Trade-Offs in Incremental View Maintenance

Dan Olteanu (University of Zurich)

fdbresearch.github.io

Logic & Algorithms in DB Theory and AI
August 25, 2023
Acknowledgments

DaST IVM team

Ahmet  Haozhe  Johann  Milos

RelationalAI colleagues

Hung  ELSeidy  Henrik  Niko
Setting & Objective of this Lecture

Incremental View Maintenance (IVM)

- Well-established and longstanding research problem
- Confusing naming: incremental vs decremental
  Alternative common naming: *Fully dynamic*
Incremental View Maintenance (IVM)

- Well-established and longstanding research problem
- Confusing naming: incremental vs decremental
  
  Alternative common naming: *Fully dynamic*

Setting

- Fully dynamic algorithms (i.e., supports inserts and deletes)
- Single-tuple updates to relational databases
- Relational queries (non-recursive)
Incremental View Maintenance (IVM)

- Well-established and longstanding research problem
- Confusing naming: incremental vs decremental
  
  Alternative common naming: *Fully dynamic*

**Setting**

- Fully dynamic algorithms (i.e., supports inserts and deletes)
- Single-tuple updates to relational databases
- Relational queries (non-recursive)

**Objective**

- Overview of recent (and *very preliminary*) results on worst-case optimal IVM, trade-offs, and IVM for complex analytics
We are interested in the trade-off between:

- preprocessing time
- enumeration delay
- update time
The Incremental View Maintenance Problem

Query + Database

preprocess

Data Structure

preprocessing time

User

access request

data structure

enumerate

Answer

We are interested in the trade-off between:

- preprocessing time
- enumeration delay
- update time
We are interested in the trade-off between:

- preprocessing time
- enumeration delay
- update time
The Incremental View Maintenance Problem

We are interested in the trade-off between:

- preprocessing time
- enumeration delay
- update time
The Incremental View Maintenance Problem

Query + Database

preprocess

Data Structure

preprocessing time

access request

User

enumerate

record #1
record #2
...

Answer

We are interested in the trade-off between:

preprocessing time - enumeration delay - update time
The Incremental View Maintenance Problem

We are interested in the trade-off between:

- Preprocessing time
- Enumeration delay
- Update time
We are interested in the trade-off between:

- preprocessing time
- enumeration delay
- update time
We are interested in the trade-off between:

- preprocessing time
- enumeration delay
- update time
The Incremental View Maintenance Problem

We are interested in the trade-off between:
- preprocessing time
- enumeration delay
- update time
The Incremental View Maintenance Problem

We are interested in the trade-off between:

- preprocessing time
- update time
- enumeration delay
We are interested in the trade-off between:
preprocessing time - enumeration delay - update time
Part 1. Main IVM techniques by example
- The triangle count query

Part 2. Constant update time and enumeration delay
- The $q$-hierarchical queries

Part 3. Update time - enumeration delay trade-offs
- The hierarchical queries and beyond

Part 4. ML models under updates
- Covariance matrix and Chow-Liu trees
1. IVM Techniques By Example
Relations are functions mapping tuples to elements from a ring (here, \( \mathbb{Z} \))

<table>
<thead>
<tr>
<th>( R )</th>
<th>( S )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( B )</td>
<td>#</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>2</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>3</td>
</tr>
<tr>
<td>( B )</td>
<td>( C )</td>
<td>#</td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( c_1 )</td>
<td>2</td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( c_2 )</td>
<td>1</td>
</tr>
<tr>
<td>( C )</td>
<td>( A )</td>
<td>#</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( a_1 )</td>
<td>1</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( a_1 )</td>
<td>3</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( a_2 )</td>
<td>3</td>
</tr>
</tbody>
</table>
Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, $\mathbb{Z}$)

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>$T$</th>
<th>$R \cdot S \cdot T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$4 + 6 + 9 = 19$

Triangle Count Query:
$\Delta \{ (a_1, b_1, c_1) \}$
Relations are functions mapping tuples to elements from a ring (here, $\mathbb{Z}$)

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>$T$</th>
<th>$R \cdot S \cdot T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
<td>$A$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$a_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_2$</td>
<td>$a_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b_1$</td>
<td>$a_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$c_1$</td>
<td>$a_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$c_2$</td>
<td>$a_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$c_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$a_2$</td>
</tr>
<tr>
<td>$b_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, \( \mathbb{Z} \))
- Triangle Count Query: \( Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \)

<table>
<thead>
<tr>
<th>( R )</th>
<th>( S )</th>
<th>( T )</th>
<th>( R \cdot S \cdot T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( B )</td>
<td>#</td>
<td>( A )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( c_1 )</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( c_2 )</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( c_1 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( c_2 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( c_2 )</td>
</tr>
</tbody>
</table>

\[ Q \]

\[ \emptyset \]

\[ ( ) | 4 + 6 + 9 = 19 \]
Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, \(\mathbb{Z}\))
- Triangle Count Query: \(Q = \sum_{a,b,c} R(a, b) \cdot S(b, c) \cdot T(c, a)\)
- A single-tuple update is a relation mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

### Relations

<table>
<thead>
<tr>
<th>(R)</th>
<th>(S)</th>
<th>(T)</th>
<th>(R \cdot S \cdot T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>(B)</td>
<td>(#)</td>
<td>(B)</td>
</tr>
<tr>
<td>(a_1)</td>
<td>(b_1)</td>
<td>2</td>
<td>(b_1)</td>
</tr>
<tr>
<td>(a_2)</td>
<td>(b_1)</td>
<td>3</td>
<td>(b_1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Updates

- \(\delta R = \{(a_2, b_1) \mapsto -2\}\)

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>(#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_2)</td>
<td>(b_1)</td>
<td>(-2)</td>
</tr>
</tbody>
</table>

- \(Q\)

<table>
<thead>
<tr>
<th>(\emptyset)</th>
<th>(#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\ ))</td>
<td>(4 + 6 + 9 = 19)</td>
</tr>
</tbody>
</table>
Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, \( \mathbb{Z} \))
- Triangle Count Query: \( Q = \sum_{a, b, c} R(a, b) \cdot S(b, c) \cdot T(c, a) \)
- A single-tuple update is a relation mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

<table>
<thead>
<tr>
<th>( R )</th>
<th>( S )</th>
<th>( T )</th>
<th>( R \cdot S \cdot T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( B )</td>
<td>#</td>
<td>( A )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>2</td>
<td>( b_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>3</td>
<td>( b_1 )</td>
</tr>
</tbody>
</table>

\( \delta R = \{(a_2, b_1) \mapsto -2\} \)

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>-2</td>
</tr>
</tbody>
</table>

\[ Q \]

<table>
<thead>
<tr>
<th>( \emptyset )</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>4 + 6 + 9 = 19</td>
</tr>
</tbody>
</table>
Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, \( \mathbb{Z} \))
- Triangle Count Query: \( Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \)
- A single-tuple update is a relation mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

\[
\begin{array}{c|c|c|c|c|c}
\hline
R & S & T & R \cdot S \cdot T \\
A & B & \# & B & C & \# & C & A & \# \\
\hline
a_1 & b_1 & 2 & b_1 & c_1 & 2 & c_1 & a_1 & 1 \\
\hline
a_2 & b_1 & 3 & b_1 & c_2 & 1 & c_2 & a_1 & 3 \\
\hline
a_2 & b_1 & 1 & c_2 & a_2 & 3 \\
\hline
\end{array}
\]

\[
\delta R = \{(a_2, b_1) \mapsto -2\}
\]

\[
\begin{array}{c|c|c|c|c|c}
\hline
A & B & \# & \hline
\emptyset & \# \\
\hline
a_2 & b_1 & -2 \\
\hline
( ) & 4 + 6 + 9 = 19
\end{array}
\]

\( Q \)
## Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, $\mathbb{Z}$)
- Triangle Count Query: $Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$
- A single-tuple update is a relation mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>$T$</th>
<th>$R \cdot S \cdot T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_2$</td>
<td>$c_2$</td>
</tr>
</tbody>
</table>

$\delta R = \{(a_2, b_1) \mapsto -2\}$

$Q$

| $\emptyset$ | $#$ |
| $()$ | $4 + 6 + 9 = 19$ |
### Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, \( \mathbb{Z} \)).

- Triangle Count Query: \( Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \).

- A single-tuple update is a relation mapping a tuple to a non-zero value (positive for insertions, negative for deletions).

\[
\begin{align*}
R & \quad S & \quad T & \quad R \cdot S \cdot T \\
A & B & \# & B & C & \# & C & A & \# \\
a_1 & b_1 & 2 & b_1 & c_1 & 2 & c_1 & a_1 & 1 \\
a_2 & b_1 & 3 & b_1 & c_2 & 1 & c_2 & a_1 & 3 \\
a_2 & b_1 & 1 & & & & c_2 & a_2 & 3 \\
\end{align*}
\]

\[Q = \emptyset \]

\[Q = ( ) \] 
\[4 + 6 + 9 = 19\]
Background: Relations and Queries

- Relations are functions mapping tuples to elements from a ring (here, $\mathbb{Z}$)
- Triangle Count Query: $Q = \sum_{a,b,c} R(a, b) \cdot S(b, c) \cdot T(c, a)$
- A single-tuple update is a relation mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>$T$</th>
<th>$R \cdot S \cdot T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
<td>$A$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>#</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$a_1$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>#</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$a_1$</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

$\delta R = \{(a_2, b_1) \mapsto -2\}$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>-2</td>
</tr>
</tbody>
</table>

$Q$

<table>
<thead>
<tr>
<th>$\emptyset$</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>4 + 6 + 9 = 19</td>
</tr>
<tr>
<td>( )</td>
<td>4 + 6 + 3 = 13</td>
</tr>
</tbody>
</table>
The Triangle Count Query

