Technion

Humboldt-Universität
Informatik

Answering (Unions of) Join Queries using Random Access and Random-Order Enumeration

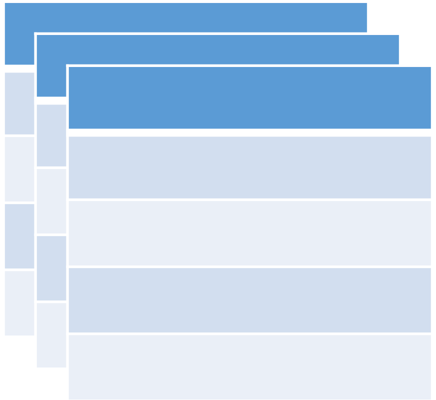
Nofar Carmeli

Joint work with Christoph Berkholz, Benny Kimelfeld,
Nicole Schweikardt, and Shai Zeevi

Tasks & Motivation

Conjunctive Queries
Unions of Conjunctive Queries

Why Random Permutation?



Database

+

🔎

Query

---

very large

<u>**Enumeration:**</u> 🍎 🍒 🍌 🌽 🍆 🍇

Downside: intermediate results not representative

<u>**Sampling:**</u> 🍌 🍆 🍒 🍒 🍌 🌽

Downside: repeating answers

<u>**Random Permutation:**</u> 🍎 🍆 🍌 🍒 🍇 🌽

Each answer once, uniformly random order

Idea: Separate the Task

- Find the number N of answers

6

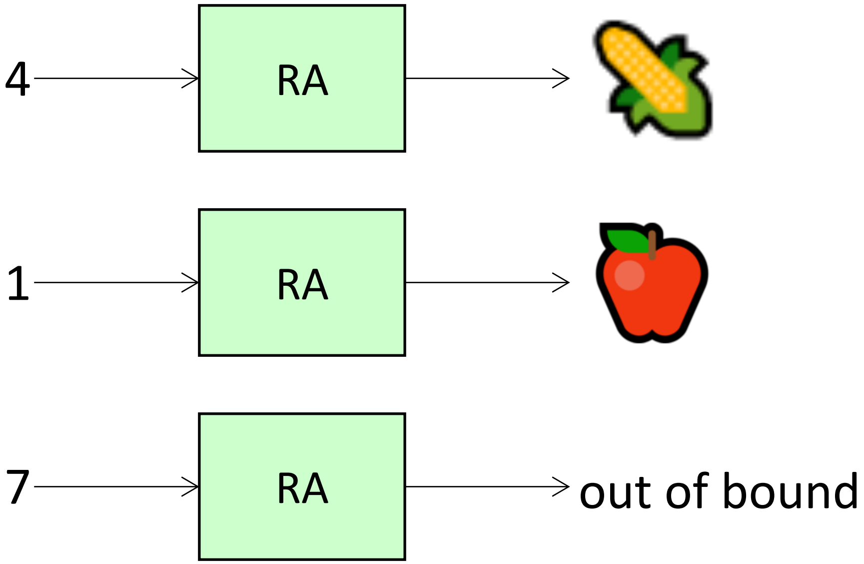
- Find a random permutation of 1,…,N

1 5 3 2 6 4

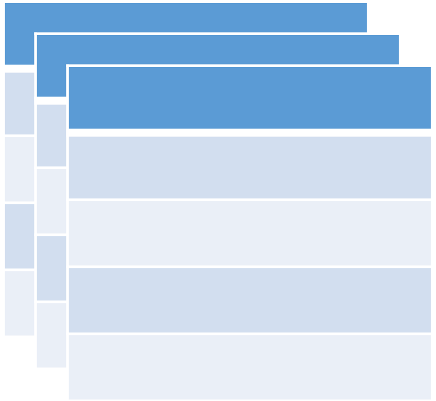
- Random access to answers

Random Access

- Simulates precomputed results stored in an array
- Given i, returns the i^{th} answer or “out of bound”
- No constraints on the ordering used



Consider 3 Tasks



Database

+

🔍

Query

Enumeration: 🍎 🍒 🍌 🌽 🍆 🍇

Random Permutation: 🍎 🍆 🍌 🍒 🍇 🌽

Random Access: 4 ⇨ 🌽

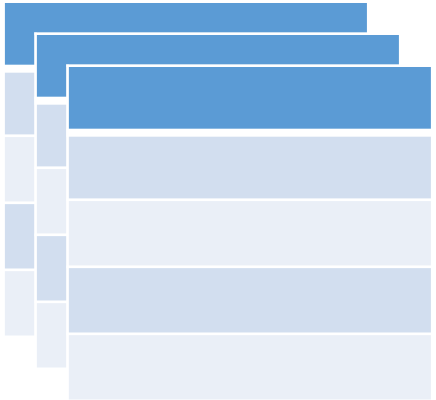
Complexity of Query Evaluation

- Treat every query as a problem
- Consider time complexity
- Data complexity
  - Input: DB instance
  - Query size: constant
- RAM model [Grandjean1996]
  - Lookup table: construction in linear time
    search in constant time

When can we solve the tasks efficiently?

(linear preprocessing + polylog per answer)

Consider 3 Tasks



Database

+

🔍

Query

Enumeration: 🍎 🍒 🍌 🌽 🍆 🍇

⇑

Random Permutation: 🍎 🍆 🍌 🍒 🍇 🌽

⇑

Random Access: 4 ⇨ 🌽

Random Access ⇒ Random Permutation

- Find the number N of answers

6

- Find a random permutation of 1,…,N

1 5 3 2 6 4

- Random access to answers

🍎 🍆 🍌 🍒 🍇 🌽

Counting via RandomAccess

- Assumption: the number of answers is bound by a polynomial
- RandomAccess returns “out of bound” if needed
  - Allows checking if $|answers| \geq k$ in polylog time
- Binary search for $|answers|$
  - Requires $O(\log(|answers|))$ calls for RandomAccess
  - If $|answers|$ is polynomial, $\log(|answers|) = O(\log(input))$
  - This takes polylog time

Random Access ⇒ Random Permutation

- Find the number N of answers

6

- Find a random permutation of 1,…,N

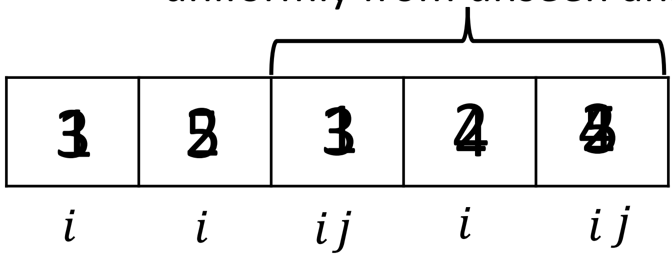
1 5 3 2 6 4

- Random access to answers

Generating a Random Permutation

- Use the Fisher-Yates Shuffle [Durstenfeld 1964]

```
place $1, \ldots, n$ in array
for $i$ in $1, \ldots, n$:
    choose j randomly from $\{i, \ldots, n\}$
    swap $i$ and $j$
```



next answer: chosen uniformly from unseen answers

Generating a Random Permutation

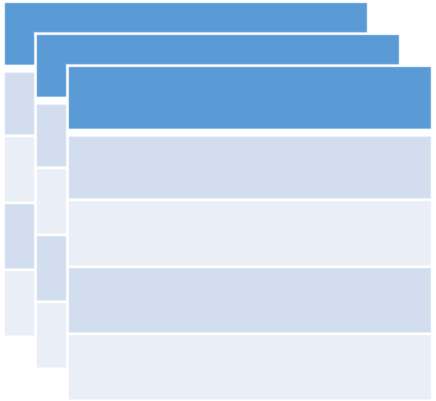
- Use the Fisher-Yates Shuffle [Durstenfeld 1964]

