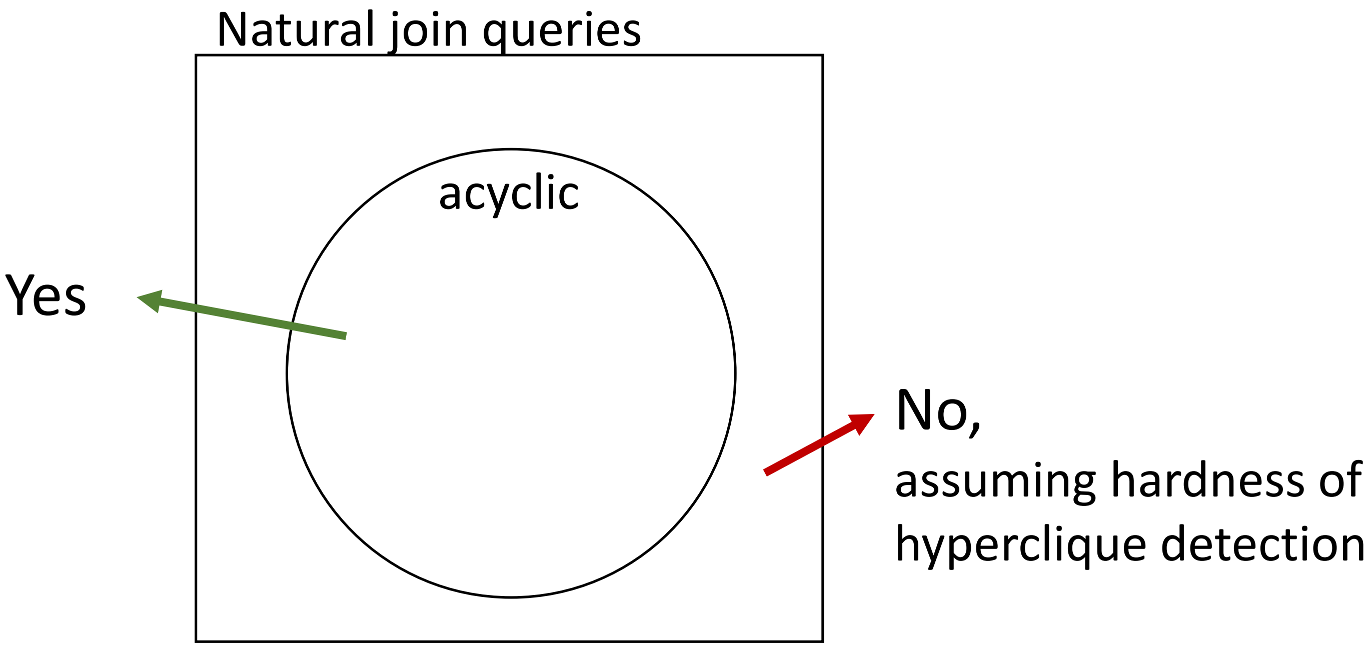
Allow log factors, data complexity

- Can we always achieve that?

Background: enumeration dichotomy

[Yannakakis 1981][Brault-Baron 2013]

Can the query be solved with linear preprocessing and constant delay?



No linear preprocessing constant access for cyclic joins

Acyclicity

- A query that has a join tree is called <u>acyclic</u>

3. For every variable:
the nodes containing it form a subtree

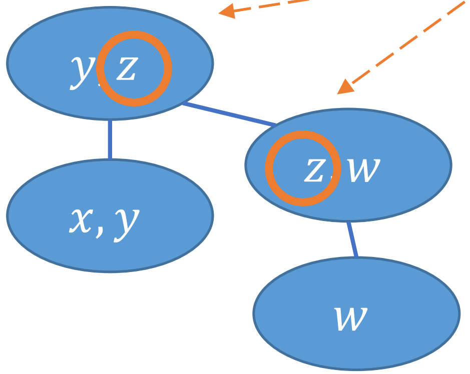
2. tree

1. a node for every atom

Query: $Q_1(x, y, z, w) \leftarrow R(x, y), S(y, z), T(z, w), U(w)$

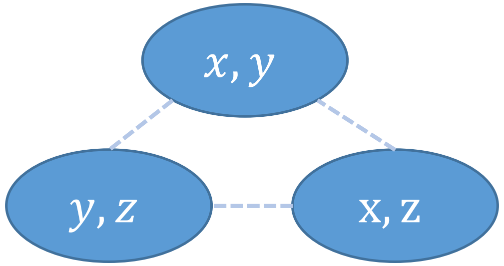
acyclic

Join Tree:



cyclic

Query: $Q_2(x, y, z) \leftarrow R(x, y), S(y, z), T(x, z)$



What is the best achievable performance?

<u>Ideal time guarantees:</u>

Preprocessing: linear (to read the input)

Access: constant

Allow log factors, data complexity

- Can we always achieve that?

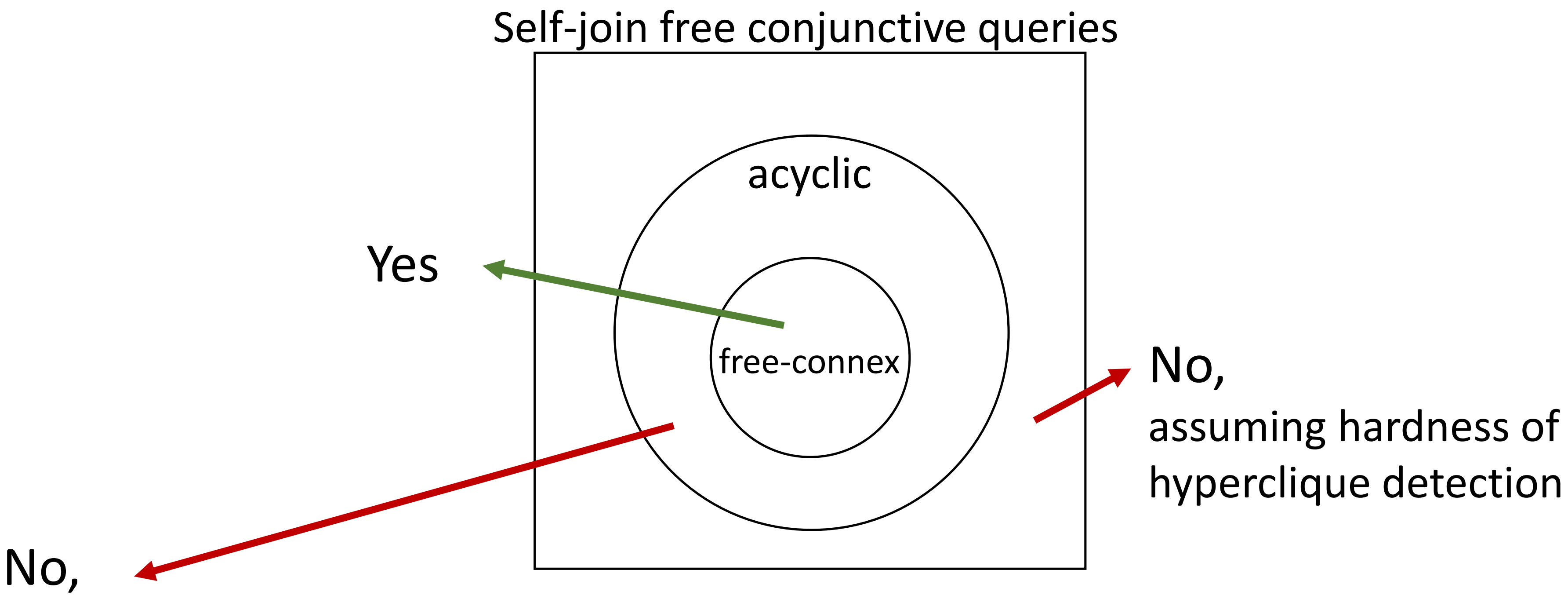
  No (never for cyclic joins)

