Conjunctive Queries with Free Access Patterns under Updates

Ahmet Kara, Milos Nikolic, Dan Olteanu, Haozhe Zhang

29th March 2023

ICDT 2023, Ioannina, Greece

Scenario

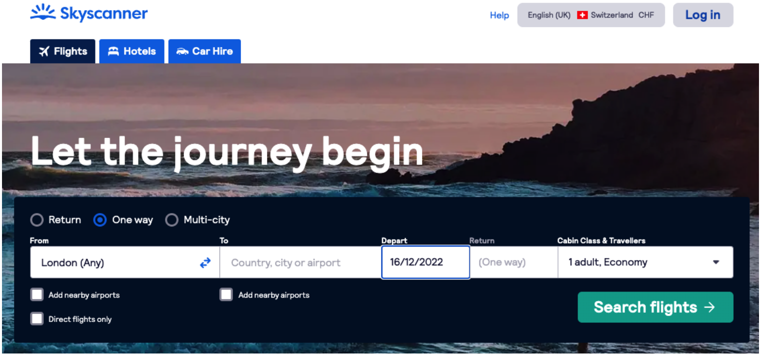


Figure: Flight Booking Interface

Scenario

The flight booking company has a database with the two relations:

- `Flight: (FlightNo, DEPAirport, ARRAirport, Date, Price)`
- `Airport: (Airport, Name, City)`

List the flights in the database with the airport information:

```
FlightSearch(FlightNo, DEPCity, ARRCity, Date) :-
    Flight(FlightNo, DEPAirport, ARRAirport, Date, Price),
    DEPAirport(DEPAirport, Name, DEPCity),
    ARRAirport(ARRAirport, Name, ARRCity).
```

Scenario

The flight booking company has a database with the two relations:

- `Flight: (FlightNo, DEPAirport, ARRAirport, Date, Price)`
- `Airport: (Airport, Name, City)`

List the flights from **London** to **Zurich** on **1st January 2023**:

```
FlightSearch(FlightNo, "London", "Zurich", "2023-01-01") :-
    Flight(FlightNo, DEPAirport, ARRAirport, "2023-01-01", Price),
    DEPAirport(DEPAirport, Name, "London"),
    ARRAirport(ARRAirport, Name, "Zurich").
```

Challenges

Query with parameters %DEPCity, %ARRCity and %Date:

```
FlightSearch(FlightNo, "%DEPCity", "%ARRCity", "%Date") :-
    Flight(FlightNo, DEPAirport, ARRAirport, "%Date", Price),
    DEPAirport(DEPAirport, Name, "%DEPCity"),
    ARRAirport(ARRAirport, Name, "%ARRCity").
```

Challenges

Query with parameters `%DEPCity`, `%ARRCity` and `%Date`:

```
FlightSearch(FlightNo, "%DEPCity", "%ARRCity", "%Date") :-
    Flight(FlightNo, DEPAirport, ARRAirport, "%Date", Price),
    DEPAirport(DEPAirport, Name, "%DEPCity"),
    ARRAirport(ARRAirport, Name, "%ARRCity").
```

- This query is asked frequently by many users with different dates and departure and arrival cities.

Challenges

Query with parameters `%DEPCity`, `%ARRCity` and `%Date`:

```
FlightSearch(FlightNo, "%DEPCity", "%ARRCity", "%Date") :-
    Flight(FlightNo, DEPAirport, ARRAirport, "%Date", Price),
    DEPAirport(DEPAirport, Name, "%DEPCity"),
    ARRAirport(ARRAirport, Name, "%ARRCity").
```

- This query is asked frequently by many users with different dates and departure and arrival cities.
- The database is subject to frequent updates, e.g., new flights are added, existing flights are cancelled, etc.

Conjunctive Queries with Free Access Patterns

We formalize such data access by ***Conjunctive Queries with Free Access Patterns*** (CQAPs)

```
FlightSearch(FlightNo | DEPCity, ARRCity, Date) =
    Flight(FlightNo, DEPAirport, ARRAirport, Date, Price),
    DEPAirport(DEPAirport, Name, DEPCity),
    ARRAirport(ARRAirport, Name, ARRCity)
```

Conjunctive Queries with Free Access Patterns

We formalize such data access by ***Conjunctive Queries with Free Access Patterns*** (CQAPs)

```
FlightSearch(FlightNo | DEPCity, ARRCity, Date) =
    Flight(FlightNo, DEPAirport, ARRAirport, Date, Price),
    DEPAirport(DEPAirport, Name, DEPCity),
    ARRAirport(ARRAirport, Name, ARRCity)
```

- Free variables are partitioned into ***input*** and ***output*** variables
  - **Input**: `DEPCity, ARRCity, Date`
  - **Output**: `FlightNo`

Conjunctive Queries with Free Access Patterns

We formalize such data access by ***Conjunctive Queries with Free Access Patterns*** (CQAPs)

```
FlightSearch(FlightNo | DEPCity, ARRCity, Date) =
    Flight(FlightNo, DEPAirport, ARRAirport, Date, Price),
    DEPAirport(DEPAirport, Name, DEPCity),
    ARRAirport(ARRAirport, Name, ARRCity)
```

- Free variables are partitioned into ***input*** and ***output*** variables
  - **Input**: `DEPCity, ARRCity, Date`
  - **Output**: `FlightNo`
- Access request: Given **a tuple over the input variables**, the query yields **the tuples over the output variables** such that the body of the query is satisfied