Constant delay variant:

```
place $1, \dots, n$ in array (lazy initialization)
for $i$ in $1, \dots, n$:
    choose j randomly from $\{i, \dots, n\}$
    swap $i$ and $j$
    print $a[i]$
```

3	5	1	2	4

Consider 3 Tasks

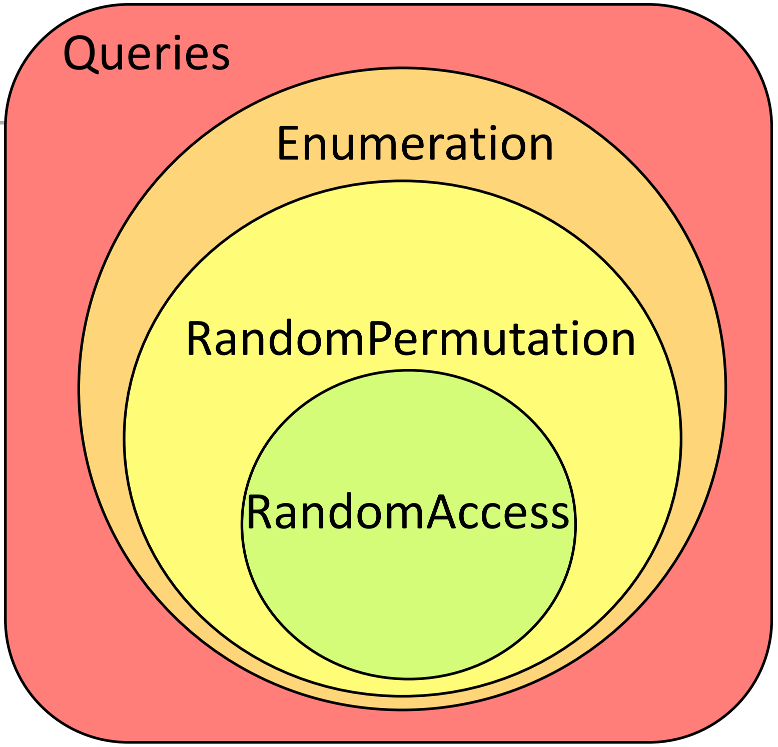


Database

+

🔍

Query



Enumeration: 🍎 🍒 🍌 🌽 🍆 🍇

Random Permutation: 🍎 🍆 🍌 🍒 🍇 🌽

Random Access: 4 ⇨ 🌽

Tasks & Motivation

Conjunctive Queries

Unions of Conjunctive Queries

CQs Dichotomy

After linear preprocessing

	Enumeration $O(1)$ delay	Random Permutation $O(\log n)$ delay	Random Access $O(\log n)$	
Acyclic Free-Connex	✓	✓	✓	Also efficient counting, membership testing, etc.
Acyclic Not Free-Connex	✗	✗	✗	Assuming the hardness of Boolean matrix multiplication.
Cyclic	✗	✗	✗	Cannot find any answer in $O(n)$ time, assuming the hardness of finding hypercliques.

The lower bounds assume
no self-joins

Definitions

An acyclic CQ has a graph with:

A free-connex CQ also requires:

1. a node for every atom
possibly also subsets

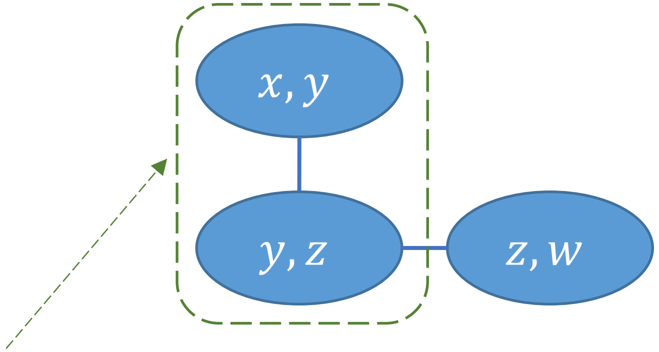
2. tree

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

free – connex

acyclic

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



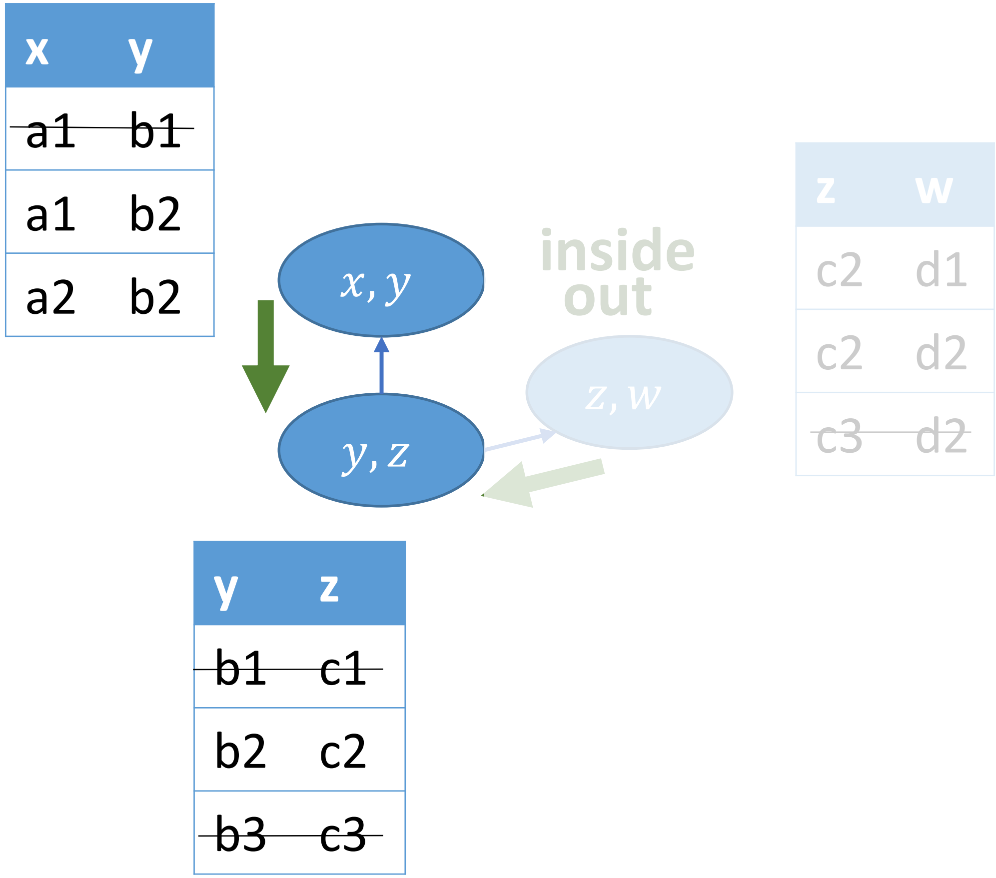
4. a subtree with exactly the free variables

Free-Connex CQs

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

Can be answered efficiently

1. Find a join tree
2. Remove dangling tuples [Yannakakis81]
3. Ignore existential variables
4. Full Acyclic:
   Do what you want