- Can we always achieve that if the query is acyclic?

Background: enumeration dichotomy

[Bagan, Durand, Grandjean; CSL 07] [Brault-Baron 13]

Can the query be solved with linear preprocessing and constant delay?



assuming hardness of Boolean matrix multiplication

Example for non-free-connex CQ: $Q_1(v_1, v_2) \leftarrow R(v_1, v_3), S(v_3, v_2)$

Enumeration with Projections via Ranked Access

- Reduction:

v_1	v_2	v_3
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_1	c_3
a_1	b_1	c_4
a_1	b_1	c_5
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_1

binary search for next different v_1, v_2 values

Enumerate

$$Q_1(v_1, v_2) \leftarrow R(v_1, v_3), S(v_3, v_2)$$

Not free-connex

using

Lexicographic access

$$Q_2(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$$

Log number of direct-access calls between answers

Q_1 has no enumeration with polylog delay $\Rightarrow$ Q_2 has no lexicographic access with polylog access time

What is the best achievable performance?

<u>Ideal time guarantees:</u>

Preprocessing: linear (to read the input)

Access: constant

Allow log factors, data complexity

- Can we always achieve that?

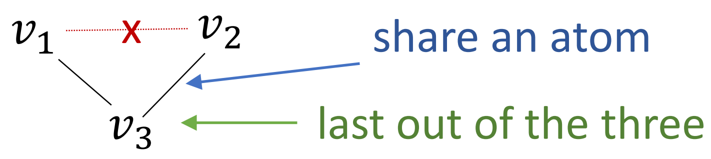
  No (never for cyclic joins)
- Can we always achieve that if the query is acyclic?

  No

Hardness Result

- Can be extended whenever there is a disruptive trio

Def: disruptive trio



- Example: $Q_1(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$

What is the best achievable performance?

<u>Ideal time guarantees:</u>

  Preprocessing: linear (to read the input)

  Access: constant

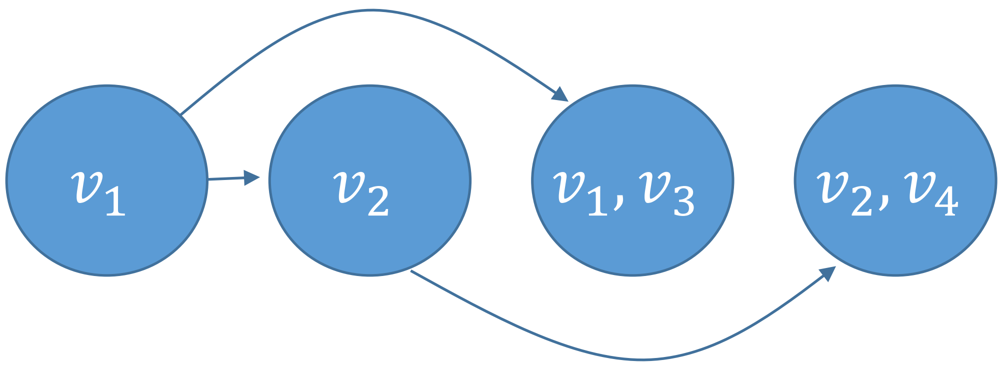
Allow log factors, data complexity

- Can we always achieve that?
  No (never for cyclic joins)
- Can we always achieve that if the query is acyclic?
  No (never if there is a disruptive trio)
- Can we always achieve that if the query is acyclic and without disruptive trios?

Algorithm

[**C**, Zeevi, Berkholz, Kimelfeld, Schweikardt; PODS 20]

- Preprocessing:
  - DP up the tree
  - computes how many answers in a subtree use each tuple
- Access:
  - recurse down the tree
  - splits the desired index between the children



Access 6

v_1	w
a1	8
a2	4

$6 = 1 \cdot 4 + 2$

Access 1

v_1	v_3	w
a1	c1	1
a1	c2	1
a2	c2	1

Access 2

v_2	w
b1	3
b2	1

Σw=4

v_2	v_4	w
b1	d1	1
b1	d2	1
b1	d3	1
b2	d4	1

Algorithm

[**C**, Zeevi, Berkholz, Kimelfeld, Schweikardt; PODS 20]

- Preprocessing:
  - DP up the tree
  - computes how many answers in a subtree use each tuple
- Access:
  - recurse down the tree
  - splits the desired index between the children

Resulting order:

v_1	v_3	v_2	v_4
a1	c1	b1	d1
a1	c1	b1	d2
a1	c1	b1	d3
a1	c1	b2	d4
a1	c2	b1	d1
a1	c2	b1	d2
a1	c2	b1	d3
a1	c2	b2	d4
			…

Access 6

v_1	w
a1	8
a2	4

Access 1

v_1	v_3	w
a1	c1	1
a1	c2	1
a2	c2	1

Access 2

v_2	w
b1	3
b2	1

v_2	v_4	w
b1	d1	1
b1	d2	1
b1	d3	1
b2	d4	1

Algorithm

[**C**, Zeevi, Berkholz, Kimelfeld, Schweikardt; PODS 20]