Conjunctive Queries with Free Access Patterns

We formalize such data access by ***Conjunctive Queries with Free Access Patterns*** (CQAPs)

```
FlightSearch(FlightNo | DEPCity, ARRCity, Date) =
    Flight(FlightNo, DEPAirport, ARRAirport, Date, Price),
    DEPAirport(DEPAirport, Name, DEPCity),
    ARRAirport(ARRAirport, Name, ARRCity)
```

- Free variables are partitioned into ***input*** and ***output*** variables
  - **Input**: `DEPCity, ARRCity, Date`
  - **Output**: `FlightNo`
- Access request: Given **a tuple over the input variables**, the query yields **the tuples over the output variables** such that the body of the query is satisfied
- Also called *parameterized queries* or *prepared statements* in DBMS

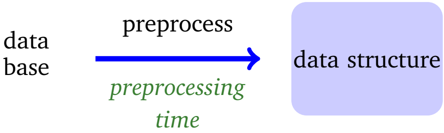
Problem Setting

We consider the problem of **fully dynamic evaluation** for **CQAPs**:

- Maintaining the database under tuple updates (inserts or deletes)
- Answering access requests

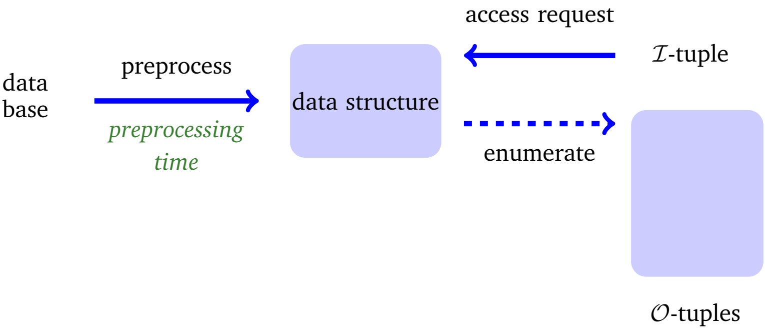
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



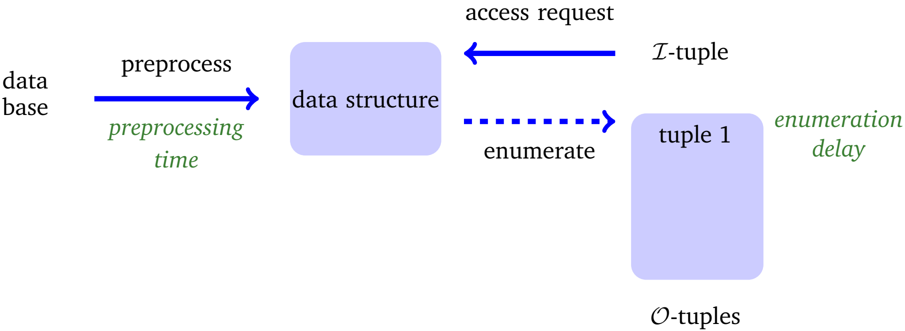
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



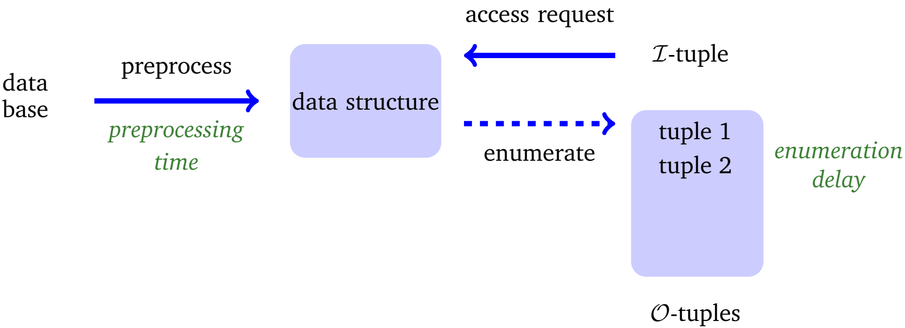
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



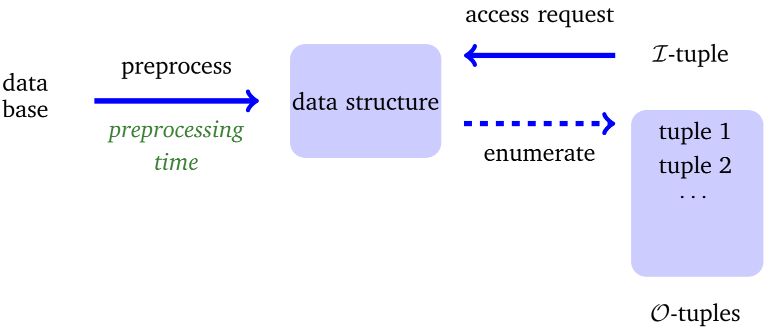
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



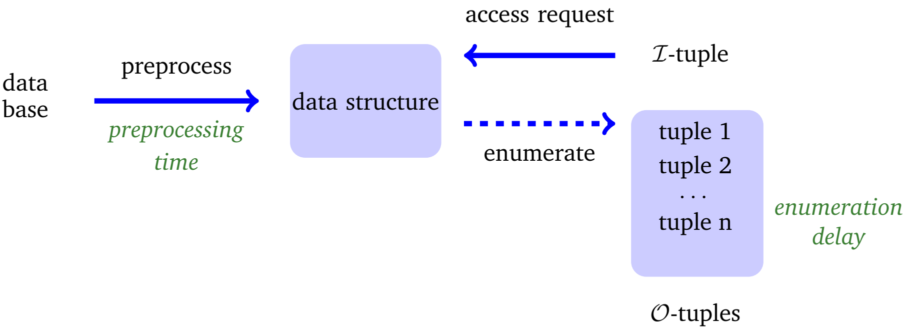
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