The triangle count query $Q$ returns the number of tuples in the join of $R$, $S$, and $T$:

$$Q = \sum_{a,b,c} R(a, b) \cdot S(b, c) \cdot T(c, a)$$

Problem: Maintain $Q$ under single-tuple updates to $R$, $S$, and $T$.
## Much Ado about Triangles

### The Triangle Query Served as Milestone in Many Fields

- Worst-case optimal join algorithms \([Algorithmica 1997, SIGMOD R. 2013]\)
- Parallel query evaluation \([Found. & Trends DB 2018]\)
- Randomized approximation in static settings \([FOCS 2015]\)
- Randomized approximation in data streams

### Answering Queries under Updates

- Theoretical developments \([PODS 2017, ICDT 2018]\)
- Lower bounds \([STOC 2015, ICM 2018]\)
Much Ado about Triangles

The Triangle Query Served as Milestone in Many Fields

- Worst-case optimal join algorithms [Algorithmica 1997, SIGMOD R. 2013]
- Parallel query evaluation [Found. & Trends DB 2018]
- Randomized approximation in static settings [FOCS 2015]

Answering Queries under Updates

- Theoretical developments [PODS 2017, ICDT 2018]
- Lower bounds [STOC 2015, ICM 2018]

Is there a **fully dynamic algorithm** that can maintain the **exact triangle count** in **worst-case optimal** time?
Naïve Maintenance

“Recompute from scratch”

\[ Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \]

\[ \delta R = \{ (\alpha,\beta) \mapsto m \} \]

N is the database size

Update time: \( O(N^{1.5}) \) using worst-case optimal join algorithms

[Algorithmica 1997, SIGMOD R. 2013, ICDT 2014]

Slightly better using Strassen-like matrix multiplication

Space: \( O(N) \) to store input relations
Naïve Maintenance

"Recompute from scratch"

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b,c} R(a, b) + \delta R(a, b) \]

\[ S(b, c) \]

\[ T(c, a) \]

N is the database size

Update time: \( O(N^{1.5}) \) using worst-case optimal join algorithms

[Algorithmica 1997, SIGMOD R. 2013, ICDT 2014]

Slightly better using Strassen-like matrix multiplication

Space: \( O(N) \) to store input relations
Naïve Maintenance

“Recompute from scratch”

\[ \delta R = \{(\alpha, \beta) \mapsto m\} \]

\[ Q = \sum_{a,b,c} R(a, b) + \delta R(a, b) \]

\[ \cdot \]

\[ S(b, c) \cdot \]

\[ T(c, a) \]
Naïve Maintenance

"Recompute from scratch"

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b,c} R(a, b) + \delta R(a, b) \]

\[ S(b, c) \] \[ T(c, a) \]

- \( N \) is the database size
- Update time: \( O(N^{1.5}) \) using worst-case optimal join algorithms
  
  \[ \text{[Algorithmica 1997, SIGMOD R. 2013, ICDT 2014]} \]
  
  Slightly better using Strassen-like matrix multiplication
- Space: \( O(N) \) to store input relations
First-Order Incremental View Maintenance

“Compute the delta”

$\delta Q = \delta R(\alpha, \beta) \cdot \sum_{c} S(\beta, c) \cdot T(c, \alpha)$

Update time: $O(N)$ to intersect $C$-values from $S$ and $T$

Space: $O(N)$ to store input relations
“Compute the delta”

\[ \delta R = \{(\alpha, \beta) \mapsto m\} \]

\[ Q = \sum_{a,b,c} \]

\[ \delta Q = \sum_{c} \]

\[ R(a,b) \quad S(b,c) \quad T(c,a) \]

\[ \delta R(\alpha, \beta) \quad S(\beta, c) \quad T(c, \alpha) \]

Update time: \( O(N) \) to intersect \( C \)-values from \( S \) and \( T \)

Space: \( O(N) \) to store input relations
First-Order Incremental View Maintenance

“Compute the delta”

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b,c} R(a, b) \cdot S(b, c) \cdot T(c, a) \]

\[ \delta Q = \sum_c \delta R(\alpha, \beta) \cdot S(\beta, c) \cdot T(c, \alpha) \]

Update time: \( O(N) \) to intersect \( C \)-values from \( S \) and \( T \)

Space: \( O(N) \) to store input relations
First-Order Incremental View Maintenance

“Compute the delta”

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \]

\[ \delta Q = \alpha \beta \sum_c \delta R(\alpha, \beta) \cdot S(\beta, c) \cdot T(c, \alpha) \]

Update time: \( O(N) \) to intersect \( C \)-values from \( S \) and \( T \)

Space: \( O(N) \) to store input relations

[Found. & Trends DB 2018]
First-Order Incremental View Maintenance

"Compute the delta"

\[ \delta R = \left\{ (\alpha, \beta) \mapsto m \right\} \]

\[ Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \]

\[ \delta Q = \alpha \beta \sum_{c} \delta R(\alpha, \beta) \]

Update time: \( O(N) \) to intersect \( C \)-values from \( S \) and \( T \)
Space: \( O(N) \) to store input relations
“Compute the delta”

\[
\delta R = \{(\alpha, \beta) \mapsto m\}
\]

\[
Q = \sum_{a,b,c} R(a,b)
\]

\[
\delta R(\alpha, \beta)
\]

\[
\delta Q = \alpha \beta \cdot \sum_c S(\beta, c) T(c, \alpha)
\]

\[
Q = Q + \delta Q
\]
First-Order Incremental View Maintenance

“Compute the delta”

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a) \]

\[ \delta R(\alpha, \beta) \]

\[ \delta Q = \alpha \beta \cdot \sum_{c} \cdot O(N) \cdot O(N) \cdot O(N) \]

\[ Q = Q + \delta Q \]

- Update time: \( O(N) \) to intersect \( C \)-values from \( S \) and \( T \)
- Space: \( O(N) \) to store input relations
"Compute the delta using materialized views"

\[ \delta R = \{ (\alpha, \beta) \mapsto \cdot \} \]

\[ Q = \sum_{a, b, c} R(a, b) \cdot S(b, c) \cdot T(c, a) \]

Time for updates to \( R \): \( O(1) \) to look up in \( VST \)
“Compute the delta using materialized views” [VLDB J 2014]

\[ \delta R = \{(\alpha, \beta) \mapsto m\} \]

\[ Q = \sum_{a, b, c} R(a, b) \cdot S(b, c) \cdot T(c, a) \]

\[ V_{ST}(b, a) = \sum_{c} S(b, c) \cdot T(c, a) \]
Higher-Order Incremental View Maintenance

"Compute the delta using materialized views" [VLDB J 2014]

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b} R(a, b) \cdot V_{ST}(b, a) \]

\[ \Delta Q = \sum_{a,b} \delta R(\alpha, \beta) \cdot V_{ST}(\beta, \alpha) \]

Time for updates to \( R \): \( O(1) \) to look up in \( V_{ST} \)
"Compute the delta using materialized views"

\[ \delta R = \{(\alpha, \beta) \mapsto m\} \]

\[
Q = \sum_{a, b} R(a, b)
\]

\[
\delta Q = \sum_{a, b} \delta R(a, b)
\]

\[
\delta Q = \sum_{\alpha, \beta} \delta R(\alpha, \beta)
\]

\[
Q = Q + \delta Q
\]

Time for updates to \( R \): \( O(1) \) to look up in \( V_{ST} \)
Higher-Order Incremental View Maintenance

“Compute the delta using materialized views”

$\delta R = \{ (\alpha, \beta) \mapsto m \}$

$Q = \sum_{a,b} R(a, b)$

$\delta Q = \sum_{a,b} \delta R(\alpha, \beta)$

$Q = Q + \delta Q$
Higher-Order Incremental View Maintenance

“Compute the delta using materialized views”

\[ \delta R = \{ (\alpha, \beta) \mapsto m \} \]

\[ Q = \sum_{a,b} R(a, b) \]  
\[ \delta Q = \sum_{a,b} \delta R(\alpha, \beta) \]  
\[ Q = Q + \delta Q \]
"Compute the delta using materialized views"

\[ \delta R = \{(\alpha, \beta) \mapsto m\} \]

\[ Q = \sum_{a,b} R(a, b) \]

\[ \delta Q = \sum_{a,b} \delta R(\alpha, \beta) \]

\[ Q = Q + \delta Q \]

- Time for updates to \( R \): \( O(1) \) to look up in \( V_{ST} \)
Higher-Order Incremental View Maintenance

Maintain $V_{ST}$ under updates

$$V_{ST}(b, a) = \sum_c S(b, c) \cdot T(c, a)$$
Maintain $V_{ST}$ under updates

$$\delta S = \{ (\beta, \gamma) \rightarrow m \}$$

$V_{ST}(b, a) = \sum_c S(b, c)$

$$\delta V_{ST}(\beta, a) = \delta S(\beta, \gamma) \cdot T(\gamma, a)$$

Time for updates to $S$ and $T$: $O(N)$ to maintain $V_{ST}$

Space: $O(N^2)$ to store input relations and $V_{ST}$
Maintain $V_{ST}$ under updates

$$\delta S = \{(\beta, \gamma) \mapsto m\}$$

$$V_{ST}(b, a) = \sum_c S(b, c) \cdot T(c, a)$$

$$\delta V_{ST}(\beta, a) = \delta S(\beta, \gamma) \cdot T(\gamma, a)$$
Maintain $V_{ST}$ under updates

$$\delta S = \{(\beta, \gamma) \mapsto m\}$$

$$V_{ST}(b, a) = \sum_c S(b, c) \cdot T(c, a)$$

$$\delta V_{ST}(\beta, a) = \delta S(\beta, \gamma) \cdot T(\gamma, a)$$

Time for updates to $S$ and $T$: $O(N)$ to maintain $V_{ST}$

Space: $O(N^2)$ to store input relations and $V_{ST}$
Higher-Order Incremental View Maintenance