- Preprocessing:
  - DP up the tree
  - computes how many answers in a subtree use each tuple
- Access:
  - recurse down the tree
  - splits the desired index between the children

v_1	w
a1	8
a2	4

v_2	w
b1	3
b2	1

v_1	v_3	w
a1	c1	1
a1	c2	1
a2	c2	1

v_2	v_4	w
b1	d1	1
b1	d2	1
b1	d3	1
b2	d4	1

Orders the algorithm can achieve:
DFS of a join tree

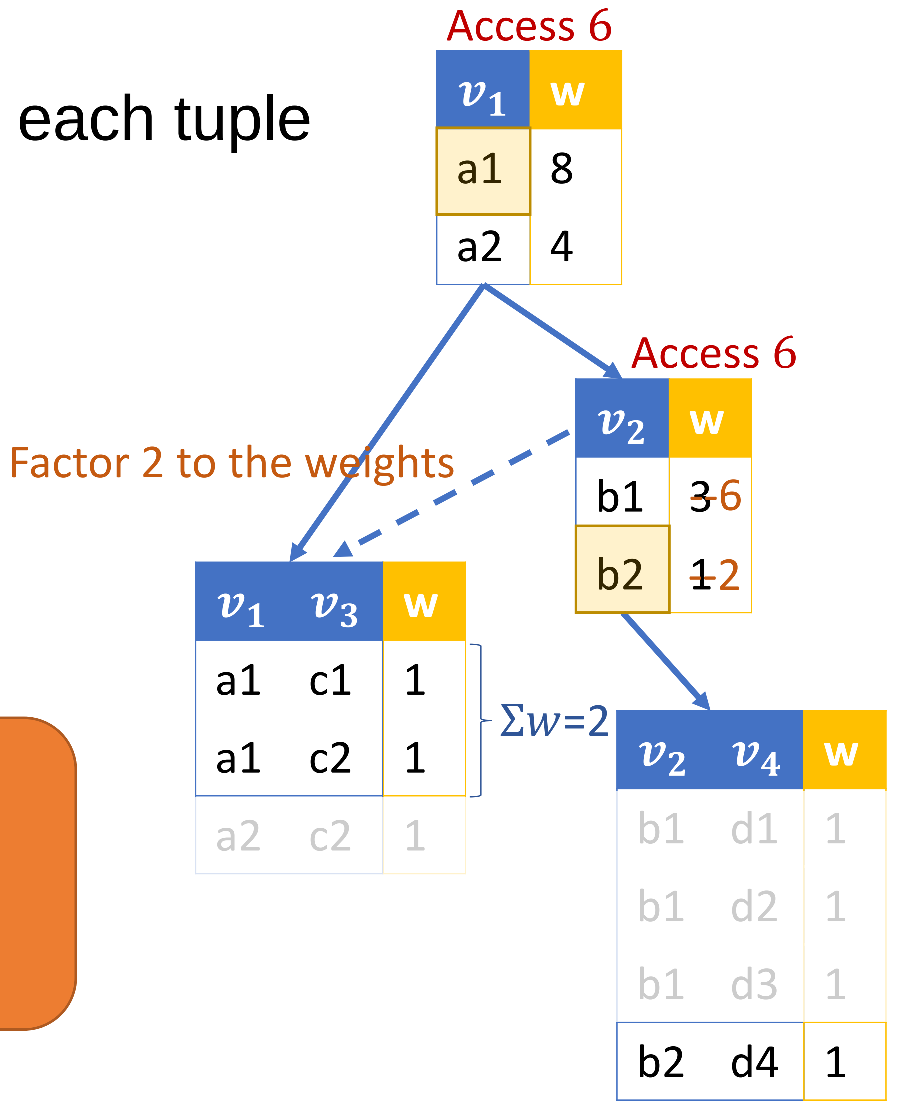
Example

$$Q_2(v_1, v_2, v_3, v_4) \leftarrow R(v_1, v_3), S(v_2, v_4)$$

- No disruptive trio
- Not a DFS of a join tree
- Can it be solved with ideal guarantees?
- Yes!

Algorithm

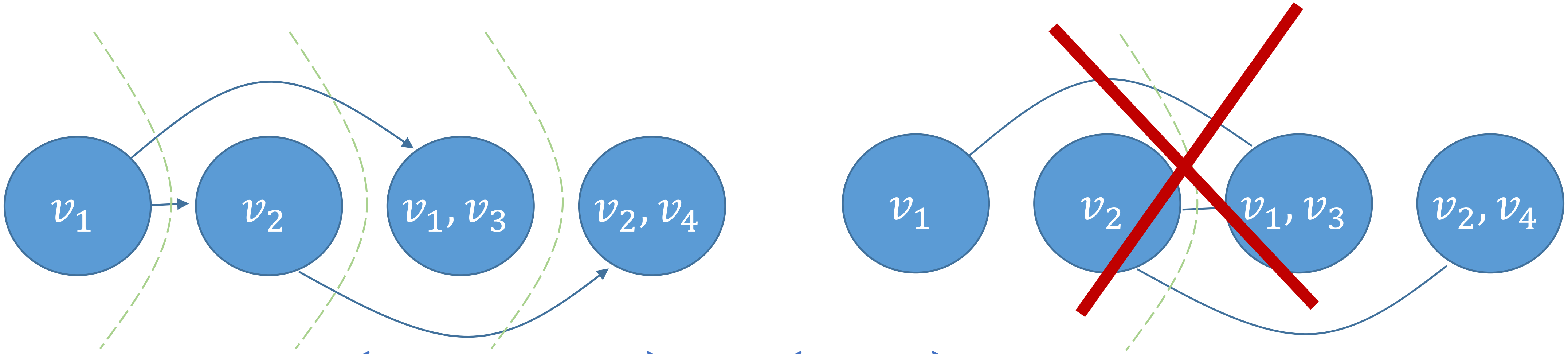
- Preprocessing:
  - DP up the tree
  - computes how many answers in a subtree use each tuple
- Access [PODS 20]:
  - recurse down the tree
  - splits the desired index between the children
- Modified Access [PODS 21]:
  - Move children on the fly



Orders the algorithm can achieve:
Orders matching a layered join tree

Layered Trees

- Layered tree for a **CQ** and a variable **ordering**:
  - Join-tree for an inclusive extension
  - Layer i = one node with last variable v_i
  - The induced graph by the first k layers is a tree, for all k



$$Q_2(v_1, v_2, v_3, v_4) \leftarrow R(v_1, v_3), S(v_2, v_4)$$

$\exists$ Layered join tree $\Leftrightarrow$ $\neg\exists$ disruptive trio

What is the best achievable performance?