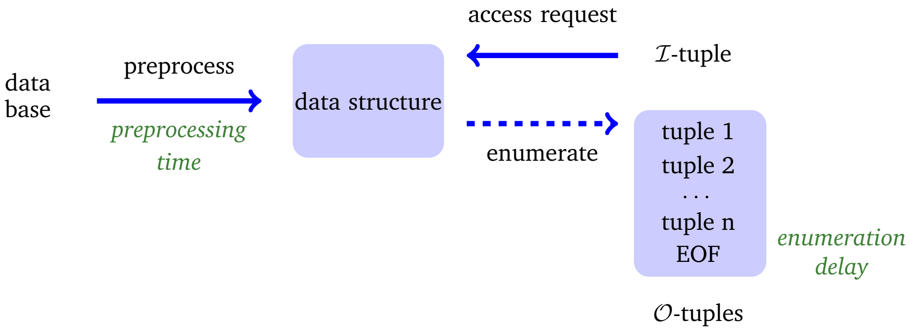
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



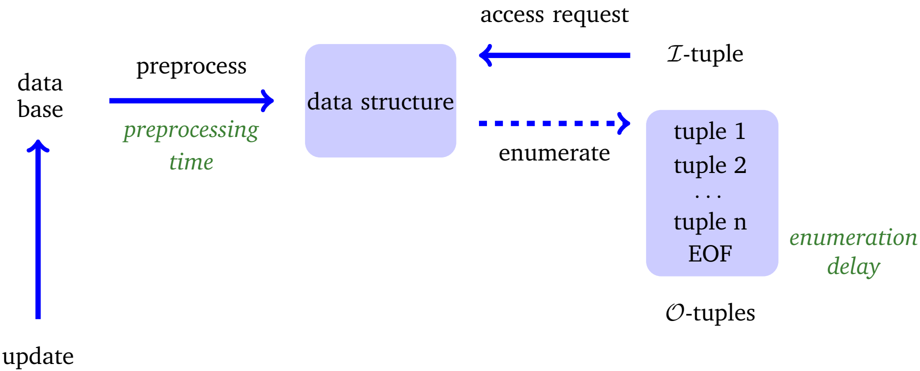
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



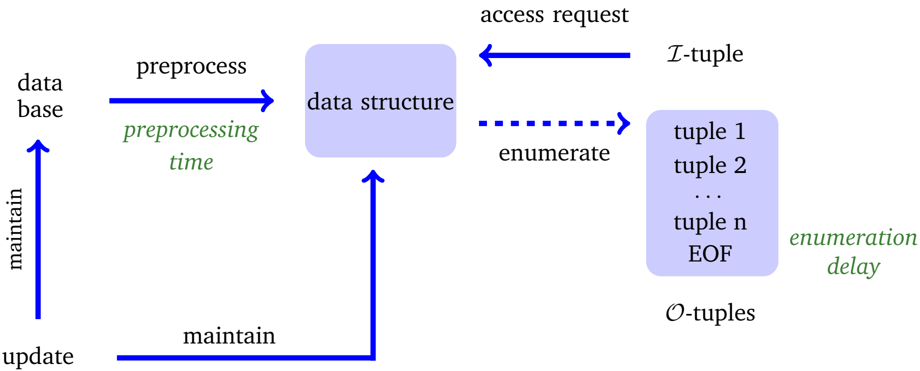
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



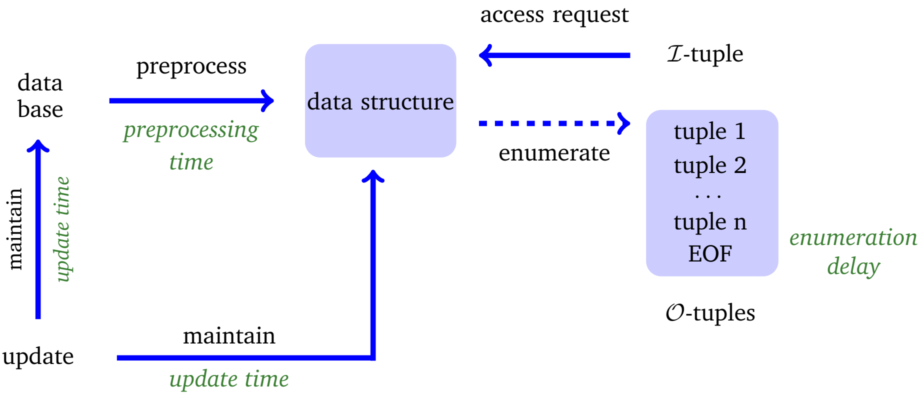
Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.



Dynamic Query Evaluation Framework

Given a CQAP $Q(\mathcal{O} \mid \mathcal{I})$, compute a data structure that supports answering the access requests and maintains it under updates.