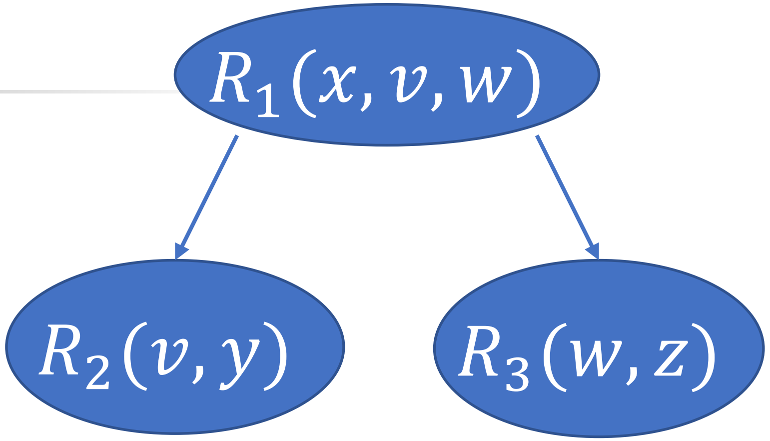
x	y
~~a1~~	~~b1~~
a1	b2
a2	b2

y	z
~~b1~~	~~c1~~
b2	c2
~~b3~~	~~c3~~

z	w
c2	d1
c2	d2
~~c3~~	~~d2~~

Enumeration [BaganDurandGrandjean 2007], Counting [DurandMengel 2011], Random Access [Brault-Baron 2013]

Random Access Algorithm



$R_1(x, v, w)$

$R_2(v, y)$

$R_3(w, z)$

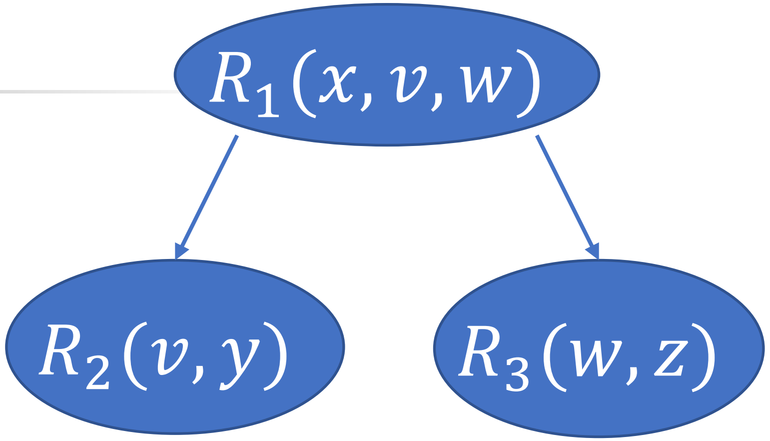
Preprocessing:

- Full reduction
- Bucketing
- Weighting (DP)

Example:

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

Random Access Algorithm



Preprocessing:

- **Full reduction**
- Bucketing
- Weighting (DP)

Example:

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

R_2
$v_1\ y_1$
$v_1\ y_2$
$v_2\ y_2$
$v_2\ y_3$

R_1
$x_1\ v_1\ w_1$
$x_1\ v_1\ w_2$
$x_2\ v_2\ w_1$
$x_2\ v_2\ w_2$
$x_1\ v_3\ w_1$

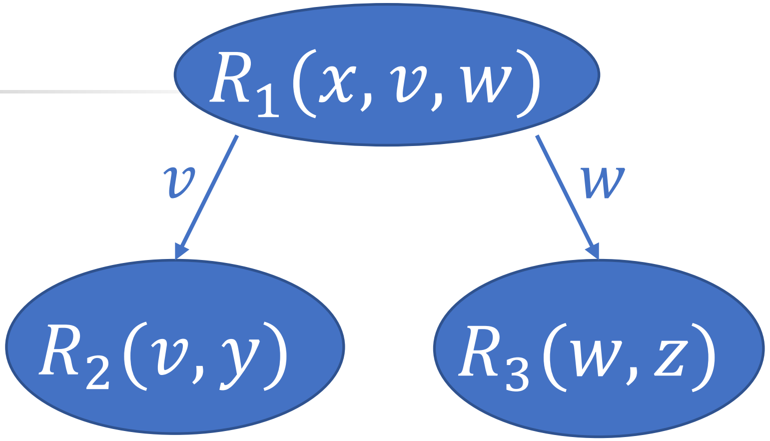
R_3
$w_1\ z_1$
$w_1\ z_2$
$w_1\ z_3$
$w_2\ z_4$
$w_3\ z_1$

R_2
$v_1\ y_1$
$v_1\ y_2$
$v_2\ y_2$
$v_2\ y_3$

R_1
$x_1\ v_1\ w_1$
$x_1\ v_1\ w_2$
$x_2\ v_2\ w_1$
$x_2\ v_2\ w_2$

R_3
$w_1\ z_1$
$w_1\ z_2$
$w_1\ z_3$
$w_2\ z_4$

Random Access Algorithm



Preprocessing:

- Full reduction
- **Bucketing**
- Weighting (DP)

Example:

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

R_2
$v_1\ y_1$
$v_1\ y_2$
$v_2\ y_2$
$v_2\ y_3$

R_1
$x_1\ v_1\ w_1$
$x_1\ v_1\ w_2$
$x_2\ v_2\ w_1$
$x_2\ v_2\ w_2$

R_3
$w_1\ z_1$
$w_1\ z_2$
$w_1\ z_3$
$w_2\ z_4$

R_2
$v_1\ y_1$
$v_1\ y_2$
$v_2\ y_2$
$v_2\ y_3$

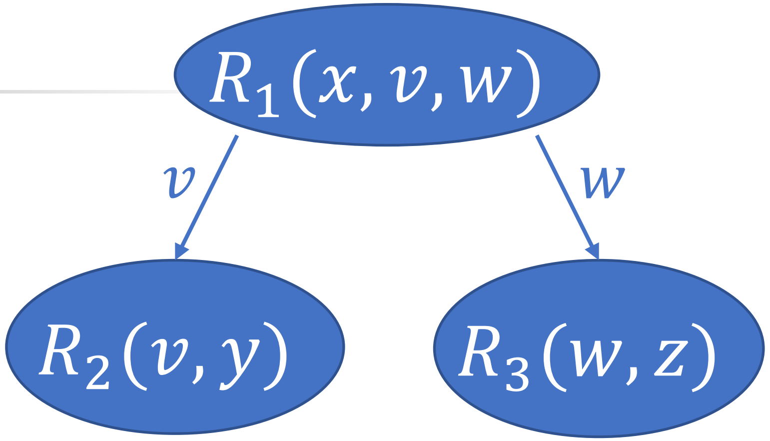
R_1
$x_1\ v_1\ w_1$
$x_1\ v_1\ w_2$
$x_2\ v_2\ w_1$
$x_2\ v_2\ w_2$

R_3
$w_1\ z_1$
$w_1\ z_2$
$w_1\ z_3$
$w_2\ z_4$

Random Access Algorithm

Preprocessing:

- Full reduction
- Bucketing
- **Weighting (DP)**



Example:

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

w = number of answers in subtree using this tuple

s = cumulative sum of w within the bucket

R_2
$v_1\ y_1$
$v_1\ y_2$
$v_2\ y_2$
$v_2\ y_3$

R_1
$x_1\ v_1\ w_1$
$x_1\ v_1\ w_2$
$x_2\ v_2\ w_1$
$x_2\ v_2\ w_2$

R_3
$w_1\ z_1$
$w_1\ z_2$
$w_1\ z_3$
$w_2\ z_4$

R_2	w	s	W
$v_1\ y_1$	1	0	2
$v_1\ y_2$	1	1	
$v_2\ y_2$	1	0	2
$v_2\ y_3$	1	1	

R_1	w	s	W
$x_1\ v_1\ w_1$	6	0	16
$x_1\ v_1\ w_2$	2	6	
$x_2\ v_2\ w_1$	6	8	
$x_2\ v_2\ w_2$	2	14	

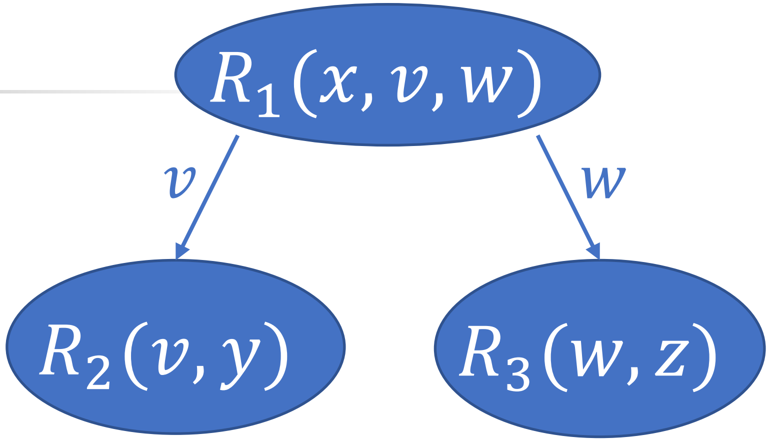
R_3	w	s	W
$w_1\ z_1$	1	0	3
$w_1\ z_2$	1	1	
$w_1\ z_3$	1	2	
$w_2\ z_4$	1	0	1

Random Access Algorithm

Preprocessing:

- Full reduction
- Bucketing
- **Weighting (DP)**

w = number of answers in subtree using this tuple
s = cumulative sum of w within the bucket



Example:

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

R_2
$v_1\ y_1$
$v_1\ y_2$
$v_2\ y_2$
$v_2\ y_3$

R_1
$x_1\ v_1\ w_1$
$x_1\ v_1\ w_2$
$x_2\ v_2\ w_1$
$x_2\ v_2\ w_2$

R_3
$w_1\ z_1$
$w_1\ z_2$
$w_1\ z_3$
$w_2\ z_4$

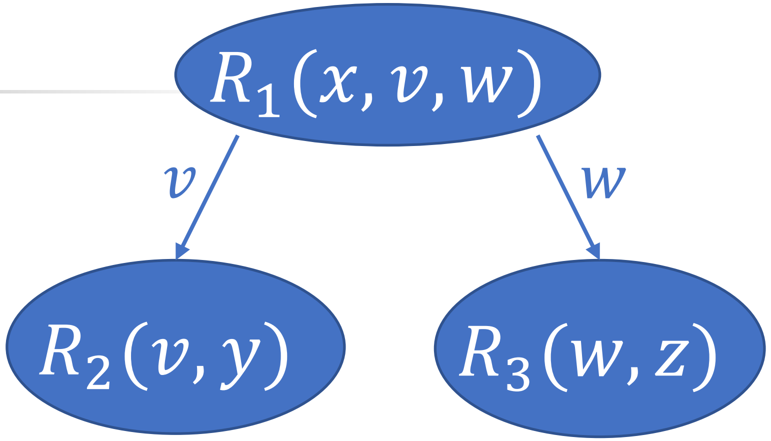
R_2	w	s	W
$v_1\ y_1$	1	0	2
$v_1\ y_2$	1	1	
$v_2\ y_2$	1	0	2
$v_2\ y_3$	1	1	

R_1	w	s	W
$x_1\ v_1\ w_1$	6	0	16
$x_1\ v_1\ w_2$	2	6	
$x_2\ v_2\ w_1$	6	8	
$x_2\ v_2\ w_2$	2	14	

$2 \times 3 = 6$

R_3	w	s	W
$w_1\ z_1$	1	0	3
$w_1\ z_2$	1	1	
$w_1\ z_3$	1	2	
$w_2\ z_4$	1	0	1

Random Access Algorithm



Example:

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

Access answer 11

$11 - 8 = 3$

Access index 3 of the answers with $(x_2\ v_2\ w_1)$ in the subtree

Split 3 like in a multidimensional array

$3 = 1 \times 3 + 0$

(🌱)R_1	w	s	W
$x_1\ v_1\ w_1$	6	0	16
$x_1\ v_1\ w_2$	2	6	
$x_2\ v_2\ w_1$	6	8	
$x_2\ v_2\ w_2$	2	14	

$8 \leq 11 < 14$

R_2	w	s	W
$v_1\ y_1$	1	0	2
$v_1\ y_2$	1	1	
$v_2\ y_2$	1	0	2
$v_2\ y_3$	1	1	

v_2 bucket
$s = 1$

R_3	w	s	W
$w_1\ z_1$	1	0	3
$w_1\ z_2$	1	1	
$w_1\ z_3$	1	2	
$w_2\ z_4$	1	0	1

w_1 bucket
$s = 0$

$$a_{11} = (x_2, v_2, w_1, y_3, z_1)$$

CQs Dichotomy

After linear preprocessing

	Enumeration $O(1)$ delay	Random Permutation $O(\log n)$ delay	Random Access $O(\log n)$	
Acyclic Free-Connex	✓	✓	✓	Also efficient counting, membership testing, etc.
Acyclic Not Free-Connex	✗	✗	✗	Assuming the hardness of Boolean matrix multiplication.
Cyclic	✗	✗	✗	Cannot find any answer in $O(n)$ time, assuming the hardness of finding hypercliques.

The lower bounds assume no self-joins

Random Access ⇒ Random Permutation

- Find the number N of answers

6

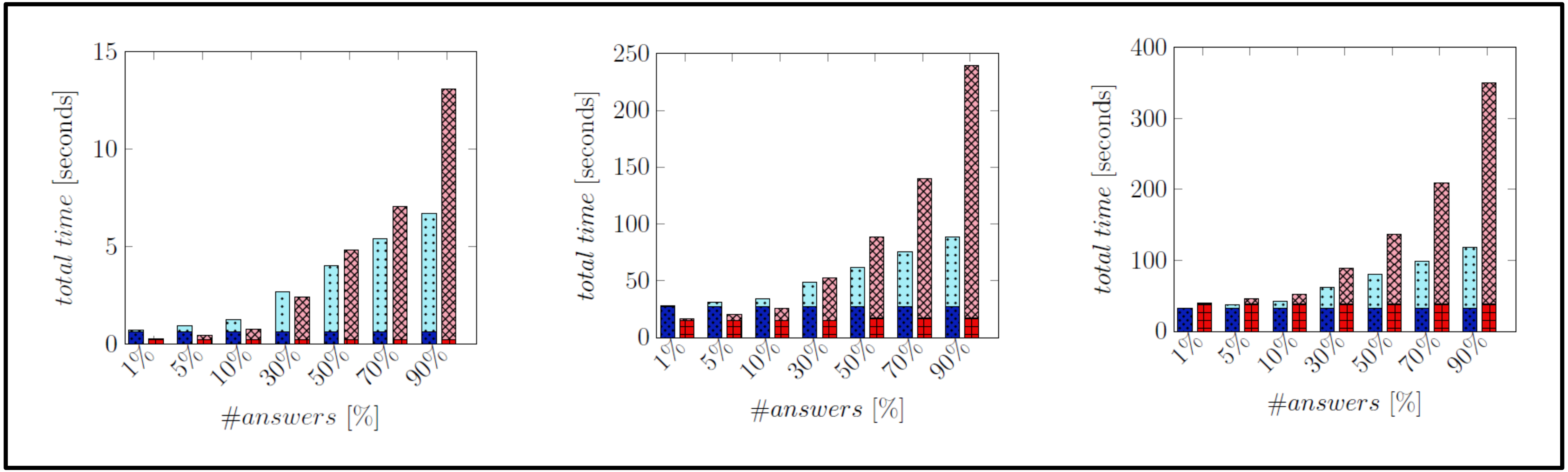
- Find a random permutation of 1,…,N

1 5 3 2 6 4

- Random access to answers

In Practice

- Compared to a sampling algorithm
  - [Zhao, Christensen, Li, Hu, and Yi SIGMOD 2018]
  - Modified to reject repeated answers
- TPC-H Queries