Maintain $V_{ST}$ under updates

$$\delta S = \{(\beta, \gamma) \mapsto m\}$$

$$V_{ST}(b, a) = \sum_c S(b, c) \cdot T(c, a)$$

$$\delta V_{ST}(\beta, a) = \delta S(\beta, \gamma) \cdot T(\gamma, a)$$

$$V_{ST}(\beta, a) = V_{ST}(\beta, a) + \delta V_{ST}(\beta, a)$$

Time for updates to $S$ and $T$: $O(N)$ to maintain $V_{ST}$

Space: $O(N^2)$ to store input relations and $V_{ST}$
Maintain $V_{ST}$ under updates

$$
\delta S = \{(\beta, \gamma) \mapsto m\}
$$

$$
V_{ST}(b, a) = \sum_c S(b, c) \cdot \delta S(\beta, \gamma) \cdot T(c, a)
$$

$$
\delta V_{ST}(\beta, a) = \delta S(\beta, \gamma) \cdot T(\gamma, a)
$$

$$
V_{ST}(\beta, a) = V_{ST}(\beta, a) + \delta V_{ST}(\beta, a)
$$

- Time for updates to $S$ and $T$: $O(N)$ to maintain $V_{ST}$
- Space: $O(N^2)$ to store input relations and $V_{ST}$
Lower Bound for Maintaining the Triangle Count
Let $D$ be the database instance and $N$ the number of tuples in $D$. For any $\gamma > 0$, there is no algorithm that incrementally maintains $Q_b$ with update time enumeration delay $O(N^{1.2 - \gamma})$ unless the Online Vector-Matrix-Vector Multiplication (OuMv) Conjecture fails.
The Boolean Triangle Detection Problem

Boolean Triangle Detection Query

\[ Q_b = \bigvee_{a,b,c} R(a, b) \land S(b, c) \land T(c, a) \]

Let \( D \) be the database instance and \( N \) the number of tuples in \( D \).
For any \( \gamma > 0 \), there is no algorithm that incrementally maintains \( Q_b \) with

\[
\begin{align*}
\text{update time} & \quad \text{enumeration delay} \\
\mathcal{O}(N^{\frac{1}{2} - \gamma}) & \quad \mathcal{O}(N^{1-\gamma})
\end{align*}
\]

unless the Online Vector-Matrix-Vector Multiplication (OuMv) Conjecture fails.
The OuMv problem:

- **Input:** An $n \times n$ Boolean matrix $M$ and $n$ pairs $(u_1, v_1), \ldots, (u_n, v_n)$ of Boolean column-vectors of size $n$ arriving one after the other.
- **Goal:** After seeing each pair $(u_r, v_r)$, output $u_r^\top M v_r$.
Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- Input: An $n \times n$ Boolean matrix $M$ and $n$ pairs $(u_1, v_1), \ldots, (u_n, v_n)$ of Boolean column-vectors of size $n$ arriving one after the other.
- Goal: After seeing each pair $(u_r, v_r)$, output $u_r^T M v_r$

The OuMv Conjecture [STOC 2015]

For any $\gamma > 0$, there is no algorithm that solves OuMv in time $O(n^{3-\gamma})$. 

Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- **Input:** An $n \times n$ Boolean matrix $M$ and $n$ pairs $(u_1, v_1), \ldots, (u_n, v_n)$ of Boolean column-vectors of size $n$ arriving one after the other.
- **Goal:** After seeing each pair $(u_r, v_r)$, output $u_r^\top M v_r$

The OuMv Conjecture [STOC 2015]

For any $\gamma > 0$, there is no algorithm that solves OuMv in time $O(n^{3-\gamma})$.

The OuMv Conjecture is implied by the OMv Conjecture [STOC 2015]

The OMv problem:

- **Input:** An $n \times n$ Boolean matrix $M$ and $n$ Boolean column-vectors $v_1, \ldots, v_n$ of size $n$ arriving one after the other
- **Goal:** After seeing each vector $v_r$, output $M v_r$
Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- Input: An $n \times n$ Boolean matrix $M$ and $n$ pairs $(u_1, v_1), \ldots, (u_n, v_n)$ of Boolean column-vectors of size $n$ arriving one after the other.
- Goal: After seeing each pair $(u_r, v_r)$, output $u_r^\top M v_r$

The OuMv Conjecture [STOC 2015]

For any $\gamma > 0$, there is no algorithm that solves OuMv in time $O(n^{3-\gamma})$.

The OuMv Conjecture is implied by the OMv Conjecture [STOC 2015]

The OMv problem:

- Input: An $n \times n$ Boolean matrix $M$ and $n$ Boolean column-vectors $v_1, \ldots, v_n$ of size $n$ arriving one after the other
- Goal: After seeing each vector $v_r$, output $M v_r$

The OMv Conjecture

For any $\gamma > 0$, there is no algorithm that solves OMv in time $O(n^{3-\gamma})$. 
Proof Idea

■ Assume there is an algorithm $\mathcal{A}$ that can maintain Triangle Detection Query $Q_b$ with

\[
\begin{align*}
\text{amortized update time} & : \mathcal{O}(N^{1/2-\gamma}) \\
\text{enumeration delay} & : \mathcal{O}(N^{1-\gamma})
\end{align*}
\]

for some $\gamma > 0$.

■ We design an algorithm $\mathcal{B}$ that uses the oracle $\mathcal{A}$ to solve OuMv in subcubic time in $n$. $\implies$ Contradicts the OuMv Conjecture!
Proof Idea

- Assume there is an algorithm $A$ that can maintain Triangle Detection Query $Q_b$ with
  amortized update time $\mathcal{O}(N^{\frac{1}{2} - \gamma})$
  enumeration delay $\mathcal{O}(N^{1-\gamma})$
  for some $\gamma > 0$.

- We design an algorithm $B$ that uses the oracle $A$ to solve OuMv in subcubic time in $n$. $\implies$ **Contradicts the OuMv Conjecture!**

**Algorithm $B$**

- Relation $S$ encodes the matrix $M$: $S(i, j) = M[i, j]$

- In each round $r \in [n]$:  
  - Relation $R$ encodes the vector $u_r$: $R(a, i) = u_r[i]$, for constant $a$
  - Relation $T$ encodes the vector $v_r$: $T(j, a) = v_r[j]$, for constant $a$
  - Then $u_r^T M v_r = Q_b$
  - Check whether $Q_b = 1$ using algorithm $A$. 
Example Encoding for \( u, M, \) and \( v \)

\[
\begin{align*}
\mathbf{u}^\top & \quad \mathbf{M} & \quad \mathbf{v} \\
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{u}^\top \mathbf{M} \mathbf{v} & \quad \mathbf{u}^\top \mathbf{M} \mathbf{v} \\
1 & \quad 1
\end{align*}
\]

\[
\begin{array}{lll}
R & S & T \\
\hline
\begin{array}{lll}
A & B & \text{val} \\
a & 2 & 1
\end{array} & \begin{array}{lll}
B & C & \text{val} \\
2 & 1 & 1 \\
3 & 1 & 1 \\
1 & 2 & 1 \\
2 & 2 & 1 \\
3 & 3 & 1
\end{array} & \begin{array}{lll}
C & A & \text{val} \\
1 & a & 1 \\
\emptyset & \text{val} \\
\end{array}
\end{array}
\]

\[
\begin{array}{lll}
Q_b & \\
\hline
\emptyset & \text{val} \\
( & 1
\end{array}
\]
Proof Sketch: Algorithm $\mathcal{B}$

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]$  \hspace{1cm} (\leq n^2$ insertions$)$
Proof Sketch: Algorithm $B$

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]$ \hspace{1cm} ($\leq n^2$ insertions)

(2) In each round $r \in [n]$:
   - Delete all tuples in $R$ and $T$ \hspace{1cm} ($\leq 2n$ deletions)
Proof Sketch: Algorithm $\mathcal{B}$

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]$  \hspace{1cm} ($\leq n^2$ insertions)

(2) In each round $r \in [n]$:

- Delete all tuples in $R$ and $T$  \hspace{1cm} ($\leq 2n$ deletions)

- Insert into $R$ and $T$:
  For $i, j \in [n]$: $R(a, i) = u_r[i]$ and $T(j, a) = v_r[j]$  \hspace{1cm} ($\leq 2n$ insertions)
Proof Sketch: Algorithm $\mathcal{B}$

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]$ \hspace{1cm} (\leq n^2$ insertions$)

(2) In each round $r \in [n]$:
   - Delete all tuples in $R$ and $T$ \hspace{1cm} (\leq 2n$ deletions$)
   - Insert into $R$ and $T$:
     - For $i, j \in [n]$:
       - $R(a, i) = u_r[i] \quad$ and $\quad T(j, a) = v_r[j]$ \hspace{1cm} (\leq 2n$ insertions$)
   - Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$
     - $u_r^T M v_r = 1 \iff \exists i, j \in [n] : u_r[i] = 1, M[i, j] = 1, v_r[j] = 1$
Proof Sketch: Algorithm $\mathcal{B}$

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]$  
\hspace{1cm} ($\leq n^2$ insertions)

(2) In each round $r \in [n]$: 
\hspace{1cm} ▶ Delete all tuples in $R$ and $T$  
\hspace{1.5cm} ($\leq 2n$ deletions) 

\hspace{1cm} ▶ Insert into $R$ and $T$:  
\hspace{1.5cm} For $i, j \in [n]$: $R(a, i) = u_r[i]$ and $T(j, a) = v_r[j]$  
\hspace{1.5cm} ($\leq 2n$ insertions) 

\hspace{1cm} ▶ Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$  
\hspace{1.5cm} $u_r^T M v_r = 1 \Leftrightarrow \exists i, j \in [n]: u_r[i] = 1, M[i, j] = 1, v_r[j] = 1$