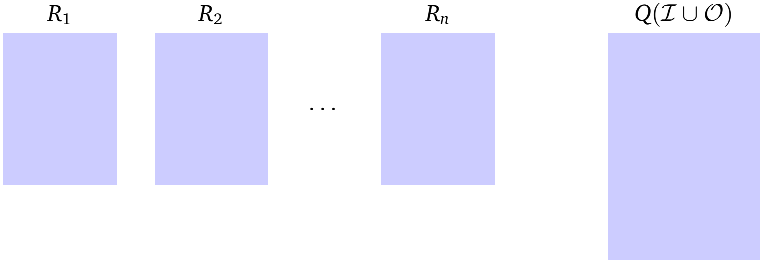


Mainstream Approaches

Eager approach:

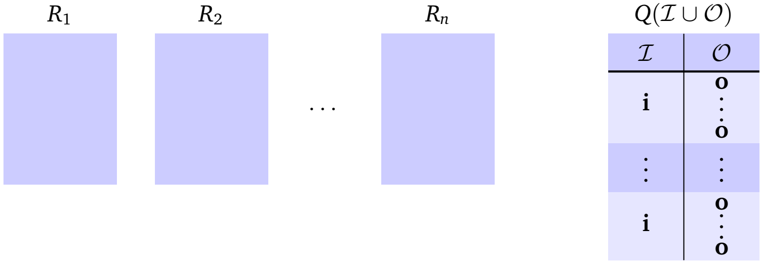
- Preprocessing: Compute the **full result** and create an index for the access pattern
- Enumeration: Lookup the index and return the result
- Update: Maintain the full result and the index under each update

Mainstream Approaches: Eager Approach



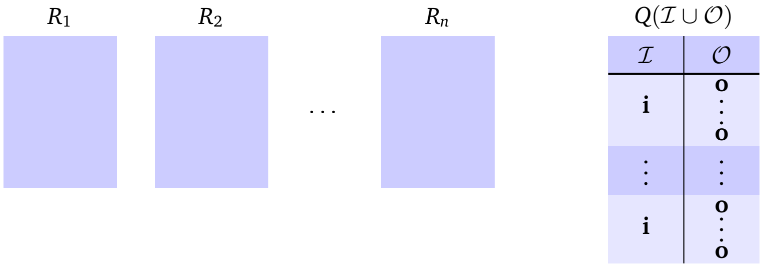
Preprocessing: Compute the **full result** of the query and create an index for the access pattern

Mainstream Approaches: Eager Approach



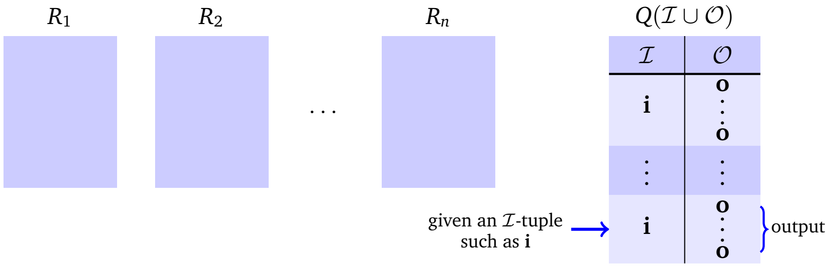
Preprocessing: Compute the **full result** of the query and create an index for the access pattern

Mainstream Approaches: Eager Approach



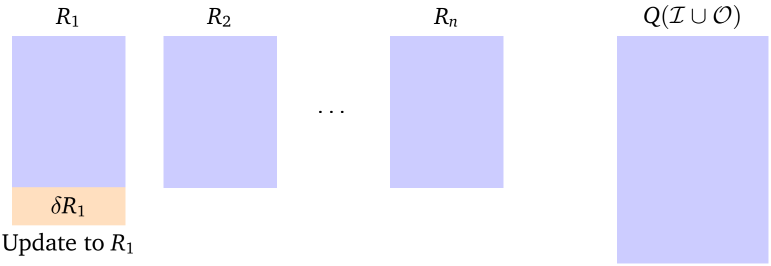
Enumeration: Lookup the index and output the result tuple by tuple

Mainstream Approaches: Eager Approach



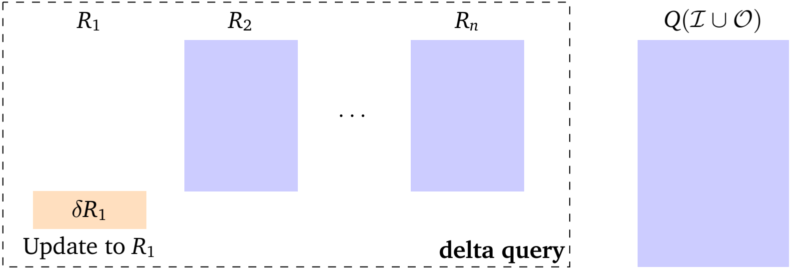
Enumeration: Lookup the index and output the result tuple by tuple

Mainstream Approaches: Eager Approach



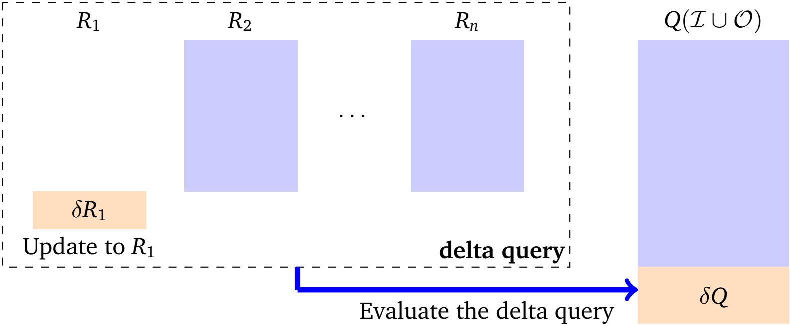
Update: Maintain the full result and the index under each update

Mainstream Approaches: Eager Approach



Update: Maintain the full result and the index under each update

Mainstream Approaches: Eager Approach



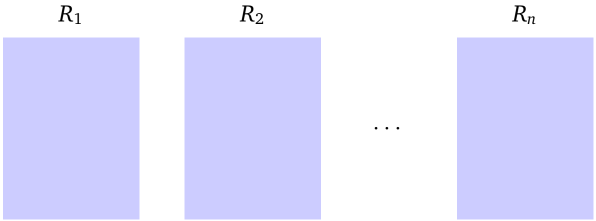
Update: Maintain the full result and the index under each update