CQs Dichotomy

After linear preprocessing

	Enumeration $O(1)$ delay	Random Permutation $O(\log n)$ delay	Random Access $O(\log n)$
Acyclic Free-Connex	✓	✓	✓
Acyclic Not Free-Connex	✗	✗	✗
Cyclic	✗	✗	✗

Also efficient counting, membership testing, etc.

Assuming the hardness of Boolean matrix multiplication.

Cannot find any answer in $O(n)$ time, assuming the hardness of finding hypercliques.

The lower bounds assume no self-joins

Acyclic non-free-connex CQs [BaganDurandGrandjean CSL’2007]

Assumption: Boolean matrices cannot be multiplied in time $O(m^{1+o(1)})$

m = number of ones in the input and output

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

Indices of ones

R_1

R	C
1	1
1	2
2	2

R_2

R	C
1	2
2	2

Q

R	C
1	2
2	2

Acyclic non-free-connex: $Q(x, z) \leftarrow R_1(x, y), R_2(y, z)$

$O(m)$ preprocessing + $O(\log(m))$ delay = $O(m \log(m))$ total $\implies$ not possible

CQs Dichotomy

After linear preprocessing

	Enumeration $O(1)$ delay	Random Permutation $O(\log n)$ delay	Random Access $O(\log n)$
Acyclic Free-Connex	✓	✓	✓
Acyclic Not Free-Connex	✗	✗	✗
Cyclic	✗	✗	✗

Also efficient counting, membership testing, etc.

Assuming the hardness of Boolean matrix multiplication.

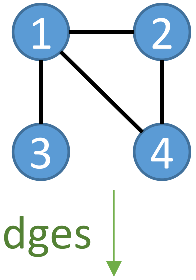
Cannot find any answer in $O(n)$ time, assuming the hardness of finding hypercliques.

The lower bounds assume no self-joins

Cyclic CQs

[Brault-Baron 2013]

Assumption: k-Hypercliques cannot be found in time $O(m)$
m = number of edges of size $k-1$



Q

x	y	z
1	2	4

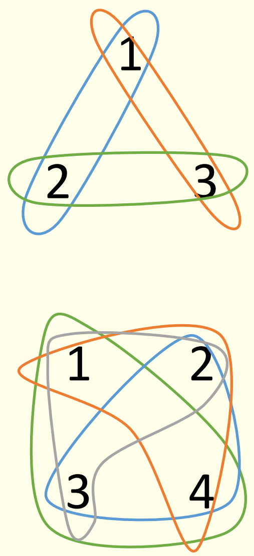
$R_1 = R_2 = R_3$

1	2
1	3
1	4
2	4

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

first answer in $O(m)$ time $\Longrightarrow$ not possible

k-Hypercliques



CQs Dichotomy

After linear preprocessing

	Enumeration $O(1)$ delay	Random Permutation $O(\log n)$ delay	Random Access $O(\log n)$
Acyclic Free-Connex	✓	✓	✓
Acyclic Not Free-Connex	✗	✗	✗
Cyclic	✗	✗	✗

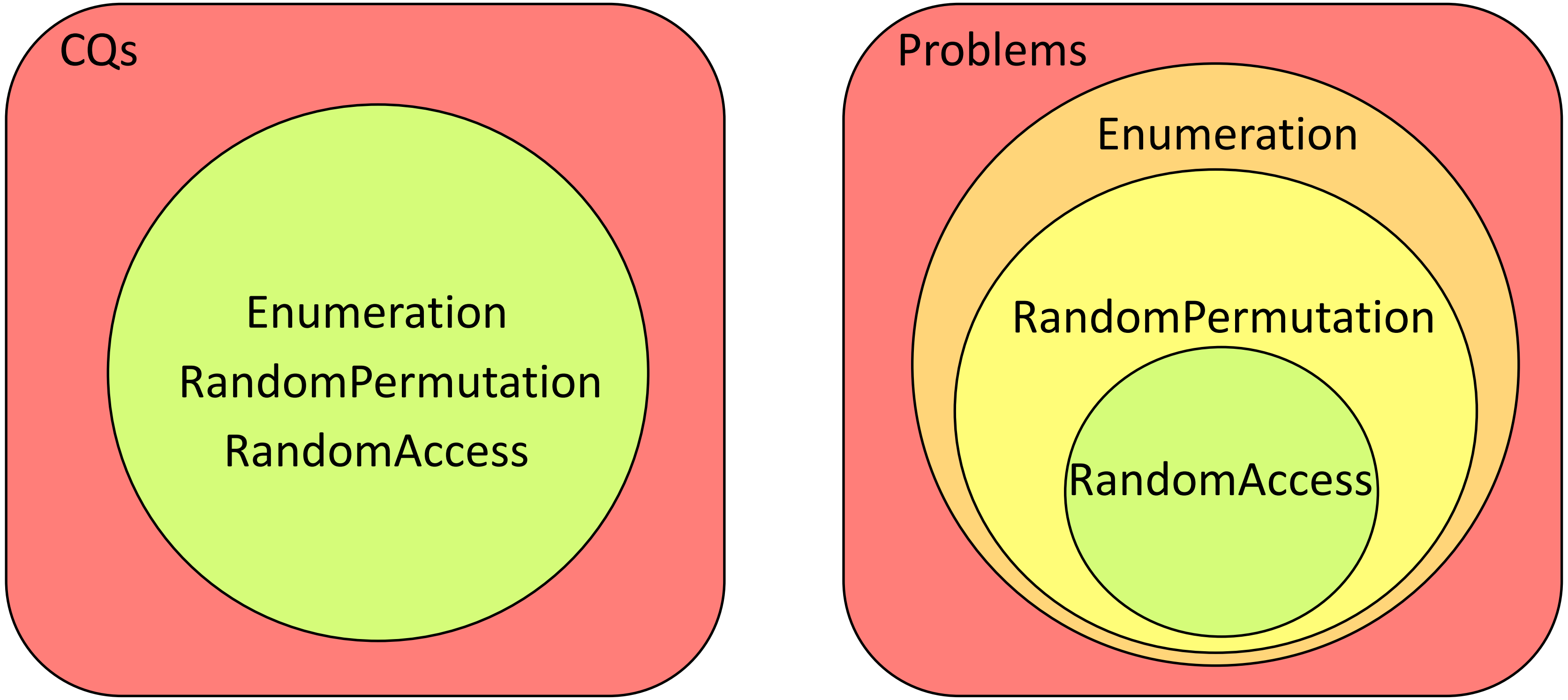
Also efficient counting, membership testing, etc.