$\mathcal{B}$ constructs a database of size $N = \mathcal{O}(n^2)$. 
Proof Sketch: Time Analysis

Recall $\mathcal{A}$ needs $O((n^2)^{\frac{1}{2} - \gamma})$ update time and $O((n^2)^{1 - \gamma})$ delay

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]

(2) In each round $r \in [n]$
   ▶ Delete all tuples in $R$ and $T$
   ▶ Insert into $R$ and $T$: For $i, j \in [n]$: $R(a, i) = u_r[i]$ and $T(j, a) = v_r[j]$

   ▶ Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$
Proof Sketch: Time Analysis

Recall $A$ needs $O((n^2)^{\frac{1}{2}-\gamma})$ update time and $O((n^2)^{1-\gamma})$ delay

(1) For $i, j \in [n]$: $S(i, j) = M[i, j]$

$$O(\underbrace{\# \text{updates}}_{\text{update time}} \cdot \underbrace{n^2 \cdot (n^2)^{\frac{1}{2}-\gamma}}_{\text{update time}}) = O(n^{3-2\gamma})$$

(2) In each round $r \in [n]$:

- Delete all tuples in $R$ and $T$
- Insert into $R$ and $T$: For $i, j \in [n]$: $R(a, i) = u_r[i]$ and $T(j, a) = v_r[j]$

- Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$
Proof Sketch: Time Analysis

Recall $A$ needs $O((n^2)^{\frac{1}{2}-\gamma})$ update time and $O((n^2)^{1-\gamma})$ delay

(1) For $i,j \in [n]$: $S(i,j) = M[i,j]$

$$O(\sqrt{n^2 \cdot (n^2)^{\frac{1}{2}-\gamma}}) = O(n^{3-2\gamma})$$

(2) In each round $r \in [n]$

- Delete all tuples in $R$ and $T$
- Insert into $R$ and $T$: For $i,j \in [n]$: $R(a, i) = u_r[i]$ and $T(j, a) = v_r[j]$

$$O(4n \cdot (n^2)^{\frac{1}{2}-\gamma}) = O(n^{2-2\gamma})$$

- Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$
Proof Sketch: Time Analysis

Recall $\mathcal{A}$ needs $O((n^2)^{\frac{1}{2} - \gamma})$ update time and $O((n^2)^{1-\gamma})$ delay

(1) For $i,j \in [n]$: $S(i,j) = M[i,j]$

\[
O(\frac{n^2}{\text{#updates}} \cdot (n^2)^{\frac{1}{2} - \gamma}) = O(n^{3-2\gamma})
\]

(2) In each round $r \in [n]$:  
- Delete all tuples in $R$ and $T$
- Insert into $R$ and $T$: For $i,j \in [n]$: $R(a,i) = u_r[i]$ and $T(j,a) = v_r[j]$

\[
O(\frac{4n}{\text{#updates}} \cdot (n^2)^{\frac{1}{2} - \gamma}) = O(n^{2-2\gamma})
\]

- Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$

\[
O((n^2)^{1-\gamma}) = O(n^{2-2\gamma})
\]

\[\Rightarrow\] Contradicts OuMv Conjecture!
Recall $\mathcal{A}$ needs $O((n^2)^{\frac{1}{2} - \gamma})$ update time and $O((n^2)^{1 - \gamma})$ delay

1. For $i, j \in [n]$: $S(i, j) = M[i, j]$

\[
O\left(\sqrt{n^2} \cdot (n^2)^{\frac{1}{2} - \gamma}\right) = O(n^{3-2\gamma})
\]

(2) In each round $r \in [n]$:  
- Delete all tuples in $R$ and $T$
- Insert into $R$ and $T$: For $i, j \in [n]$: $R(a, i) = u_r[i]$ and $T(j, a) = v_r[j]$

\[
O(4n \cdot (n^2)^{\frac{1}{2} - \gamma}) = O(n^{2-2\gamma})
\]

- Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$

\[
O((n^2)^{1-\gamma}) = O(n^{2-2\gamma})
\]

For $n$ rounds: $O(n(n^{2-2\gamma} + n^{2-2\gamma})) = O(n^{3-2\gamma})$
Proof Sketch: Time Analysis

Recall $A$ needs $O((n^2)^{1/2-\gamma})$ update time and $O((n^2)^{1-\gamma})$ delay

(1) For $i,j \in [n]$: $S(i,j) = M[i,j]$

\[
O\left( \frac{n^2}{\text{#updates}} \cdot (n^2)^{1/2-\gamma} \right) = O(n^{3-2\gamma})
\]

(2) In each round $r \in [n]$: 
   - Delete all tuples in $R$ and $T$
   - Insert into $R$ and $T$: For $i,j \in [n]$: $R(a,i) = u_r[i]$ and $T(j,a) = v_r[j]$

\[
O\left( \frac{4n}{\text{#updates}} \cdot (n^2)^{1/2-\gamma} \right) = O(n^{2-2\gamma})
\]

- Check $Q_b = 1$: This holds if and only if $u_r^T M v_r = 1$

\[
O\left( (n^2)^{1-\gamma} \right) = O(n^{2-2\gamma})
\]

For $n$ rounds: $O(n(n^{2-2\gamma} + n^{2-2\gamma})) = O(n^{3-2\gamma})$

Overall time: $O(n^{3-2\gamma} + n^{3-2\gamma}) = O(n^{3-2\gamma}) \Rightarrow$ Contradicts OuMv Conjecture!
Closing the Complexity Gap
Closing the Complexity Gap

Complexity bounds for the maintenance of the triangle count

**Known Upper Bound**

- Update Time: $O(N)$
- Space: $O(N)$

**Known Lower Bound**

- Update time: not $O(N^{\frac{1}{2} - \gamma})$ for any $\gamma > 0$
- under the OuMv Conjecture
Closing the Complexity Gap

Complexity bounds for the maintenance of the triangle count

**Known Upper Bound**

Update Time: $O(N)$

Space: $O(N)$

Can the triangle count be maintained with sublinear update time?

**Known Lower Bound**

Update time: not $O(N^{\frac{1}{2}-\gamma})$ for any $\gamma > 0$

under the OuMv Conjecture
Closing the Complexity Gap

Complexity bounds for the maintenance of the triangle count

<table>
<thead>
<tr>
<th>Known Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Update Time:</strong></td>
</tr>
<tr>
<td><strong>Space:</strong></td>
</tr>
</tbody>
</table>

Can the triangle count be maintained with sublinear update time?  

Yes: IVM$^ε$  
Amortized update time:  
$O(N^{\frac{1}{2}})$  
This is worst-case optimal

<table>
<thead>
<tr>
<th>Known Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Update time:</strong></td>
</tr>
</tbody>
</table>
IVM$^\varepsilon$ Exhibits a Time-Space Tradeoff

Given $\varepsilon \in [0, 1]$, IVM$^\varepsilon$ maintains the triangle count with

- $O(N^{\max\{\varepsilon, 1-\varepsilon\}})$ amortized update time
- $O(N^{1+\min\{\varepsilon, 1-\varepsilon\}})$ space
- $O(N^{\frac{3}{2}})$ preprocessing time
- $O(1)$ answer time.

(Linear space possible with a slightly more involved argument)
Inside $\text{IVM}^\varepsilon$
Main Techniques used in IVM

- Compute the delta like in first-order IVM
- Materialize views like in higher-order IVM

**New ingredient:** Use adaptive processing based on data skew

⇒ Treat *heavy* values differently from *light* values
Heavy/Light Partitioning of Relations

Partition $R$ based on $A$ into

- a light part $R_L = \{ t \in R \mid |\sigma_{A=t} A| < N^\epsilon \}$,
- a heavy part $R_H = R \setminus R_L$.

**Derived Bounds**

- from light part:
  - for all $A$-values $a$, $|\sigma_{A=a} R_L| < N^\epsilon$

- from heavy part:
  - for all $A$-values $a$, $|\sigma_{A=a} R_H| \geq N^\epsilon$
  - and $|\pi_A R_H| \cdot N^\epsilon \leq N$
  - $\Rightarrow |\pi_A R_H| \leq N^{1-\epsilon}$
Heavy/Light Partitioning of Relations

Define partition $R$ based on $A$ into
- a light part $R_L = \{ t \in R \mid |\sigma_{A=t}.A| < N^\epsilon \}$,
- a heavy part $R_H = R \setminus R_L$.

**Derived Bounds**

- From light part:
  for all $A$-values $a$, $|\sigma_{A=a} R_L| < N^\epsilon$

- From heavy part:
  for all $A$-values $a$, $|\sigma_{A=a} R_H| \geq N^\epsilon$
  and $|\pi_A R_H| \cdot N^\epsilon \leq N$
Heavy/Light Partitioning of Relations

Likewise, partition

- $S = S_L \cup S_H$ based on $B$, and
- $T = T_L \cup T_H$ based on $C$!