<u>Ideal time guarantees:</u>

Preprocessing: linear (to read the input)

Access: constant

Allow log factors, data complexity

- Can we always achieve that?

  No (never for cyclic joins)
- Can we always achieve that if the query is acyclic?

  No (never if there is a disruptive trio)
- Can we always achieve that if the query is acyclic and without disruptive trios?

  Yes!

Dichotomy Result

[**C**, Tziavelis , Gatterbauer, Kimelfeld, Riedewald; PODS 21]

Given: join query Q, ordering L of free(Q),

lexicographic access in <n, $\log n$>

⇳*

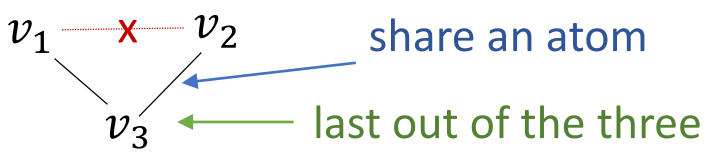
acyclic, no disruptive trio

* Lower bounds assume:

(1) no self-joins

(2) hardness of matrix multiplication and hyperclique detection

Disruptive Trio



Content

- Task
- Dichotomy for ideal time complexity:
  - Hardness
  - Algorithm
- **Solutions for hard cases:**
  - Functional dependencies
  - Selection problem
  - Extended preprocessing
- Concluding remarks

Talk focus:
Join queries, lex orders

What do we do in the hard cases?

- Can we use **dependencies** in the schema?

Unary Functional Dependencies

- Sometimes there are equivalent tractable (query, order) pairs

- Generic reduction [Carmeli, Kröll; ICDT 18]:

$$Q_1(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2) \qquad S: v_3 \rightarrow v_2$$

$\Downarrow$ linear construction

$$Q_1'(v_1, v_2, v_3) \leftarrow R'(v_1, v_3, v_2), S(v_3, v_2)$$

- Task-specific reduction:

$$Q_1(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2) \qquad R: v_1 \rightarrow v_3$$

$$\Downarrow$$

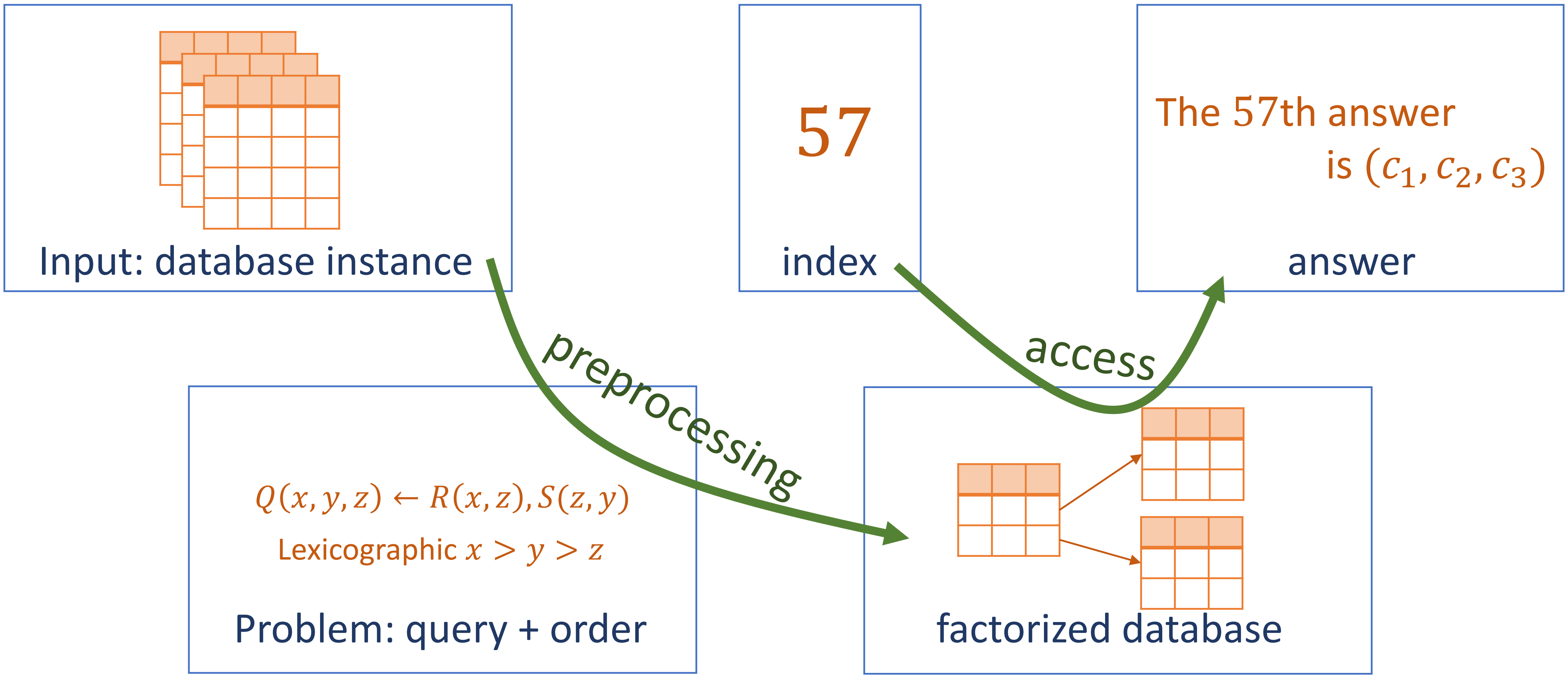
$$Q_1''(v_1, v_3, v_2) \leftarrow R(v_1, v_3), S(v_3, v_2)$$

- Dichotomy result: consider an FD-reordered extension

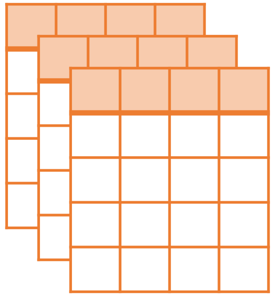
What do we do in the hard cases?

- Can we use **dependencies** in the schema?
  Yes, in some cases.
- Can we do it with linear **access** time?

Direct Access Problem



Selection Problem (supports a single access call)



Input: database instance

57

index

The 57th answer is (c_1, c_2, c_3)

answer

access

$Q(x,y,z) \leftarrow R(x,z), S(z,y)$

Lexicographic $x > y > z$

Problem: query + order

factorized database

Selection

- Support single access (instead of multiple calls)
- Allow linear time for an access call
- Additional queries become tractable.
  Example: $Q_1(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$
- Every acyclic join and lex order have linear time selection!
- Linear time access $\Leftrightarrow$ acyclic query (regardless of lex order)

What do we do in the hard cases?