Assuming the hardness of Boolean matrix multiplication.

Cannot find any answer in $O(n)$ time, assuming the hardness of finding hypercliques.

The lower bounds assume no self-joins

CQs Dichotomy

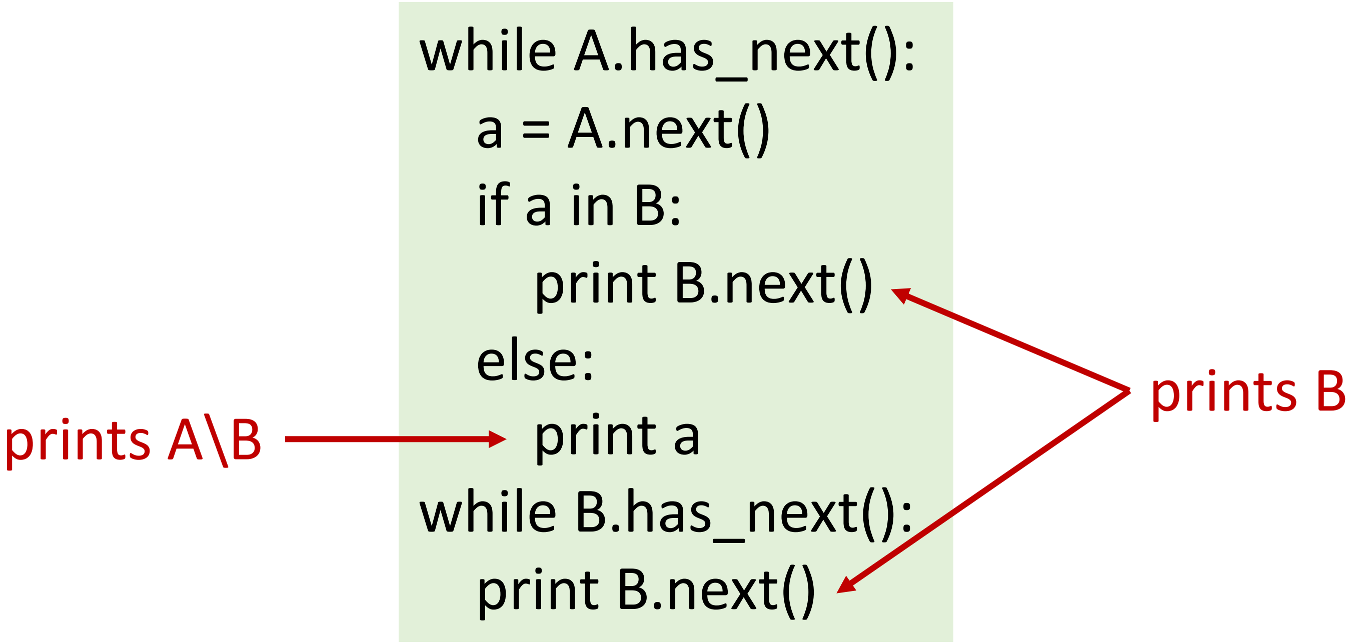


Tasks & Motivation
Conjunctive Queries

Unions of Conjunctive Queries

Enumeration: Easy ∪ Easy = Easy

[DurandStrozecki CSL’2011]



```
while A.has_next():
    a = A.next()
    if a in B:
        print B.next()
    else:
        print a
while B.has_next():
    print B.next()
```

prints A\B

prints B

A\B and B are a partition of A∪B

Cases for UCQs Enumeration

[CarmeliKröll PODS’2019]

All CQs are Easy

always easy

Some CQs are Hard

sometimes hard

sometimes easy

All CQs are Hard

sometimes hard

sometimes easy

* Even when considering non-redundant unions

Some UCQs containing only hard CQs are easy!

Access: Easy ∪ Easy = Sometimes Hard

Proof (Example):

- $Q_1(x, y, z) \leftarrow R(x, y), S(y, z)$ free-connex
- $Q_2(x, y, z) \leftarrow S(y, z), T(x, z)$ free-connex
- $Q_1 \cap Q_2(x, y, z) \leftarrow R(x, y), S(y, z), T(x, z)$ cyclic
  - Cannot count in linear time

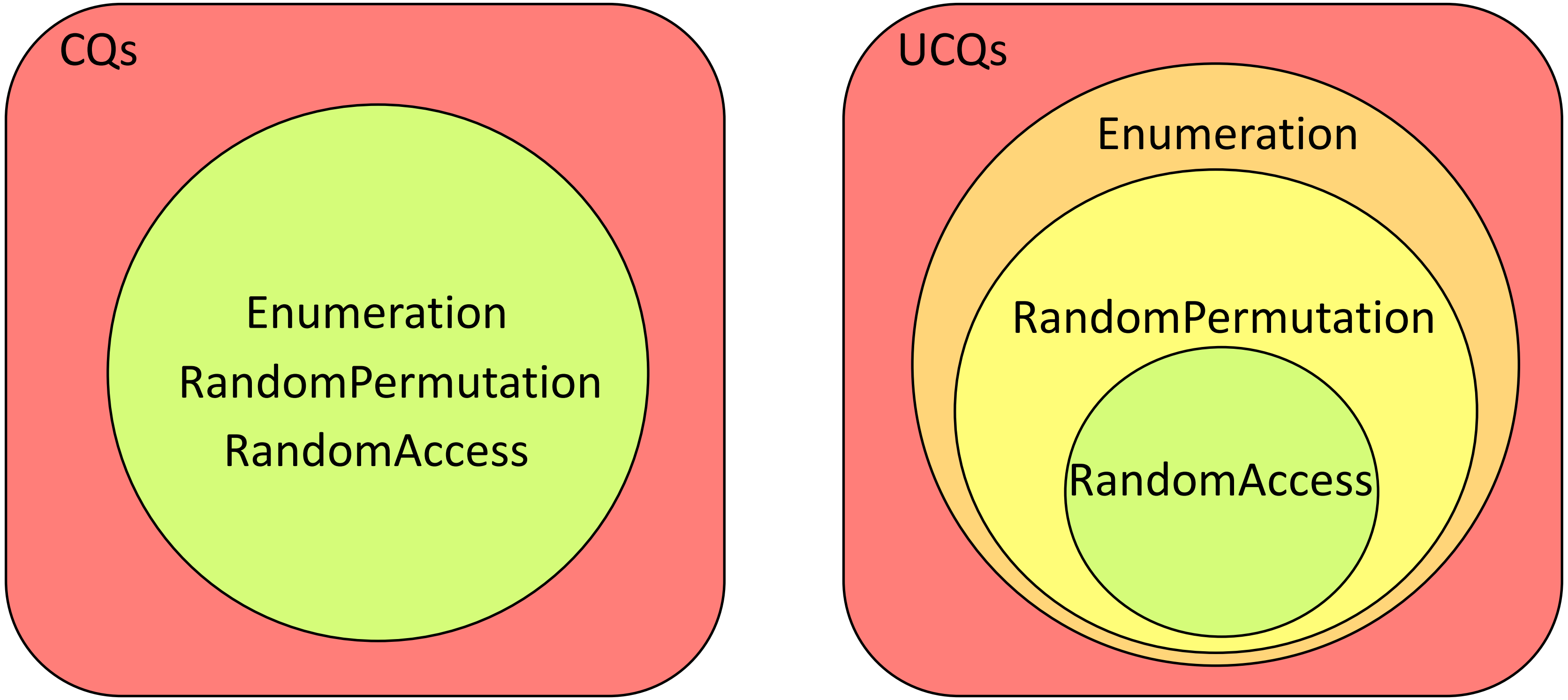
    * assumption: cannot find a triangle in a graph in linear time.