Mainstream Approaches

Lazy approach:

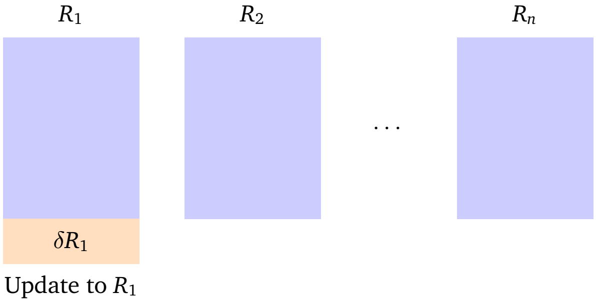
- **No preprocessing**
- Enumeration: Upon each access request, set the input variables to constants and evaluate the residual query
- Update: Maintain only the base relations

Mainstream Approaches: Lazy Approach



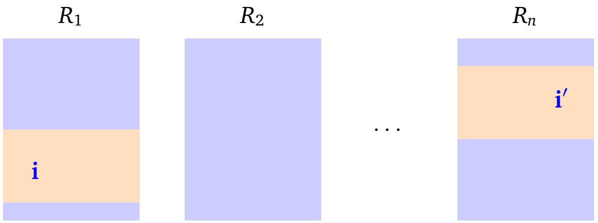
Preprocessing: **No preprocessing**

Mainstream Approaches: Lazy Approach



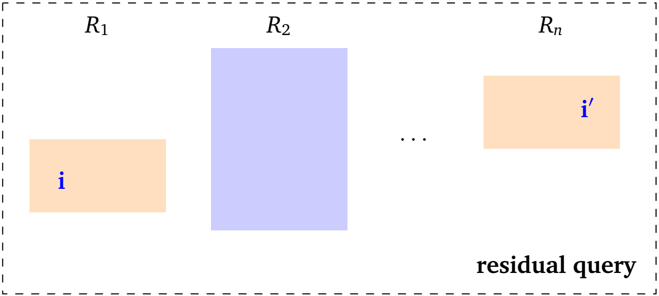
Update: Maintain only the base relations

Mainstream Approaches: Lazy Approach



Enumeration: Given **a tuple over the input variables**, set the input variables to constants and evaluate the residual query

Mainstream Approaches: Lazy Approach



Enumeration: Given **a tuple over the input variables**, set the input variables to constants and evaluate the residual query

Contributions of the Paper

We have discussed the two mainstream approaches in the previous slides.

Contributions of the Paper

We have discussed the two mainstream approaches in the previous slides.

We propose two algorithms that **improve** the mainstream approaches:

Contributions of the Paper

We have discussed the two mainstream approaches in the previous slides.

We propose two algorithms that **improve** the mainstream approaches:

1. **Avoiding full materialization** in the eager approach while keeping constant-delay enumeration

Contributions of the Paper

We have discussed the two mainstream approaches in the previous slides.

We propose two algorithms that **improve** the mainstream approaches:

1. **Avoiding full materialization** in the eager approach while keeping constant-delay enumeration
2. Understanding the update time - enumeration delay **trade-off** space
   - Eager and lazy approaches are extremes in this space

Contribution 1: Eager-Factorized Approach

The **eager** approach maintains the **listing representation** of the CQAP result.

Contribution 1: Eager-Factorized Approach

The **eager** approach maintains the **listing representation** of the CQAP result.

We propose the **eager-factorized** approach that maintains a set of **factorized representations** that is **specialized for the access pattern**:

Contribution 1: Eager-Factorized Approach

The **eager** approach maintains the **listing representation** of the CQAP result.

We propose the **eager-factorized** approach that maintains a set of **factorized representations** that is **specialized for the access pattern**:

1. Decompose the CQAP into a set of (smaller) **sub-queries**

Contribution 1: Eager-Factorized Approach

The **eager** approach maintains the **listing representation** of the CQAP result.

We propose the **eager-factorized** approach that maintains a set of **factorized representations** that is **specialized for the access pattern**:

1. Decompose the CQAP into a set of (smaller) **sub-queries**
2. For each sub-query, maintain a **factorized representation** of its result
   - More succinct than the listing representation of the query result
   - Less time to compute and maintain

Contribution 1: Eager-Factorized Approach

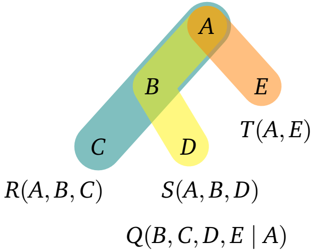
The **eager** approach maintains the **listing representation** of the CQAP result.

We propose the **eager-factorized** approach that maintains a set of **factorized representations** that is **specialized for the access pattern**:

1. Decompose the CQAP into a set of (smaller) **sub-queries**
2. For each sub-query, maintain a **factorized representation** of its result
   - More succinct than the listing representation of the query result
   - Less time to compute and maintain
3. The factorized representations support
   - Access requests with any values over the input variables
   - Constant-delay enumeration

Example 1

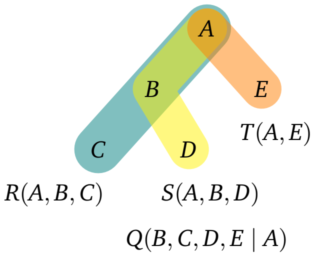
Consider the query $Q(B, C, D, E \mid A) = R(A, B, C), S(A, B, D), T(A, E)$