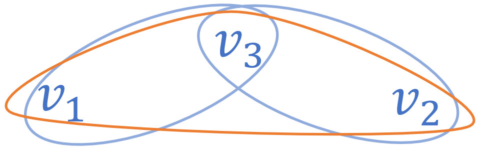
- Can we use **dependencies** in the schema?
  Yes, in some cases.
- Can we do it with linear **access** time?
  Yes, for all acyclic joins (regardless of the lex order).
- How much do we need to pay in **preprocessing** to get log access time?

Extended Preprocessing: Algorithm

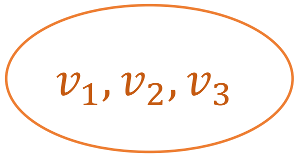
- Join problematic relations at preprocessing
- What to join?
  Disruption-free decomposition:
  For variables in reverse order: bag with variable and smaller neighbors

Examples:

- $Q_1(v_1, v_2, v_3) \leftarrow R_1(v_1, v_3), R_2(v_3, v_2)$
  Disruptive trios: (v_1, v_3, v_2)



query hypergraph

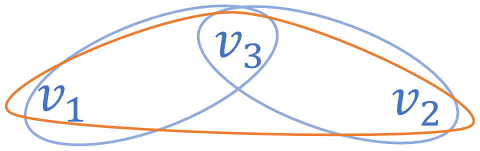


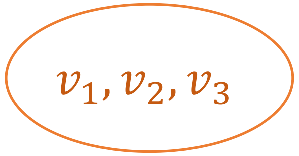
decomposition

Extended Preprocessing: Algorithm

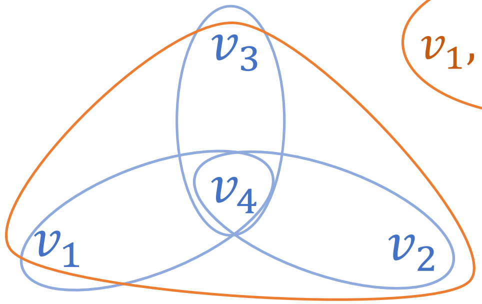
Examples:

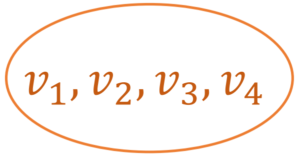
- $Q_1(v_1, v_2, v_3) \leftarrow R_1(v_1, v_3), R_2(v_3, v_2)$
  Disruptive trios: (v_1, v_3, v_2)



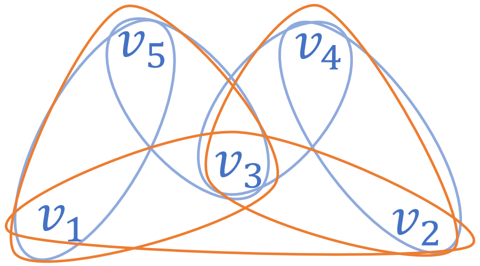


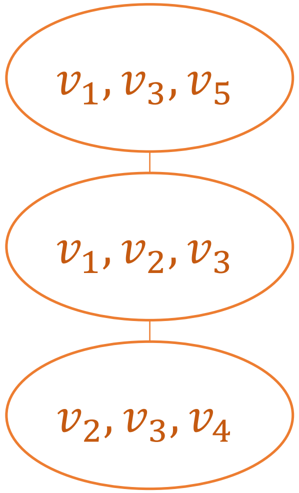
- $Q_1(v_1, v_2, v_3, v_4) \leftarrow R_1(v_1, v_4), R_2(v_4, v_2), R_3(v_4, v_3)$
  Disruptive trios: $(v_1, v_4, v_3), (v_1, v_4, v_2), (v_2, v_4, v_3)$





- $Q_1(v_1, v_2, v_3, v_4, v_5) \leftarrow R_1(v_1, v_5), R_2(v_5, v_3), R_3(v_3, v_4), R_4(v_4, v_2)$
  Disruptive trios: $(v_1, v_5, v_3), (v_3, v_4, v_2)$





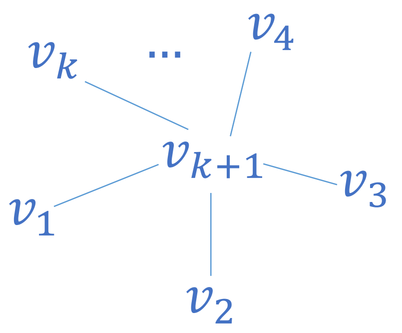
Disruption-free decomposition:
For variables in reverse order: bag with variable and smaller neighbors

Extended Preprocessing: Hardness

- Cost: n^{ι}
  ι = incompatibility number = largest fractional edge cover of a bag
- Can we do better?
  - No better decomposition exists
  - Other techniques? probably not (due to conditional lower bound)
- Reductions:
  access to query $\Rightarrow$ access to star query $\Rightarrow$ testing for projected star
  $\Rightarrow$ online set-disjointness $\Rightarrow$ Zero-clique

ι = fractional edge cover of a bag
= fractional independent set of the same bag

star query:



- Zero-clique conjecture:
  $\forall k, \varepsilon$: no randomized algorithm to detect a k-clique with 0 weight in a weighted graph in $O(n^{k-\varepsilon})$

Preprocessing Result

[Bringmann, **C**, Mengel; PODS 22]

Given: join query Q, ordering L of free(Q),

lexicographic access in <$n^k, \log n$>

$\Updownarrow$*

$\iota(Q, L) \leq k$

* Lower bounds assume:
  (1) no self-joins
  (2) hardness of zero-clique detection

Content

- Task
- Dichotomy for ideal time complexity:
  - Hardness
  - Algorithm
- Solutions for hard cases:
  - Functional dependencies
  - Selection problem
  - Extended preprocessing
- **Concluding remarks**

Talk focus:
Join queries, lex orders

Connections to known notions

- Elimination order
  - No disruptive trio ⇔ reverse elimination order [Brault-Baron 13]

- d-trees
  - Translation d-tree to tree decompositions [Olteanu, Závodný; TODS 15] gives a layered join tree of an order matching the tree
  - The access algorithm can be applied directly on d-trees

Conclusion

- More in our papers:
  - Joins with projection
  - Partial lexicographic orders
  - Sum of weights order

- Future work:
  - Preprocessing needed for the above extensions
  - More expressive query classes
  - Preprocessing-access tradeoff