$Q$ is the sum of skew-aware queries

$$Q = \sum_{a,b,c} R_U(a, b) \cdot S_V(b, c) \cdot T_W(c, a), \text{ for } U, V, W \in \{L, H\}.$$
Given an update $\delta R_* = \{(\alpha, \beta) \mapsto m\}$, compute the delta for each of the following skew-aware queries using a different strategy:

$$Q_{*LL} = \sum_{a,b,c} R_*(a, b) \cdot S_L(b, c) \cdot T_L(c, a)$$

$$Q_{*HH} = \sum_{a,b,c} R_*(a, b) \cdot S_H(b, c) \cdot T_H(c, a)$$

$$Q_{*LH} = \sum_{a,b,c} R_*(a, b) \cdot S_L(b, c) \cdot T_H(c, a)$$

$$Q_{*HL} = \sum_{a,b,c} R_*(a, b) \cdot S_H(b, c) \cdot T_L(c, a)$$
Adaptive Maintenance Strategy

Given an update \( \delta R_\ast = \{(\alpha, \beta) \mapsto m\} \), compute the delta for each of the following skew-aware queries using a different strategy:

\[
\delta Q_{\ast LL} = \delta R_\ast (\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)
\]

\[
\delta Q_{\ast HH} = \delta R_\ast (\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_H(c, \alpha)
\]

\[
\delta Q_{\ast LH} = \delta R_\ast (\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_H(c, \alpha)
\]

\[
\delta Q_{\ast HL} = \delta R_\ast (\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_L(c, \alpha)
\]
Adaptive Maintenance Strategy

\[ \delta Q_{*LL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha) \]
Adaptive Maintenance Strategy

$$
\delta Q_{*LL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)
$$

$$
\delta Q_{*LL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)
$$
Adaptive Maintenance Strategy

$$\delta Q_{*LL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)$$

$$\delta Q_{*LL} = \alpha \beta \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)$$

Update time: $O(N^\varepsilon)$ to intersect the lists of $C$-values from $S_L$ and $T_L$
Adaptive Maintenance Strategy

$$\delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_H(c, \alpha)$$
Adaptive Maintenance Strategy

\[ \delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_H(c, \alpha) \]

\[ \delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_c \beta \cdot c \cdot S_H(\beta, c) \cdot T_H(c, \alpha) \]

Update time: \( O(N^{1-\varepsilon}) \) to intersect the lists of \( C \)-values from \( S_H \) and \( T_H \).
Adaptive Maintenance Strategy

\[ \delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_H(c, \alpha) \]

Update time: \( \mathcal{O}(N^{1-\varepsilon}) \) to intersect the lists of \( C \)-values from \( S_H \) and \( T_H \)
\[ \delta Q_{*LH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_H(c, \alpha) \]
Adaptive Maintenance Strategy

$$\delta Q_{\ast LH} = \delta R_{\ast}(\alpha, \beta) \cdot \sum_{c} S_{L}(\beta, c) \cdot T_{H}(c, \alpha)$$
Adaptive Maintenance Strategy

$$\delta Q^{*}_{LH} = \delta R^{*}(\alpha, \beta) \cdot \sum_{c} S_{L}(\beta, c) \cdot T_{H}(c, \alpha)$$

$$\delta R^{*}(\alpha, \beta)$$

$$\delta Q^{*}_{LH} = \begin{array}{cc} \alpha & \beta \end{array} \cdot \sum_{c}$$

$$S_{L}(\beta, c)$$

$$T_{H}(c, \alpha)$$
Adaptive Maintenance Strategy

\[ \delta Q_{LH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_H(c, \alpha) \]

- \( \delta R_*(\alpha, \beta) \)
- \( S_L(\beta, c) \)
- \( T_H(c, \alpha) \)

Update time: \( O(N^{\min\{\varepsilon, 1-\varepsilon\}}) \) to intersect the lists of C-values from \( S_L \) and \( T_H \)
Adaptive Maintenance Strategy

$$\delta Q_{HL} = \delta R_{\star}(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_L(c, \alpha)$$

$$\delta Q_{HL} = \sum_c \delta R_{\star}(\alpha, \beta) \cdot S_H(\beta, c) \cdot T_L(c, \alpha)$$

$$V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a)$$
Adaptive Maintenance Strategy

\[ \delta Q_{*HL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_L(c, \alpha) \]
Adaptive Maintenance Strategy

\[ \delta Q_{*HL} = \delta R_{*}(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_L(c, \alpha) \]
Adaptive Maintenance Strategy

$$\delta Q_{*HL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_L(c, \alpha)$$

Update time: $O(1)$ to look up in $V_{ST}$, assuming $V_{ST}$ is already materialized
## Summary of Adaptive Maintenance Strategies

Maintenance for an update $\delta R_* = \{(\alpha, \beta) \mapsto m\}$:

<table>
<thead>
<tr>
<th>Skew-aware View</th>
<th>Evaluation from left to right</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{a,b,c} R_*(a, b) \cdot S_L(b, c) \cdot T_L(c, a)$</td>
<td>$\delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)$</td>
<td>$\mathcal{O}(N^\varepsilon)$</td>
</tr>
<tr>
<td>$\sum_{a,b,c} R_*(a, b) \cdot S_H(b, c) \cdot T_H(c, a)$</td>
<td>$\delta R_*(\alpha, \beta) \cdot \sum_c T_H(c, \alpha) \cdot S_H(\beta, c)$</td>
<td>$\mathcal{O}(N^{1-\varepsilon})$</td>
</tr>
<tr>
<td>$\sum_{a,b,c} R_*(a, b) \cdot S_L(b, c) \cdot T_H(c, a)$</td>
<td>$\delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_H(c, \alpha)$</td>
<td>$\mathcal{O}(N^\varepsilon)$</td>
</tr>
<tr>
<td>or $\delta R_*(\alpha, \beta) \cdot \sum_c T_H(c, \alpha) \cdot S_L(\beta, c)$</td>
<td>$\mathcal{O}(N^{1-\varepsilon})$</td>
<td></td>
</tr>
<tr>
<td>$\sum_{a,b,c} R_*(a, b) \cdot S_H(b, c) \cdot T_L(c, a)$</td>
<td>$\delta R_*(\alpha, \beta) \cdot V_{ST}(\beta, \alpha)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
</tbody>
</table>

Overall update time: $\mathcal{O}(N^{\max(\varepsilon,1-\varepsilon)})$
Auxiliary Materialized Views

\[ V_{RS}(a, c) = \sum_b R_H(a, b) \cdot S_L(b, c) \]

\[ V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a) \]

\[ V_{TR}(a, c) = \sum_a T_H(c, a) \cdot R_L(a, b) \]
Maintain $V_{ST}(b, a) = \sum_{c} S_H(b, c) \cdot T_L(c, a)$ under update $\delta S_H = \{(\beta, \gamma) \mapsto m\}$

$\delta V_{ST}(\beta, a) = \delta S_H(\beta, \gamma) \cdot T_L(\gamma, a)$

Update time: $O(N\varepsilon)$ to iterate over $a$-values paired with $\gamma$ from $T_L$
Maintain \( V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a) \) under update \( \delta S_H = \{(\beta, \gamma) \mapsto m\} \)

\[
\delta V_{ST}(\beta, a) = \langle \delta S_H(\beta, \gamma) \cdot T_L(\gamma, a) \rangle < N^\varepsilon
\]
Maintain $V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a)$ under update $\delta S_H = \{((\beta, \gamma) \mapsto m)\}$

$$\delta V_{ST}(\beta, a) = \delta S_H(\beta, \gamma) \cdot T_L(\gamma, a) < N^\varepsilon$$

Update time: $\mathcal{O}(N^\varepsilon)$ to iterate over $a$-values paired with $\gamma$ from $T_L$
Maintain $V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a)$ under update $\delta T_L = \{(\gamma, \alpha) \mapsto m\}$

$\delta V_{ST}(b, \alpha) = \delta T_L(\gamma, \alpha) \cdot S_H(b, \gamma)$
Maintenance of Auxiliary Views

Maintain $V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a)$ under update $\delta T_L = \{(\gamma, \alpha) \mapsto m\}$

$$\delta V_{ST}(b, \alpha) = \delta T_L(\gamma, \alpha) \cdot S_H(b, \gamma)$$

$N^{1-\varepsilon} \geq$
Maintenance of Auxiliary Views

Maintain $V_{ST}(b, a) = \sum_c S_H(b, c) \cdot T_L(c, a)$ under update $\delta T_L = \{(\gamma, \alpha) \mapsto m\}$

$\delta V_{ST}(b, \alpha) = \delta T_L(\gamma, \alpha) \cdot S_H(b, \gamma)$

Update time: $\mathcal{O}(N^{1-\varepsilon})$ to iterate over $b$-values paired with $\gamma$ from $S_H$
Maintenance of Auxiliary Views: Summary

\[ V_{RS}(a, c) = \sum_{b} R_H(a, b) \cdot S_L(b, c) \]

\[ V_{ST}(b, a) = \sum_{c} S_H(b, c) \cdot T_L(c, a) \]

\[ V_{TR}(a, c) = \sum_{a} T_H(c, a) \cdot R_L(a, b) \]

**Maintenance Complexity**

- **Time:** \( \mathcal{O}(N^{\max\{\varepsilon, 1-\varepsilon\}}) \)
- **Space:** \( \mathcal{O}(N^{1+\min\{\varepsilon, 1-\varepsilon\}}) \)
Updates can change frequencies of values & heavy/light threshold
Rebalancing Partitions

Updates can change the frequencies of values in the relation parts

---

$R_L$

Insertions

Threshold

$a$ is light

$a$ is heavy

---

Minor Rebalancing

- Transfer $O(N^\varepsilon)$ tuples from one to the other part of the same relation
- Time complexity: $O(N^{\varepsilon + \max\{\varepsilon, 1-\varepsilon\}})$
Rebalancing Partitions

Updates can change the heavy-light threshold!

\[ R_H \]

Database size increases

Threshold

\[ a \quad \cdots \quad b \]

\[ b \]

\[ a \quad \cdots \quad b \]

\[ a \]

\[ b \]

\[ a \]

\[ b \]

Threshold

\[ a \quad \cdots \quad b \]

\[ b \]

\[ a \quad \cdots \quad b \]

\[ a \]

\[ b \]

\[ a \]

\[ b \]

\[ a \]

\[ b \]

Major Rebalancing

- Recompute partitions and views from scratch

- Time complexity: \( O(N^{1+\max\{\varepsilon, 1-\varepsilon\}}) \)
Amortization of Rebalancing Times

- Both forms of rebalancing require superlinear time
Amortization of Rebalancing Times