- Assume by contradiction $Q_1 \cup Q_2 \in$ RandomAccess
  - We can count $|Q_1 \cup Q_2|$ in linear time
  - Computes $|Q_1 \cap Q_2| = |Q_1| + |Q_2| - |Q_1 \cup Q_2|$

Contradiction!

Comparing the Tasks

- UCQs: Enumeration $\not\Rightarrow$ RandomAccess



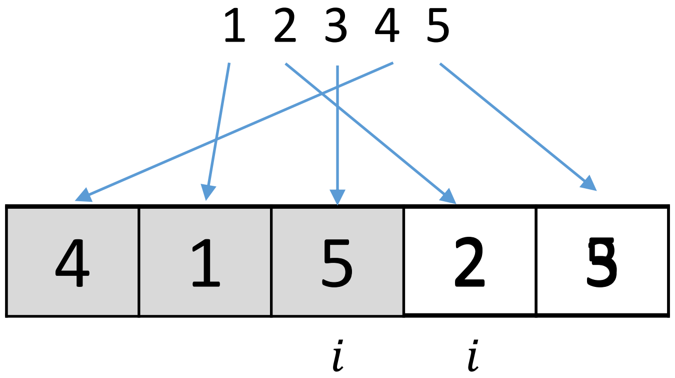
Unions of Free-connex CQs

- Random access is not always possible
- What can we do?

1. Mutually Compatible UCQs
    - Subclass, allows for random access in $\log^2$ time
2. Relax the delay requirements
    - Random permutation algorithm with expected log delay

Easy ∪ Easy: Random Permutation

- Random permutation algorithm for a union
- Requirements from each CQ:
  - Counting
  - Sampling
  - Testing
  - Deletion
- Free-connex CQs admit:
  - Counting
  - Random access
  - Inverted random access



Deletion:

1. Get the answer index
2. Swap the index with i
3. i++

Easy ∪ Easy: Random Permutation

Algorithm

```
while $\sum_j |Q_j| > 0$:
    choose $Q_i$ with probability $\frac{|Q_i|}{\sum_j |Q_j|}$
    $ans$ = random answer of $Q_i$

        We don't need this part

        delete $ans$ from $Q_i$
        print $ans$
```

Example

If the answers are disjoint,

Q_1	a	b	c	d
Q_2	e	f	g	

Probability of d : $\underbrace{\frac{4}{4+3}}_{\text{Choosing } Q_1}\underbrace{\frac{1}{4}}_{\text{Choosing d}} = \frac{1}{7}$

Every answer is selected with probability $\frac{1}{7}$

Easy ∪ Easy: Random Permutation

Algorithm

```
while $\sum_j |Q_j| > 0$:
    choose $Q_i$ with probability $\frac{|Q_i|}{\sum_j |Q_j|}$
    $ans$ = random answer of $Q_i$

    We don't need this part

        delete $ans$ from $Q_i$
        print $ans$
```

Example

Q_1	a	b	c	d
Q_2	e	f	b	

Every cell is selected with probability $\frac{1}{7}$

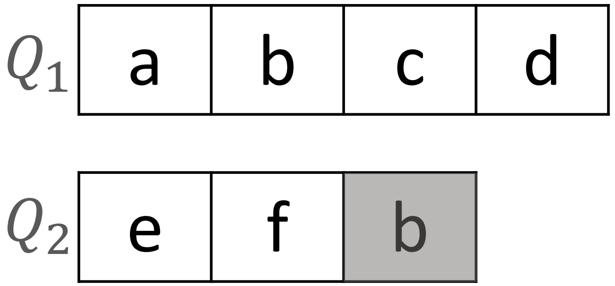
b is selected with probability $\frac{2}{7}$

Easy ∪ Easy: Random Permutation

Algorithm

```
while $\sum_j |Q_j| > 0$:
    choose $Q_i$ with probability $\frac{|Q_i|}{\sum_j |Q_j|}$
    $ans$ = random answer of $Q_i$
    $providers$ = $\{Q_j | ans \in Q_j\}$
    $owner$ = first from $providers$
    for $Q_j \in providers \setminus \{owner\}$
        delete $ans$ from $Q_j$
    If $owner = Q_i$:
        delete $ans$ from $Q_i$
        print $ans$
```

Example



Q_1	a	b	c	d
Q_2	e	f	b	

Every cell is selected with probability $\frac{1}{7}$

b is selected with probability $\frac{1}{7}$

No answer with probability $\frac{1}{7}$

Easy ∪ Easy: Random Permutation

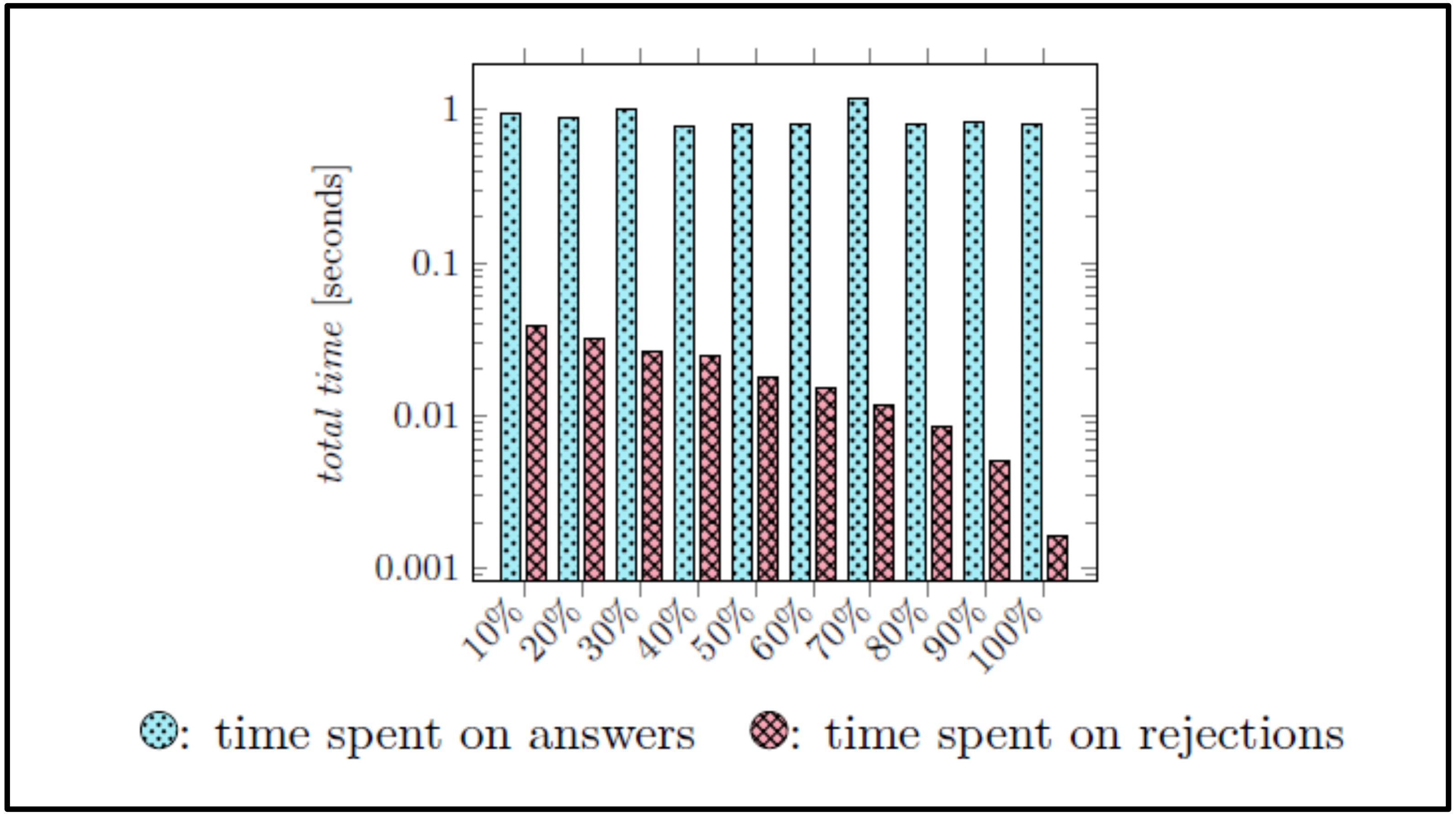
Algorithm

```
while $\sum_j |Q_j| > 0$:
    choose $Q_i$ with probability $\frac{|Q_i|}{\sum_j |Q_j|}$
    $ans$ = random answer of $Q_i$
    $providers$ = $\{Q_j | ans \in Q_j\}$
    $owner$ = first from $providers$
    for $Q_j \in providers \setminus \{owner\}$
        delete $ans$ from $Q_j$
    If $owner = Q_i$:
        delete $ans$ from $Q_i$
        print $ans$
```