	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$

Example 1

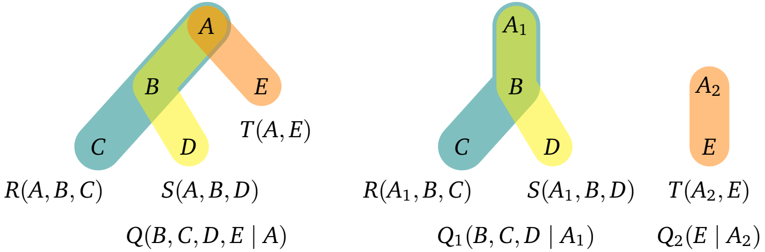
Consider the query $Q(B, C, D, E \mid A) = R(A, B, C), S(A, B, D), T(A, E)$



	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

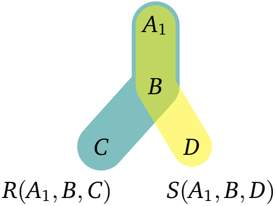
Consider the query $Q(B, C, D, E \mid A) = R(A, B, C), S(A, B, D), T(A, E)$



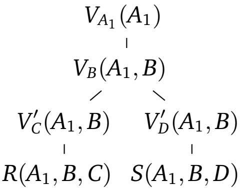
	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



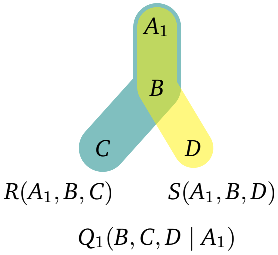
$Q_1(B, C, D \mid A_1)$

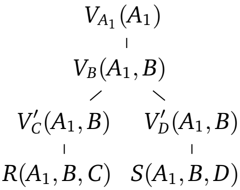


	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



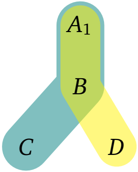


Given an input A_1-value a

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

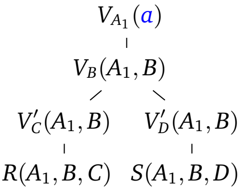
Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

$Q_1(B, C, D \mid A_1)$



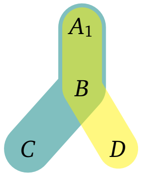
Given an input A_1-value a

check whether $a \in V_{A_1}(A)$

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

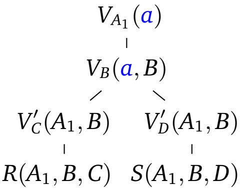
Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

$Q_1(B, C, D \mid A_1)$



Given an input A_1-value a

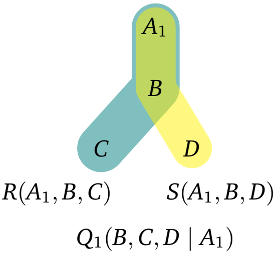
check whether $a \in V_{A_1}(A)$

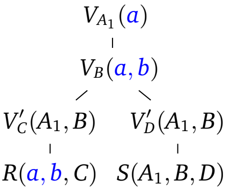
for each $b \in V_B(a, B)$

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$





Given an input A_1-value a

check whether $a \in V_{A_1}(A)$

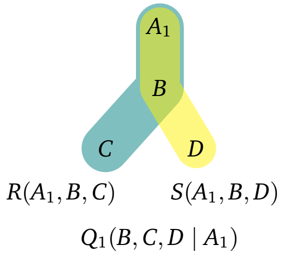
for each $b \in V_B(a, B)$

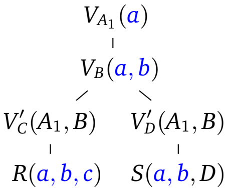
for each $c \in R(a, b, C)$

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$





Given an input A_1-value a

check whether $a \in V_{A_1}(A)$

for each $b \in V_B(a, B)$

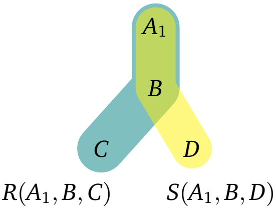
for each $c \in R(a, b, C)$

for each $d \in S(a, b, D)$

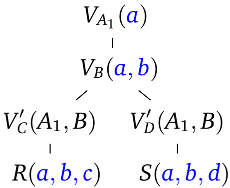
	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



$Q_1(B, C, D \mid A_1)$



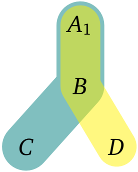
Given an input A_1-value a

check whether $a \in V_{A_1}(A)$
for each $b \in V_B(a, B)$
  for each $c \in R(a, b, C)$
    for each $d \in S(a, b, D)$
      output (a, b, c, d)

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

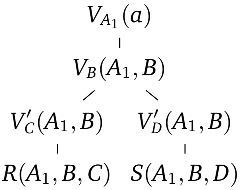
Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

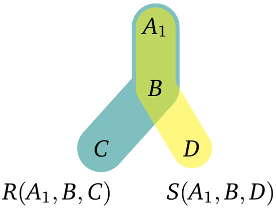
$Q_1(B, C, D \mid A_1)$



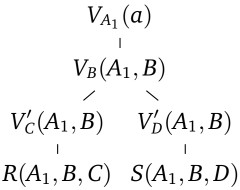
	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



$Q_1(B, C, D \mid A_1)$

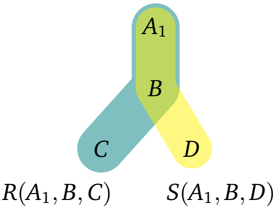


Given an update $\delta R(a, b, c)$

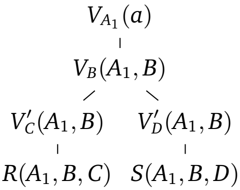
	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