- Both forms of rebalancing require superlinear time
- The rebalancing times amortize over sequences of updates
  - Amortized minor rebalancing time: $\mathcal{O}(N^{\max\{\varepsilon,1-\varepsilon\}})$
  - Amortized major rebalancing time: $\mathcal{O}(N^{\max\{\varepsilon,1-\varepsilon\}})$
Amortization of Rebalancing Times

- Both forms of rebalancing require superlinear time
- The rebalancing times amortize over sequences of updates
  - Amortized minor rebalancing time: $O(N^{\max\{\varepsilon, 1-\varepsilon\}})$
  - Amortized major rebalancing time: $O(N^{\max\{\varepsilon, 1-\varepsilon\}})$
- Overall amortized rebalancing time: $O(N^{\max\{\varepsilon, 1-\varepsilon\}})$
Follow-up work & Open Questions

Follow-up work

- **TODS 2020**
  - Triangle queries with different free variables
  - Strong and weak Pareto optimality

- **APOCS 2021**
  - Extend the triangle counting algorithm to $k$-clique counting
  - Parallel batch-dynamic triangle count algorithm based on the (sequential single-tuple dynamic) triangle count algorithm

- **ICDT 2021**
  - Update time-approximation quality trade-off for triangle counting
  - Complexity of triangle counting based on the arboricity of the data graph
Follow-up Work & Open Questions

Follow-up work

- **TODS 2020**
  - Triangle queries with different free variables
  - Strong and weak Pareto optimality

- **APOCS 2021**
  - Extend the triangle counting algorithm to $k$-clique counting
  - Parallel batch-dynamic triangle count algorithm based on the (sequential single-tuple dynamic) triangle count algorithm

- **ICDT 2021**
  - Update time-approximation quality trade-off for triangle counting
  - Complexity of triangle counting based on the arboricity of the data graph

Open questions

- Worst-case optimal (and beyond) maintenance and the update-space trade-off for functional aggregate queries
- Single-tuple updates versus batch updates


[PODS 2017] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. *Answering Conjunctive Queries Under Updates.*


[ICDT 2018] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs Under Updates and in the Presence of Integrity Constraints.


[SIGMOD 2018] Nikolic, Milos, and Dan Olteanu. Incremental view maintenance with triple lock factorization benefits.


2. Constant Update Time & Enumeration Delay
Q: Which queries admit constant update time and enumeration delay in the worst-case?
Queries with Constant Update Time & Delay

Q: Which queries admit constant update time and enumeration delay in the worst-case?

A: Q-hierarchical queries [PODS 2017]

A: Queries that become q-hierarchical under functional dependencies [ICDE 2009, VLDBJ 2023, RelationalAI]

A: Queries with free access patterns Q(out | in) whose fractures are q-hierarchical [ICDT 2023]

A: Queries that become q-hierarchical under rewritings using q-hierarchical views and specific enumeration order [UZH 2023]
Q: Which queries admit constant update time and enumeration delay in the worst-case?

A: Q-hierarchical queries [PODS 2017]

A: Queries that become q-hierarchical under functional dependencies [ICDE 2009, VLDBJ 2023, RelationalAI]
Q: Which queries admit constant update time and enumeration delay in the worst-case?

A: \(Q\)-hierarchical queries \[PODS 2017\]

A: Queries that become \(q\)-hierarchical under functional dependencies \[ICDE 2009, VLDBJ 2023, RelationalAI\]

A: Queries with free access patterns \(Q(out|in)\) whose fractures are \(q\)-hierarchical \[ICDT 2023\]
Q: Which queries admit constant update time and enumeration delay in the worst-case?

A: Q-hierarchical queries [PODS 2017]

A: Queries that become $q$-hierarchical under functional dependencies [ICDE 2009, VLDBJ 2023, RelationalAI]

A: Queries with free access patterns $Q(out|in)$ whose fractures are $q$-hierarchical [ICDT 2023]

A: Queries that become $q$-hierarchical under rewritings using $q$-hierarchical views and specific enumeration order [UZH 2023]
A query is **hierarchical** if for any two variables, their sets of atoms in the query are either disjoint or one is contained in the other.

\[ Q(b, d) = \sum_{a,c,e,f} R(a, b, d) \cdot S(a, b) \cdot T(a, c, e) \cdot U(a, c, f) \]

[VLDB 2004]
Hierarchical Queries

A query is **hierarchical** if for any two variables, their sets of atoms in the query are either disjoint or one is contained in the other.

\[ Q(b, d) = \sum_{a, c, e, f} R(a, b, d) \cdot S(a, b) \cdot T(a, c, e) \cdot U(a, c, f) \]

**not hierarchical**

\[ Q(a, b) = R(a) \cdot S(a, b) \cdot T(b) \]
A query is \( q \)-hierarchical if it is hierarchical and the free variables dominate the bound variables.

\[
q \text{-hierarchical} \quad Q(a, b, c) = \sum_{d,e,f} R(a, b, d) \cdot S(a, b) \cdot T(a, c, e) \cdot U(a, c, f)
\]

[PODS 2017]
A query is *q*-hierarchical if it is hierarchical and the free variables dominate the bound variables.

\[
q\text{-hierarchical} \\
Q(a, b, c) = \sum_{d,e,f} R(a, b, d) \cdot S(a, b) \cdot T(a, c, e) \cdot U(a, c, f)
\]

hierarchical but not *q*-hierarchical

\[
Q(a) = \sum_{b} S(a, b) \cdot T(b)
\]
Let $Q$ be any conjunctive query without self-joins and $D$ a database.

- If $Q$ is \textbf{q-hierarchical}, then the query answer admits $O(1)$ single-tuple updates and enumeration delay.

- If $Q$ is \textbf{not q-hierarchical}, then there is no algorithm with $O(|D|^{1/2-\gamma})$ update time and enumeration delay for any $\gamma > 0$, unless the OMv conjecture fails.

[PODS 2017]
Rewriting queries under functional dependencies [ICDE 2009]

- Given: Query $Q$ and set $\Sigma$ of functional dependencies

- Replace the set of variables of each atom in $Q$ by its closure under $\Sigma$ called $\Sigma$-reduct

  Under $\Sigma = \{x \rightarrow y, y \rightarrow z\}$, the closure of $\{x\}$ is $\{x, y, z\}$

- If the $\Sigma$-reduct is $q$-hierarchical, then $Q$ admits constant update time and enumeration delay [VLDB J 2023]
Maintenance of Q-Hierarchical Queries

How to achieve constant update time and enumeration delay?

Recipe: [PODS 2017]

- Construct a factorized representation of the query answer [ICDT 2012]
- Such factorizations admit constant-delay enumeration
- Apply updates directly on the factorization

F-IVM system [https://github.com/fdbresearch/FIVM] [SIGMOD 2018]

- Factorize the query answer as a tree of views
- Materialize the views to speed up updates and enumeration
Example: Query Rewriting

\[ Q(w, x, y, z) = R(w, x) \cdot S(x, y) \cdot T(y, z) \]

Assume the functional dependencies: \( X \rightarrow Y \) and \( Y \rightarrow Z \)

\( Q \) is not \( q \)-hierarchical, but its rewriting under FDs is:

\[ Q'(w, x, y, z) = R'(w, x, y, z) \cdot S'(x, y, z) \cdot T'(y, z) \]
Example: Variable Order

\[ Q'(w, x, y, z) = R'(w, x, y, z) \cdot S'(x, y, z) \cdot T'(y, z) \]

Top-down construction of variable order for \( Q' \):
- \( Z \) and \( Y \) are first as they dominate \( X \) and \( W \)
- Then \( X \), which dominates \( W \)
- Finally \( W \)

We use this variable order also for \( Q \)
Example: View Tree

View tree construction:

- Place relations at leaves
- Create parent view to join children

\[ V'_Z(z, y) = T(y, z) \cdot V_X(y) \]
\[ V'_X(y, x) = S(x, y) \cdot V_W(x) \]
- Aggregate away variables not needed for further joins

\[ V_Z() = \sum_z V_Y(z) \]
\[ V_Y(z) = \sum_y V'_Z(y, z) \]
\[ V_X(y) = \sum_x V'_X(x, y) \]
\[ V_W(x) = \sum_w R'(x, w) \]
Example: Single-Tuple Update to $T$

$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V'_X(y, x)$

$V_W(x)$

$R(x, w)$

$S(x, y)$

Single-tuple update to $T$

$\delta V'_Z(z_0, y_0) = \delta T(y_0, z_0) \cdot V_X(y_0)$

$\delta V'_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)$
Example: Single-Tuple Update to $T$

$$V_Z()$$

$$V_Y(z)$$

$$V_Y'(z, y)$$

$$V_Z'(z, y)$$

$$V_X(y)$$

$$V_X'(y, x)$$

$$V_X'(y, x)$$

$$V_W(x)$$

$$R(x, w)$$

$$S(x, y)$$

Single-tuple update to $T$

$$\delta T(y_0, z_0)$$

$$\delta V_Z'(z_0, y_0) = \delta V_Y(z_0) = \delta V_Z(z_0)$$

For each updated view/relation $A$:

$$A := A + \delta A$$

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $T$

$$
\delta V'_{Z}(z_0, y_0) = \delta T(y_0, z_0) \cdot V_X(y_0)
$$
Example: Single-Tuple Update to $T$

Single-tuple update to $T$

\[
\delta V'_Z(z_0, y_0) = \delta T(y_0, z_0) \cdot V_X(y_0)
\]

\[
\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)
\]

For each updated view/relation $A$:

\[
A := A + \delta A
\]

Each view update takes $O(1)$ time.
Example: Single-Tuple Update to $T$

Single-tuple update to $T$

$$
\delta V'_Z(z_0, y_0) = \delta T(y_0, z_0) \cdot V_X(y_0)
$$

$$
\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)
$$

$$
\delta V'_Z(z_0, y_0) = \delta V'_Y(z_0) = \delta V_Y(z_0)
$$

For each updated view/relation $A$: $A := A + \delta A$

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $S$

For each updated view/relation $A$:

$$A := A + \delta A$$

Each view update takes $O(1)$ time.
Example: Single-Tuple Update to $S$

Single-tuple update to $S$

$$V_Z()$$

$$V_Y(z)$$

$$V_Z'(z, y)$$

$$V_X(y)$$

$$T(y, z)$$

$$V_X'(y, x)$$

$$V_W(x)$$

$$R(x, w)$$

$$\delta S(x_0, y_0)$$

For each updated view/relation $A$:

$$A := A + \delta A$$

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $S$

$V_Z()$

$V_Y(z)$

$V_Y'(z, y)$

$V_Z'(z, y)$

$V_X(y)$

$T(y, z)$

$\delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0)$

$\delta S(x_0, y_0)$

For each updated view/relation $A$:

$A := A + \delta A$

Each view update takes $O(1)$ time
**Example: Single-Tuple Update to $S$**

For each updated view/relation $A$:

$$A := A + \delta A$$

Each view update takes $O(1)$ time.