- Constant number of operations per iteration
- Each operation takes log time
  → Each iteration takes log time
- Every iteration prints with probability $\frac{1}{\#Queries} \leq P \leq 1$
  → Expected log delay
- At most two iterations per answer
  → Amortized log delay

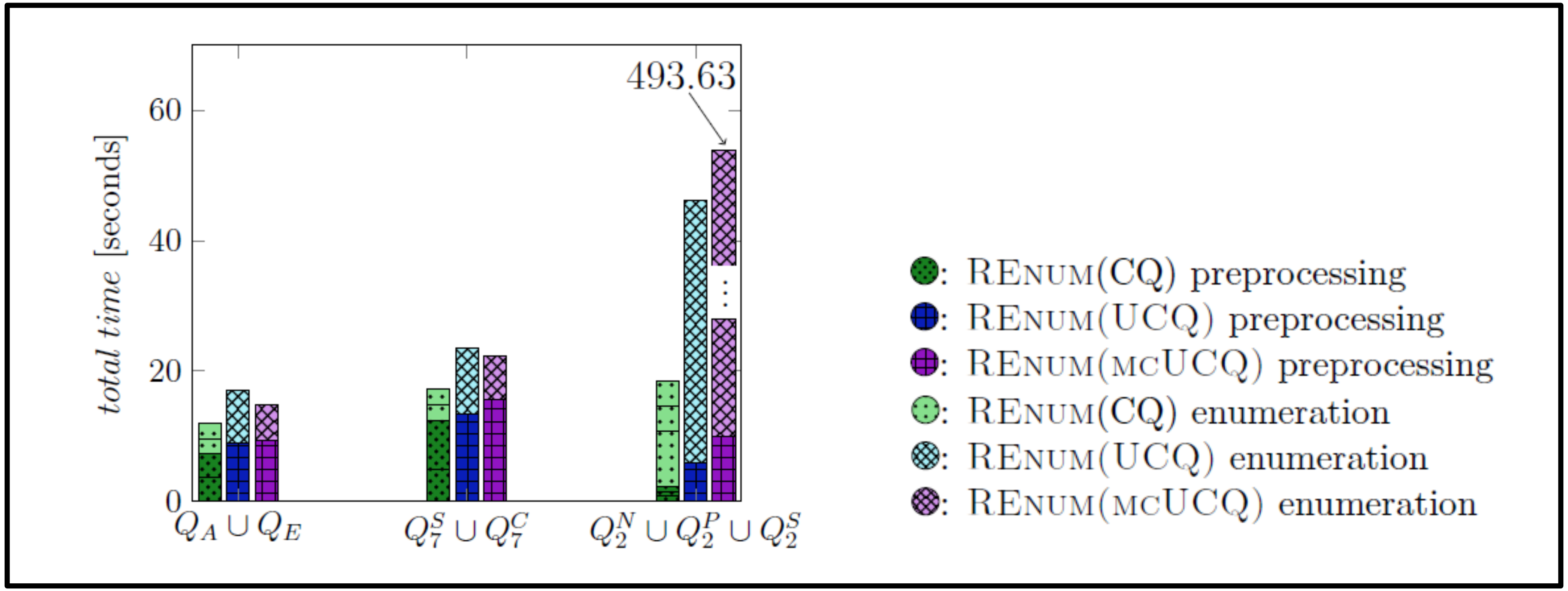
In Practice

- Time spent on rejections declines with time

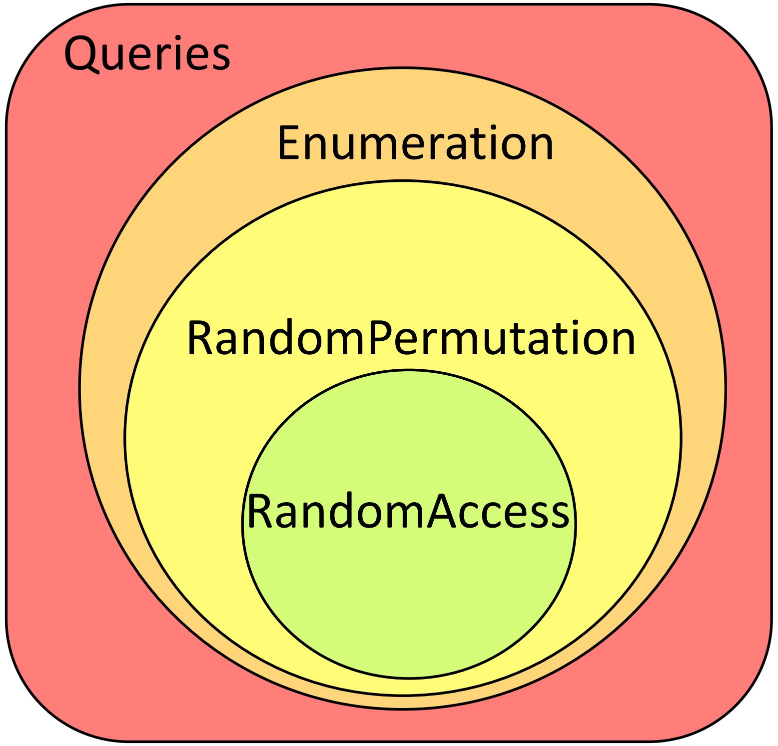


In Practice

- Compares the UCQ alternatives
- Demonstrates the overhead caused by the union



Conclusions



- CQs:
  - 3 tasks tractable ⇔ free-connex
- UCQs:
  - Enumeration ⇏ RandomAccess
  - mcUCQs: 3 tasks tractable
  - Union of free-connex: RandomPermutation with expected log delay
- Future Work:
  - Characterizing unions of free-connex CQs
  - Reducing space consumption