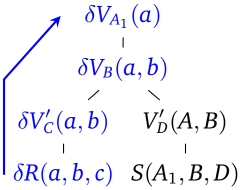
Example 1

Consider the sub-query $Q_1(B, C, D \mid A_1) = R(A_1, B, C), S(A_1, B, D)$



$Q_1(B, C, D \mid A_1)$



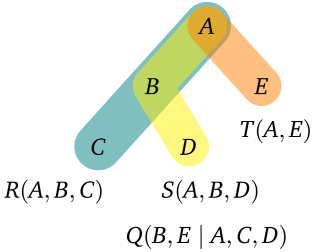


Given an update $\delta R(a, b, c)$

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Example 2

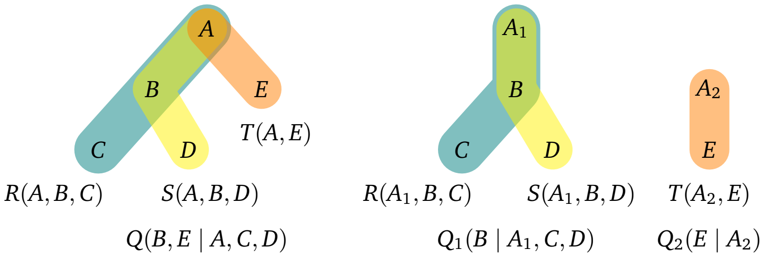
Consider the query $Q(B, E \mid A, C, D) = R(A, B, C), S(A, B, D), T(A, E)$



	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

Example 2

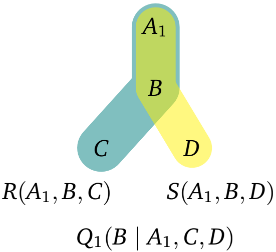
Consider the query $Q(B, E \mid A, C, D) = R(A, B, C), S(A, B, D), T(A, E)$

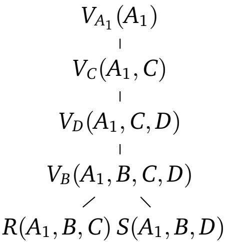


	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

Example 2

Consider the sub-query $Q_1(B \mid A_1, C, D) = R(A_1, B, C), S(A_1, B, D)$

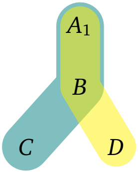




	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

Example 2

Consider the sub-query $Q_1(B \mid A_1, C, D) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

$Q_1(B \mid A_1, C, D)$

$V_{A_1}(a)$

$V_C(a, c)$

$V_D(a, c, d)$

$V_B(a, c, d, B)$

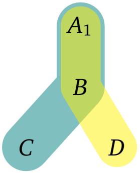
$R(A_1, B, C)$ $S(A_1, B, D)$

Given an input tuple (a, c, d) over (A_1, C, D)

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

Example 2

Consider the sub-query $Q_1(B \mid A_1, C, D) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

$Q_1(B \mid A_1, C, D)$

$V_{A_1}(a)$

$V_C(a, c)$

$V_D(a, c, d)$

$V_B(a, c, d, B)$

$R(A_1, B, C)$ $S(A_1, B, D)$

Given an input tuple (a, c, d) over (A_1, C, D)

check $(a, c, d) \in V_D(A_1, C, D)$

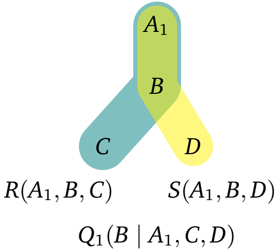
for each $b \in V_B(a, c, d, B)$

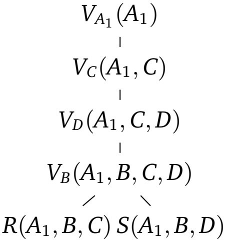
output (a,b,c,d)

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

Example 2

Consider the sub-query $Q_1(B \mid A_1, C, D) = R(A_1, B, C), S(A_1, B, D)$

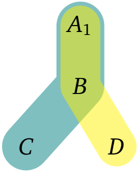




	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

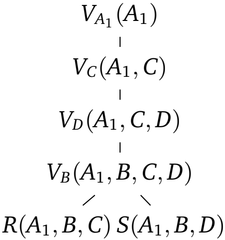
Example 2

Consider the sub-query $Q_1(B \mid A_1, C, D) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

$Q_1(B \mid A_1, C, D)$

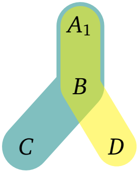


Given an update $\delta R(a, b, c)$

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

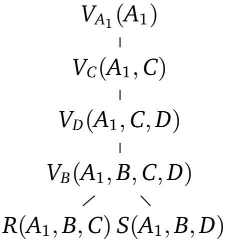
Example 2

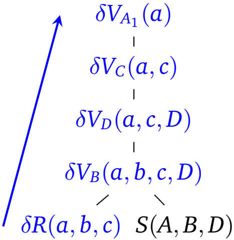
Consider the sub-query $Q_1(B \mid A_1, C, D) = R(A_1, B, C), S(A_1, B, D)$



$R(A_1, B, C)$ $S(A_1, B, D)$

$Q_1(B \mid A_1, C, D)$





Given an update $\delta R(a, b, c)$

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$

Different Access Patterns Result in Different Costs

Consider the query $Q(\mathcal{O} \mid \mathcal{I}) = R(A,B,C), S(A,B,D), T(A,E)$. The table shows the evaluation costs of our approach for different access patterns.