**Single-tuple update to $S$**

$$\delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0)$$

$$\delta V'_X(y_0) = \sum_{x_0} \delta V'_X(y_0, x_0) = \delta V'_X(y_0, x_0)$$

$$\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)$$

$$\delta V_Z() = \delta S(x_0, y_0)$$

For each updated view/relation $A$:
Example: Single-Tuple Update to $S$

Single-tuple update to $S$

\[
\delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0)
\]

\[
\delta V_X(y_0) = \sum_{x_0} \delta V'_X(y_0, x_0) = \delta V'_X(y_0, x_0)
\]

\[
\delta V'_Z(z_0, y_0) : \delta V'_X(y_0) \cdot T(y_0, z)^{y \rightarrow z} = \delta V'_X(y_0) \cdot T(y_0, z_0)
\]
**Example: Single-Tuple Update to S**

Single-tuple update to $S$

\[
\delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0)
\]

\[
\delta V_X(y_0) = \sum_{x_0} \delta V'_X(y_0, x_0) = \delta V'_X(y_0, x_0)
\]

\[
\delta V'_Z(z_0, y_0) : \delta V'_X(y_0) \cdot T(y_0, z) \rightleftarrows \delta V'_X(y_0) \cdot T(y_0, z_0)
\]

\[
\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)
\]

For each updated view/relation $A$:

\[
A := A + \delta A
\]

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $S$

Single-tuple update to $S$

\[ \delta V_Z() \]
\[ \delta V_Y(z_0) \]
\[ \delta V_Z'(z_0, y_0) \]
\[ \delta V_X(y_0) \]
\[ \delta V_X'(y_0, x_0) \]
\[ V_W(x) \]
\[ R(x, w) \]
\[ \delta S(x_0, y_0) \]

\[ \delta V_Z'(z_0, y_0) : \delta V_X'(y_0) \cdot T(y_0, z) \overset{\gamma \rightarrow z}{\Longrightarrow} \delta V_X'(y_0) \cdot T(y_0, z_0) \]

\[ \delta V_Y(z_0) = \sum_{y_0} \delta V_Z'(z_0, y_0) = \delta V_Z'(z_0, y_0) \]

\[ \delta V_X(y_0) = \sum_{x_0} \delta V_X'(y_0, x_0) = \delta V_X'(y_0, x_0) \]

\[ \delta V_Z() = \sum_{z_0} \delta V_Y(z_0) = \delta V_Y(z_0) \]

\[ \delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0) \]

\[ \delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0) \]

For each updated view/relation $A$: $A := A + \delta A$

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $R$

Single-tuple update to $R$

$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V'_X(y, x)$

$V_W(x)$

$R(x, w)$

$S(x, y)$
Single-tuple update to $R$

For each updated view/relation $A$:

\[ A = A + \delta A \]

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $R$

Single-tuple update to $R$

$$\delta V_{W}(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)$$
Example: Single-Tuple Update to $R$

Single-tuple update to $R$

$$
\delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)
$$

$$
\delta V'_X(y_0, x_0) : \delta V_W(x_0) \cdot S(x_0, y) \xrightarrow{x \rightarrow y} \delta V_W(x_0) \cdot S(x_0, y_0)
$$

For each updated view/relation $A$:

$A := A + \delta A$

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $R$

Single-tuple update to $R$

\[ \delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0) \]

\[ \delta V'_X(y_0, x_0) : \delta V_W(x_0) \cdot S(x_0, y) \xrightarrow{\text{equiv}} \delta V_W(x_0) \cdot S(x_0, y_0) \]

\[ \delta V_X(y_0) = \sum_{x_0} \delta V'_X(y_0, x_0) = \delta V'_X(y_0, x_0) \]

For each updated view/relation $A$:

\[ A := A + \delta A \]

Each view update takes $O(1)$ time.
Example: Single-Tuple Update to $R$

Single-tuple update to $R$

\[
\delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)
\]

\[\delta V'_Z(z_0, y_0) : \delta V_W(x_0) \cdot S(x_0, y) \xrightarrow{x \leftrightarrow y} \delta V_W(x_0) \cdot S(x_0, y_0)\]

\[\delta V'_X(y_0) : \delta V'_X(y_0) \cdot T(y_0, z) \xrightarrow{y \leftrightarrow z} \delta V'_X(y_0) \cdot T(y_0, z_0)\]

For each updated view/relation $A$:

\[A := A + \delta A\]

Each view update takes $O(1)$ time
Single-tuple update to $R$

$$\delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)$$

$$\delta V'_X(y_0, x_0) : \delta V_W(x_0) \cdot S(x_0, y) \xrightarrow{y \to y} \delta V_W(x_0) \cdot S(x_0, y_0)$$

$$\delta V_X(y_0) = \sum_{x_0} \delta V'_X(y_0, x_0) = \delta V'_X(y_0, x_0)$$

$$\delta V'_Z(z_0, y_0) : \delta V'_X(y_0) \cdot T(y_0, z) \xrightarrow{y \to z} \delta V'_X(y_0) \cdot T(y_0, z_0)$$

$$\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)$$

For each updated view/relation $A$:

$$A := A + \delta A$$

Each view update takes $O(1)$ time
Example: Single-Tuple Update to $R$

Single-tuple update to $R$

- $\delta V_Z()$
  - $\delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)$
  - $\delta V'_X(y_0, x_0) = \delta V'_Z(z_0, y_0) \cdot T(y_0, z_0) = T(y_0, z_0) \delta V'_X(y_0, x_0)$
  - $\delta V'_Y(z_0) = \delta V'_Z(z_0, y_0) \cdot T(y_0, z_0) = T(y_0, z_0) \delta V'_Z(z_0, y_0)$
- $\delta V_Y(z_0)$
  - $\delta V'_Y(z_0) = \delta V'_Z(z_0, y_0) \cdot T(y_0, z_0) = T(y_0, z_0) \delta V'_Z(z_0, y_0)$
  - $\delta V_Z() = \sum_{z_0} \delta V'_Y(z_0) = \delta V'_Y(z_0)$

For each updated view/relation $A$: $A := A + \delta A$

Each view update takes $O(1)$ time
Example: Enumeration of Query Answers

Enumeration for $Q(z, y, x, w)$ with constant delay

- Top-down in the view tree
- Views calibrated for variables underneath
- Guaranteed to get matching tuples in views below

Diagram:

- $V_Z()$
- $V_Y(z)$
- $V'_Z(y, z)$
- $V_X(y)$
- $T(y, z)$
- $V'_X(x, y)$
- $V_W(x)$
- $R(x, w)$
- $S(x, y)$
Example: Enumeration of Query Answers

Enumeration for $Q(z, y, x, w)$ with constant delay

- Top-down in the view tree
- Views calibrated for variables underneath
- Guaranteed to get matching tuples in views below

Enumeration from the join:

$$1_{V_Z} \cdot 1_{V_Y(z)} \cdot 1_{V'_Z(z, y)} \cdot 1_{V'_X(y, x)} \cdot T(z, y) \cdot S(x, y) \cdot R(x, w)$$

with variable order: $Z - Y - X - W$
Example: Enumeration of Query Answers

Enumeration for $Q(z, y, x, w)$ with constant delay

- Top-down in the view tree
- Views calibrated for variables underneath
- Guaranteed to get matching tuples in views below

Enumeration from the join:

$$1_{V_Z} \cdot 1_{V_Y(z)} \cdot 1_{V'_Z(z, y)} \cdot 1_{V'_X(y, x)} \cdot T(z, y) \cdot S(x, y) \cdot R(x, w)$$

with variable order: $Z - Y - X - W$

- Is $V_Z()$ empty? If yes, stop.
- Iterate over $z$’s in $V_Y(z)$
- For each $z$, iterate over $y$’s in index $V'_Z(z, y)$
- For each $y$, iterate over $x$’s in index $V'_X(y, x)$
- Iterate over $T(z, y), S(x, y), R(x, w)$
Open Questions

Can we achieve worst-case optimality per single-tuple update beyond the $q$-hierarchical queries?
Open Questions

- Can we achieve worst-case optimality per single-tuple update beyond the $q$-hierarchical queries?

- In practice, *average* constant time might be enough.

Which queries admit average constant time for single-tuple updates?
Open Questions

- Can we achieve worst-case optimality per single-tuple update beyond the $q$-hierarchical queries?

- In practice, *average* constant time might be enough.

Which queries admit average constant time for single-tuple updates?

- What is the complexity trade-off between update time and enumeration delay if we drop:
  
  - the "$q$" property?
  
  - the hierarchical property?


[PODS 2017] Christoph Berkholz, Jens Keppeler, Nicole Schweikardt. Answering Conjunctive Queries under Updates.