Extra Slides

Introduction

Related work

- Enumeration [BaganDurandGrandjean CSL'07] [Brault-Baron thesis 2013]
  const (or log) delay possible $\Leftrightarrow^*$ free-connex

- Ranked enumeration [TziavelisAjwaniGatterbauerRiedewaldYang PVLDB'20]
  sum of weights (or lexicographic), log delay, free-connex

- Direct access (underlying restricted order support)
  - via elimination order [Brault-Baron thesis 2013]
  - via join tree [**C**ZeeviBerkholzKimelfeldSchweikardt PODS'20]
  - via q-tree (dynamic settings, q-hierarchical only) [Keppeler thesis 2020]

All using: data complexity, RAM model

Definitions

An acyclic CQ has a graph with:

A free-connex CQ also requires:

1. a node for every atom
possibly also subsets

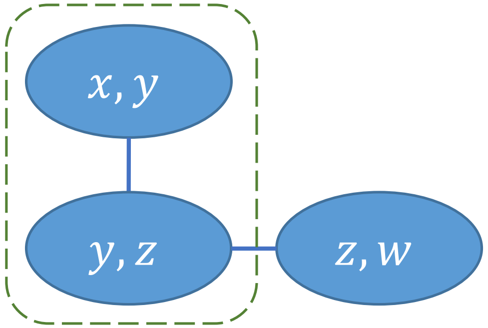
2. tree

3. for every variable X:
the nodes containing X form a subtree

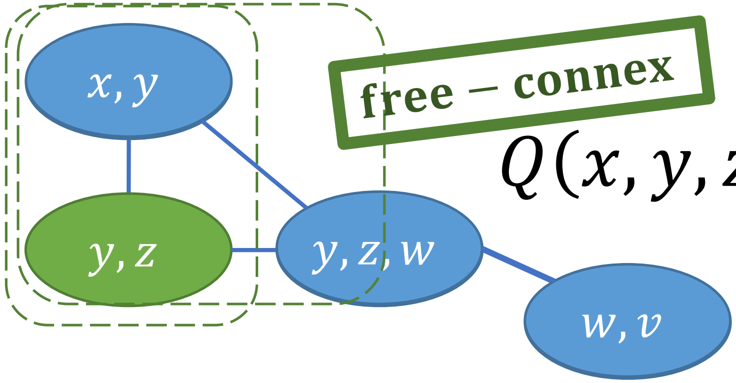
free – connex

acyclic

$$Q(x, y, z) \leftarrow R_1(x, y), R_2(y, z), R_3(z, w)$$



4. a subtree with exactly the free variables



free – connex

$$Q(x, y, z) \leftarrow R_1(x, y), R_2(y, z, w), R_3(w, v)$$

Free-Connex CQs

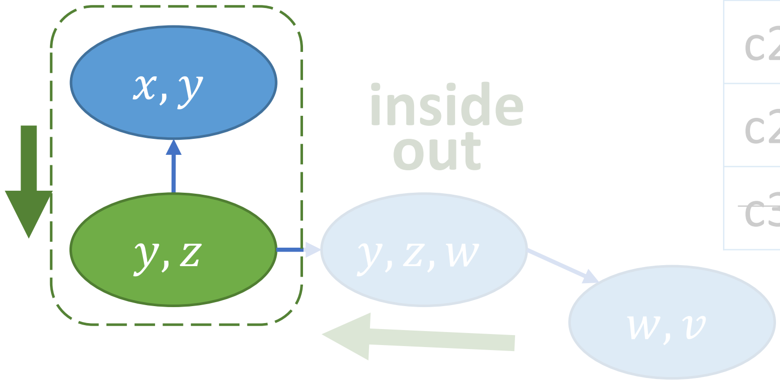
$$Q(x, y, z) \leftarrow R_1(x, y), R_2(y, z, w), R_3(w, v)$$

Can be reduced to full acyclic

1. Find a join tree
2. Remove dangling tuples [Yannakakis81]
3. Ignore existential variables

Then, joined efficiently

x	y
~~a1~~	~~b1~~
a1	b2
a2	b2



w	v
c2	d1
c2	d2
~~c3~~	~~d2~~

y	z
~~b1~~	~~e1~~
b2	e2
~~b3~~	~~e3~~

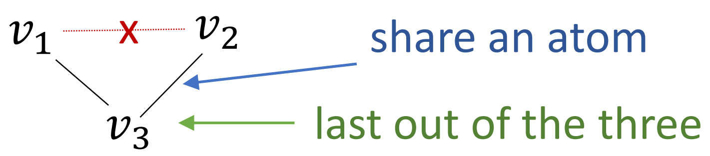
y	z	w
~~b1~~	~~e1~~	~~c1~~
b2	e2	c2
~~b3~~	~~e3~~	~~c3~~

Lexicographic Orders

Hardness Result

- Can be extended whenever there is a disruptive trio

Def: disruptive trio



Example: $Q_1(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$

- Proof idea:

Using binary search: truncate the lex order to make v_3 existential and v_1, v_2 free
Known for acyclic CQs: not free-connex ⇔ exists free-path
(chordless path, endpoints free, middle existential)
The obtained CQ has a free-path $v_1 - v_3 - v_2$

Dichotomy

✓ $Q_1(v_1, v_2, v_3) \leftarrow R(v_1, v_2), S(v_2, v_3)$

✗ $Q_2(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$

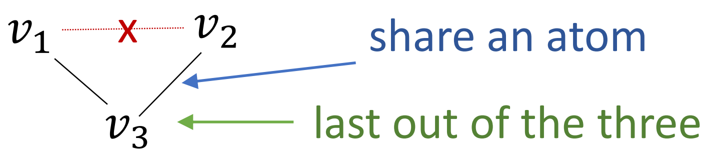
Given: CQ Q, ordering L of free(Q),

lexicographic access in $<n \operatorname{polylog} n, \operatorname{polylog} n>$

$\Updownarrow$*

free-connex, no disruptive trio

<u>Disruptive Trio</u>



* Lower bounds assume:

(1) no self-joins

(2) hardness of matrix multiplication and hyperclique detection

Partial Lexicographical Ordering

- possible ⇔ a completion for a feasible full ordering

a subset of

Given: CQ Q, ordering L of free(Q),

partial

lexicographic access in $<n \text{ polylog } n, \text{polylog } n>$

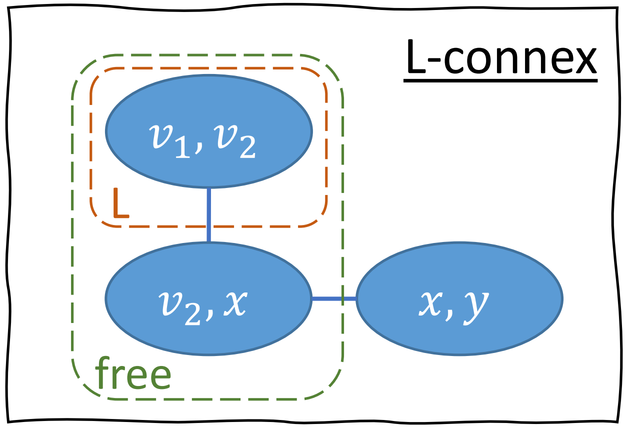
⇕*

L-connex,

free-connex, no disruptive trio

$$Q_1(v_1, v_2, x) \leftarrow R(v_1, v_2), S(v_2, x), T(x, y) \checkmark$$

$$Q_2(v_1, v_2, x) \leftarrow R(v_1, x), S(x, v_2) \text{ ✗}$$



* Lower bounds assume:

(1) no self-joins

(2) hardness of matrix multiplication and hyperclique detection

Sum of Weights

Dichotomy

Given: CQ Q,

sum-of-weights access in $<n \operatorname{polylog} n, \operatorname{polylog} n>$

$\Updownarrow$*

acyclic, an atom contains all free variables

Tractability is trivial

$Q_1(x, z) \leftarrow R(x, y, z), S(y, z)$ ✓

$Q_2(x, z) \leftarrow R(x, y), S(y, z)$ ✗

* Lower bounds assume:

(1) no self-joins

(2) hardness of 3-SUM and hyperclique detection

Hardness

- Observation: Binary search finds a weight with logarithmic accesses

3SUM hypothesis
given 3 sets of integers $|A| = |B| = |C| = n$,
deciding $\exists\ a \in A, b \in B, c \in C$ s.t. $a + b + c = 0$
cannot be done in time $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$

- Use two independent free variables

$$Q_2(x, z) \leftarrow R(x, y), S(y, z)$$

Direct access impossible in $<n^{2-\varepsilon}, n^{1-\varepsilon}>$

x	y
a_1	0
a_2	0

A

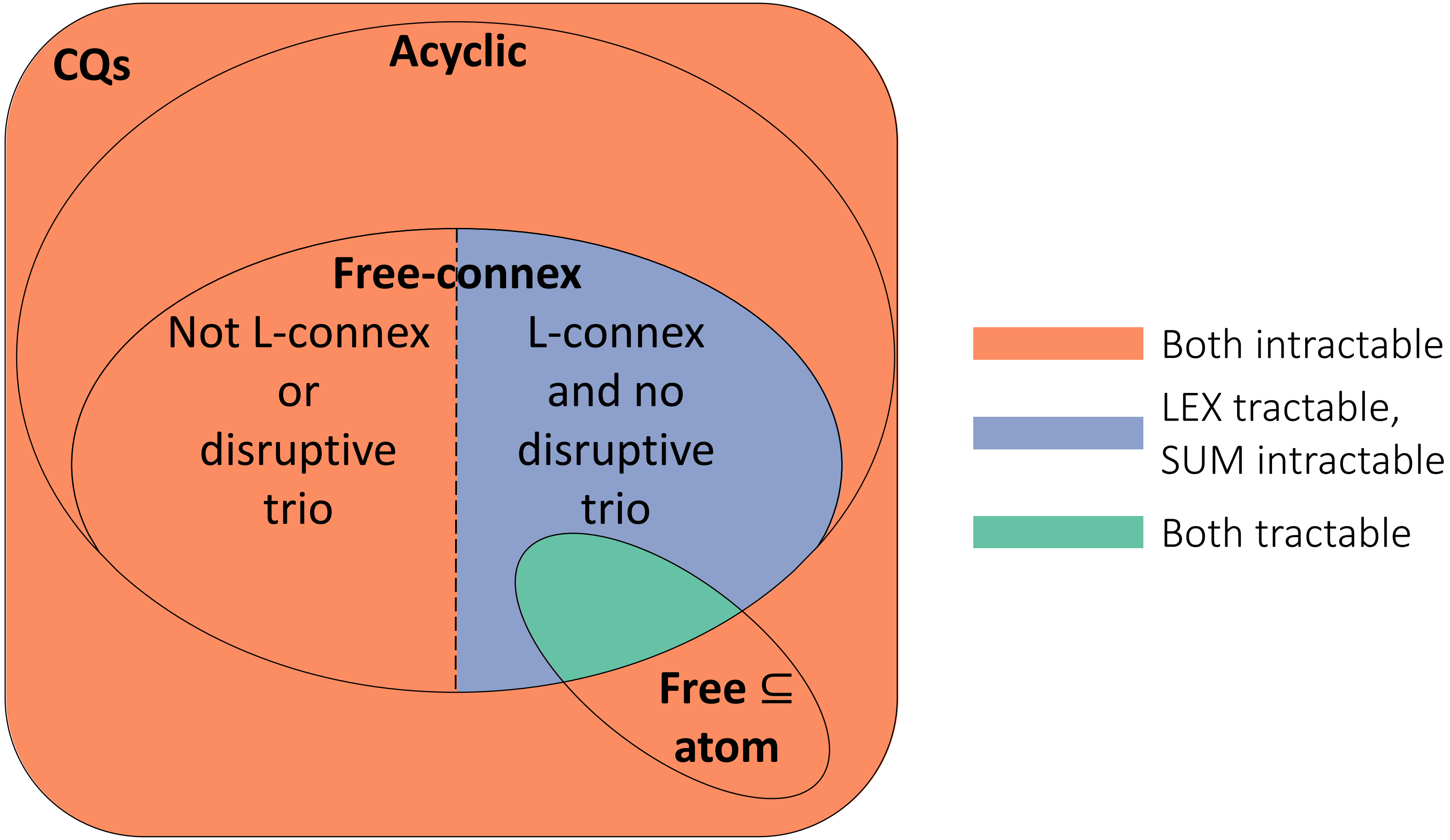
y	z
0	b_1
0	b_2

B

x	y	z	w
a_1	0	b_1	$a_1 + b_1$
a_1	0	b_2	$a_1 + b_2$
a_2	0	b_1	$a_2 + b_1$
a_2	0	b_2	$a_2 + b_2$

Binary search for $-c$ $(\forall c)$

Direct-Access Overview



* Lower bounds assume:

(1) no self-joins

(2) hardness of 3-SUM, Boolean matrix multiplication, and hyperclique detection

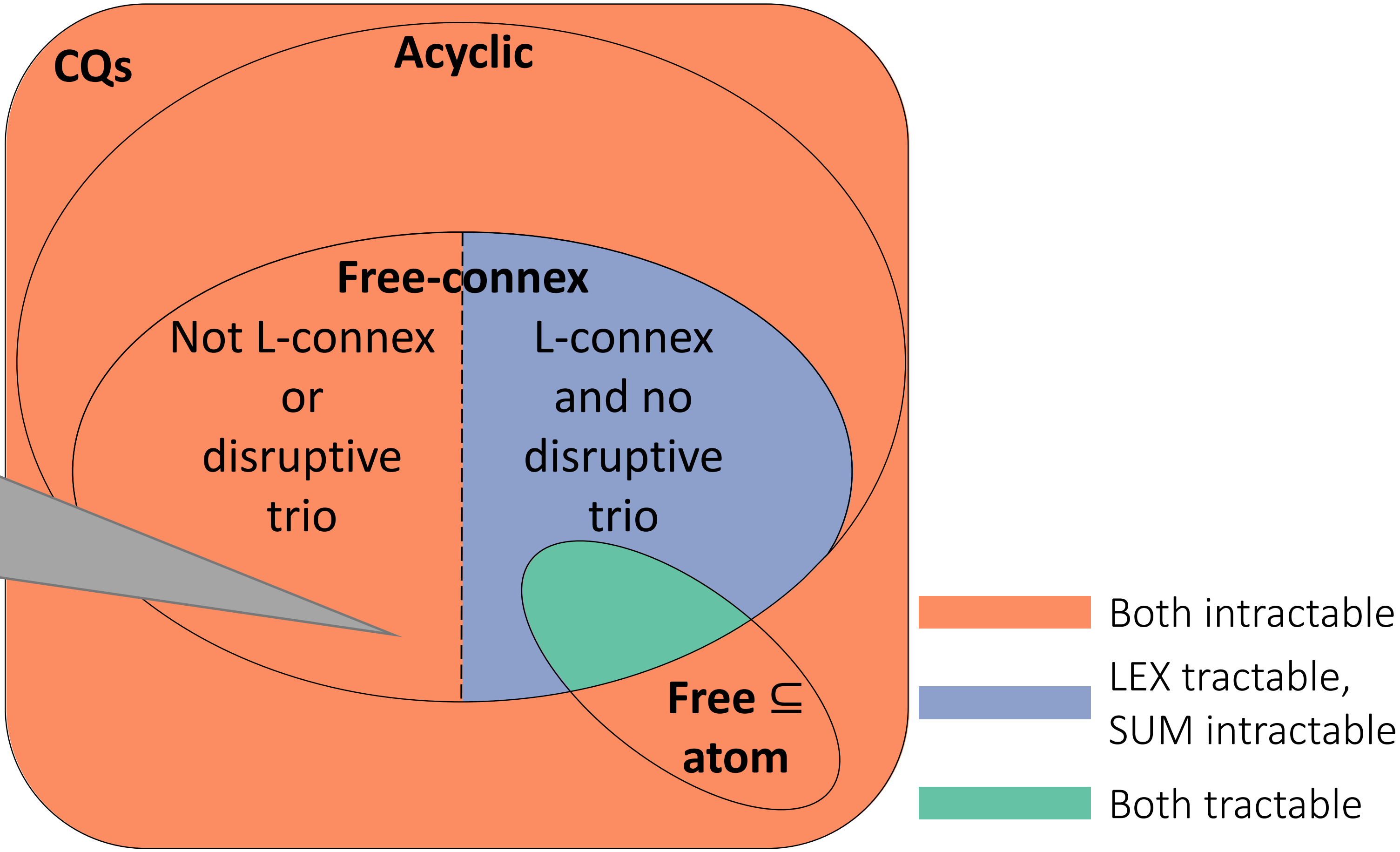
Selection

Direct-Access Overview

Example:
$Q(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$

✗ Direct access
✓ Selection

(for both order types)



* Lower bounds assume:
(1) no self-joins
(2) hardness of 3-SUM, Boolean matrix multiplication, and hyperclique detection

Selection Dichotomy

Given: **full** CQ Q,

sum-of-weights selection in $O(nlogn)$

$\Updownarrow$* w.r.t. hyperedge containment

At most two maximal atoms

$Q_1(x,y,z) \leftarrow R(x,y), S(y,z), T(y)$ ✓

$Q_2(x,y,z,u) \leftarrow R(x,y), S(y,z), T(z,u)$ ✗

* Lower bounds assume:
  (1) no self-joins
  (2) hardness of 3-SUM and hyperclique detection

Selection

- Sometimes: efficient selection, no efficient direct access

$$Q_2(v_1, v_2, v_3) \leftarrow R(v_1, v_3), S(v_3, v_2)$$

v_1 \ v_2	f
a	111
b	211

$v_3 =$ d

v_1 \ v_2	f	g
a	112	122
c	312	322

$v_3 =$ e

v_1	v_3	w
a	d	101
b	d	201
a	e	102
c	e	302

v_3	v_2	w
d	f	10
e	f	10
e	g	20

[Frederickson Johnson 1984]

Selection on a union of sorted matrices
of dimensions $m_i \times n_i$
possible in time $O(\sum \max(m_i, n_i))$

* We do not materialize the matrices

Hardness

- Assumption: Q_1 cannot be decided in quasilinear time
- Reduction:
  - Use the same relations
  - Need to identify answers to Q_2 with $x = w$
  - Can identify answers with weight 0 (binary search)

y weights
b: 0
l: 0

z weights
c: 0
m: 0

x weights
a: 1
k: 2

w weights
a: -1
n: -3

Decide

$$Q_1() \leftarrow R(x,y), S(y,z), T(z,x)$$

using

Sum-of-weights selection

$$Q_2(x,y,z,w) \leftarrow R(x,y), S(y,z), T(z,w)$$

Q_2 answers:

x	y	z	w	weight
a	b	c	a	0
k	l	m	n	-1

x	y
a	b
k	l

y	z
b	c
l	m

z	w
c	a
m	n

- Log number of selection calls
  - Q_1 selection with quasilinear time, contradiction

Extended Preprocessing

Preprocessing Result

[Bringmann, **C**, Mengel; PODS 22]

Given: join query Q, ordering L of free(Q),

1. lexicographic access in $<n^{\iota(Q,L)}, \log n>$
2. $\forall \varepsilon$, no lex access in $<n^{\iota(Q,L)-\varepsilon}, \text{polylog } n>$, assuming hardness of zero-clique detection