$\mathcal{O}$	$\mathcal{I}$	Preprocessing	Delay	Update
$\{A,B,C,D,E\}$	$\{\,\}$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\{\,\}$	$\{A,B,C,D,E\}$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\{A,C,D,E\}$	$\{B\}$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$
$\{A,C,D\}$	$\{B,E\}$	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$
$\{A,E\}$	$\{B,C,D\}$	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
$\{A,B\}$	$\{C,D,E\}$	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
...	...	...	...	...

	Preprocessing	Delay	Update
Eager	$\mathcal{O}(N^3)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$

Summary of Contribution 1

- A dynamic evaluation approach for arbitrary CQAPs
- For a CQAP with *static width* w and *dynamic width* δ, our approach admits
  - Preprocessing: $\mathcal{O}(N^{\mathsf{w}})$
  - Update: $\mathcal{O}(N^{\delta})$
  - Enumeration delay: $\mathcal{O}(1)$
- Static width: $s^{\uparrow}$[OZ15] or faqw[AKNR16] specialized to the access pattern
- Dynamic width: maximal static width over all delta queries

Dichotomy Result

$CQAP_0$: static width $\mathsf{w} = 1$ and dynamic width $\delta = 0$

Consider a CQAP query Q and a database of size N.

- If Q is in $CQAP_0$, then it admits $\mathcal{O}(N)$ preprocessing time, $\mathcal{O}(1)$ enumeration delay, and $\mathcal{O}(1)$ update time for single-tuple updates.
- If Q is not in $CQAP_0$ and has no repeating relation symbols, then there is no algorithm that computes Q with arbitrary preprocessing time, $\mathcal{O}(N^{\frac{1}{2}-\gamma})$ enumeration delay, and $\mathcal{O}(N^{\frac{1}{2}-\gamma})$ amortized update time, for any $\gamma > 0$, unless the OMv conjecture fails.

Contribution 2: Trade-Offs in Dynamic Query Evaluation

The eager (listing or factorized) and lazy approaches are the two extremes.

Eager	Lazy
Constant enumeration delay	**High** enumeration delay
High update time	**Constant** update time

Contribution 2: Trade-Offs in Dynamic Query Evaluation

The eager (listing or factorized) and lazy approaches are the two extremes.

Eager

Lazy

Constant enumeration delay

High update time

Performs well when **updates** are less frequent than **access requests**

High enumeration delay

Constant update time

Performs well when **access requests** are less frequent than **updates**

Contribution 2: Trade-Offs in Dynamic Query Evaluation

The eager (listing or factorized) and lazy approaches are the two extremes.

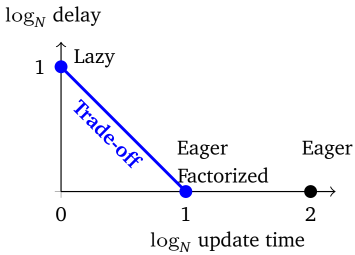
Eager	Ours	Lazy
Constant enumeration delay **High** update time	Trade-offs ⟷	**High** enumeration delay **Constant** update time
Performs well when **updates** are less frequent than **access requests**		Performs well when **access requests** are less frequent than **updates**

Contribution 2: Trade-Offs in Dynamic Query Evaluation

Example: Consider query $Q(A, C, D, E \mid B) = R(A, B, C), S(A, B, D), T(A, E)$.

	Delay	Update
Eager	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$
Eager-Factorized	$\mathcal{O}(1)$	$\mathcal{O}(N)$
Lazy	$\mathcal{O}(N)$	$\mathcal{O}(1)$
Trade-off	$\mathcal{O}(N^{1-\epsilon})$	$\mathcal{O}(N^{\epsilon})$

Evaluation costs of different approaches; $\epsilon \in [0, 1]$



Contribution 2: Trade-Offs in Dynamic Query Evaluation

Trade-offs between preprocessing time, enumeration delay and update time

For **a CQAP with a hierarchical fracture** with

- Static width w
- Dynamic width δ (δ can be w or w $-$ 1)

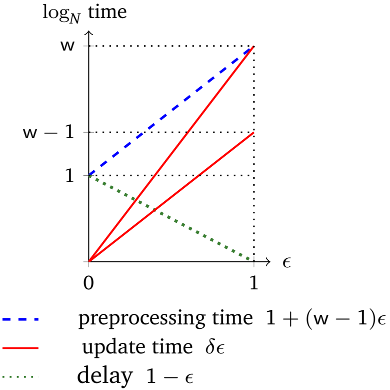
Optimality results for the CQAPs with hierarchical fractures

- Strongly Pareto optimal:

  $CQAP_0$ (w $= 1$, $\delta = 0$)

- Weakly Pareto optimal:

  $CQAP_1$ (w $= 1$, $\delta = 1$)



Summary

Fully dynamic evaluation for CQAPs

1. A dynamic evaluation approach for arbitrary CQAPs
2. A dichotomy result in the evaluation of CQAPs
3. Evaluation trade-offs for CQAPs

References I

[AKNR16] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *PODS*, pages 13–28, 2016.

[OZ15] Dan Olteanu and Jakub Závodný. Size Bounds for Factorised Representations of Query Results. *ACM TODS*, 40(1):2:1–2:44, 2015.

Appendix: Query Fracture

Consider the triangle query

$$Q(B, C \mid A) = R(A, B), S(B, C), T(A, C).$$

By replacing A with A_1 and A_2, we get

$$Q'(B, C \mid A_1, A_2) = R(A_1, B), S(B, C), T(C, A_2).$$

The static width increases from $\mathsf{w}(Q) = 1.5$ to $\mathsf{w}(Q') = 2$.

Appendix: Query Fracture

Consider the 4-cycle query

$$Q(B, D|A, C) = R(A, B), S(B, C), T(C, D), U(A, D).$$