[UZH 2023] Johann Schwabe. *CaVieR: CAscading VIEw tRees.* MSc thesis, University of Zurich
3. Beyond “Q”
Simplest Hierarchical Query without “Q” Property

\[ Q(a) = \sum_{b} R(a, b) \cdot S(b) \]
Simplest Hierarchical Query without “Q” Property

\[ Q(a) = \sum_b R(a, b) \cdot S(b) \]

Lower bound

For this query, there is no algorithm that admits

- preprocessing time arbitrary
- update time \( \mathcal{O}(N^{1/2-\gamma}) \)
- enumeration delay \( \mathcal{O}(N^{1/2-\gamma}) \)

for any \( \gamma > 0 \), unless the OMv Conjecture fails

[PODS 2017]
Simplest Hierarchical Query without “Q” Property

\[
Q(a) = \sum_b R(a, b) \cdot S(b)
\]

Known approach: **Eager** update, quick enumeration

- **Preprocessing**: Materialize the result.
- **Upon update**: Maintain the materialized result.
- **Enumeration**: Enumerate from materialized result.

Lower bound

For this query, there is no algorithm that admits

\[O(N^{1/2 - \gamma})\]

for any \(\gamma > 0\), unless the OMv Conjecture fails [PODS 2017]

Yet, there is an algorithm that admits

sub-linear update time and sub-linear enumeration delay

**Weak Pareto optimality**
Simplest Hierarchical Query without “Q” Property

\[ Q(a) = \sum_b R(a, b) \cdot S(b) \]

Known approach: Lazy update, heavy enumeration

- Preprocessing: Eliminate dangling tuples
- Upon update: Update only base relations
- Enumeration: Eliminate dangling tuples and enumerate from \( R \)
Simplest Hierarchical Query without “Q” Property

\[ Q(a) = \sum_b R(a, b) \cdot S(b) \]

\[
\log_N \text{delay} \\
\log_N \text{preprocessing time} \\
\log_N \text{update time}
\]

\[
\delta = \frac{1}{2}
\]

\[(1, 0, 1) \]

\[
(1, 0, 1/2)
\]

Yet, there is an algorithm that admits

sub-linear update time and sub-linear enumeration delay
Simplest Hierarchical Query without “Q” Property

$$Q(a) = \sum_b R(a, b) \cdot S(b)$$

known approach: Eager update, quick enumeration
- Preprocessing: Materialize the result.
- Upon update: Maintain the materialized result.
- Enumeration: Enumerate from materialized result.

known approach: Lazy update, heavy enumeration
- Preprocessing: Eliminate dangling tuples
- Upon update: Update only base relations
- Enumeration: Eliminate dangling tuples and enumerate from $R$

lower bound
- For this query, there is no algorithm that admits preprocessing time, update time, enumeration delay in $O(N^{1/2-\gamma})$ for any $\gamma > 0$, unless the OMv Conjecture fails [PODS 2017]
- Yet, there is an algorithm that admits sub-linear update time and sub-linear enumeration delay

weak pareto optimality
Relation Partitioning

$$Q(a) = \sum_b R(a, b) \cdot S(b)$$

Partition $R$ based on the values $b$ into

- a **light part** $R^L = \{(a, b) \in R \mid |\sigma_{B=b}R| < N^\varepsilon\}$
- a **heavy part** $R^H = R - R^L$
Relation Partitioning

\[ Q(a) = \sum_b R(a, b) \cdot S(b) \]

Partition \( R \) based on the values \( b \) into

- a light part \( R^L = \{ (a, b) \in R \mid |\sigma_{B=b} R| < N^\varepsilon \} \)
- a heavy part \( R^H = R - R^L \)

\[ Q(a) = Q_L(a) + Q_H(a) \]

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

\[ Q_H(a) = \sum_b R^H(a, b) \cdot S(b) \]
Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

Materialize the result
Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

**Materialize the result**

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

\[ R^L(a, b) \]

\[ S(b) \]

\[ a_i \]

\[ ... \]

\[ a_j \]

\[ a_1 \]

\[ b_1 \]

\[ ... \]

\[ a_n \]

\[ b_n \]

\[ b'_1 \]

\[ ... \]

\[ b'_m \]

\[ Q_L(A) \text{ can be computed in time } O(N) \]

\[ Q_L(A) \text{ allows constant-time lookups and constant-delay enumeration} \]
Preprocessing in the Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

- \( Q_L \) can be computed in time \( \mathcal{O}(N) \)
Enumeration in the Light Case

$$Q_L(a) = \sum_b R^L(a, b) \cdot S(b)$$

- $Q_L$ allows constant-time lookups and constant-delay enumeration
Updates in the Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

Updates to \( R^L \):
- \( O(1) \)

Updates to \( S \):
- \( O(N^\varepsilon) \)

- Update \( a_0 \) \( b_0 \)
- Iterate over values 
  - \( a_1 \)...
  - \( a_n \) with \( N \in \varepsilon > 0 \)
  - Propagate updates 

Diagram:
- \( R^L(a, b) \) and \( S(b) \) connected to \( Q_L(a) \)
- Update arrow \( a_0 \) \( b_0 \)
Updates in the Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]
Updates in the Light Case

\[ Q_L(a) = \sum_b R_L(a, b) \cdot S(b) \]

- update \( a_0 \) \( b_0 \)
- propagate update \( a_0 \)
- look up \( b_0 \)
Updates in the Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

- Update: \( a_0 b_0 \) → \( a_0 \) and \( b_0 \)
- Propagate update: \( a_0 \) → \( a_1 \)...
- Look up: \( b_0 \)

- Updates to \( R^L \): \( O(1) \)
Updates in the Light Case

\[ Q_L(a) = \sum_b R_L(a, b) \cdot S(b) \]

- Updates to \( R_L \): \( O(1) \)

- Updates to \( S \): \( O(N\varepsilon) \)
Updates in the Light Case

$Q_L(a) = \sum_b R^L(a, b) \cdot S(b)$

iterate over values $a$

$N^\epsilon \xrightarrow{\text{iterate over values } a} \left\{ \begin{array}{c} a_1 \\ \vdots \\ b_0 \\ a_n \end{array} \right\}$

Updates to $R^L$: $O(1)$
Updates in the Light Case

\[ Q_L(a) = \sum_b R_L(a, b) \cdot S(b) \]

- **Iterate over values** \( a \)
- **Propagate update** \( R_L(a, b) \)
- **Update** \( S(b) \)

**Updates to** \( R_L \): \( \mathcal{O}(1) \)
Updates in the Light Case

\[ Q_L(a) = \sum_b R^L(a, b) \cdot S(b) \]

- Iterate over values of \( a \)
- Propagate update

- \( R^L(a, b) \)
- \( S(b) \)

- \( N^\varepsilon \)

- Updates to \( R^L \): \( O(1) \)
- Updates to \( S \): \( O(N^\varepsilon) \)
Heavy Case

\[ Q_H(a) = \sum_{b} R_H^H(a, b) \cdot S(b) \]

Materialize the \( b \) values in the join result
Heavy Case

\[ Q_H(a) = \sum_{b} R^H(a, b) \cdot S(b) \]

*Materialize the b values in the join result*

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

\[ R^H(a, b) \]

\[ a_1 b_1 \]

\[ \ldots \]

\[ a_n b_n \]

\[ b_1 \]

\[ \ldots \]

\[ b_i \]

\[ b_j \]

\[ \leq N^{1-\epsilon} \]

\[ S(b) \]

\[ b'_1 \]

\[ \ldots \]

\[ b'_m \]

\[ \leq N^{1-\epsilon} \]

For each \( b \) value, tuples \((a, b)\) in the join of \( R_H \) and \( S \) admit constant lookup time and enumeration delay.
Preprocessing in the Heavy Case

\[ Q_H(a) = \sum_b R^H(a, b) \cdot S(b) \]

*Materialize the \( b \) values in the join result*

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

\[ R^H(a, b) \]

\[ a_1 \ b_1 \]

\[ \ldots \ldots \]

\[ a_n \ b_n \]

\[ b_i \]

\[ \ldots \]

\[ b_j \]

\[ b_1' \]

\[ \ldots \]

\[ b_m' \]

\[ S(b) \]

\[ \leq N^{1-\varepsilon} \]

\[ \leq N^{1-\varepsilon} \]

\[ \leq N^{1-\varepsilon} \]

\[ V_{RS} \] can be computed in time \( \mathcal{O}(N^{1-\varepsilon}) \) and has at most \( N^{1-\varepsilon} \) values
Enumeration in the Heavy Case

\[ Q_H(a) = \sum_b R^H(a, b) \cdot S(b) \]

\[ V_{RS}(b) = V_R(b) \cdot S(b) \leq N^{1-\epsilon} \]

\[ V_R(b) = \sum_a R^H(a, b) \leq N^{1-\epsilon} \]

\[ R^H(a, b) \leq N^{1-\epsilon} \]

- \( V_{RS} \) contains at most \( N^{1-\epsilon} \) values \( b \)
- For each value \( b \) in \( V_{RS} \), the values \( a \) in \( R^H \) paired with \( b \) admit constant enumeration delay
Enumeration of Distinct Tuples from Union

- $V_{RS}(b)$ contains at most $N^{1-\varepsilon}$ values

- For each value $b$ in $V_{RS}$, the values $a$ in $R^H$ paired with $b$ admit constant enumeration delay

- Yet: For two distinct $b_1$ and $b_2$, the sets of values $a$ in $R^H(a, b_1)$ and $R^H(a, b_2)$ may not be disjoint

  $\implies$ Enumerating all the values $a$ in $R^H(a, b_1)$ and $R^H(a, b_2)$ can lead to duplicates
Enumeration of Distinct Tuples from Union

- $V_{RS}(b)$ contains at most $N^{1-\varepsilon}$ values

- For each value $b$ in $V_{RS}$, the values $a$ in $R^H$ paired with $b$ admit constant enumeration delay

- Yet: For two distinct $b_1$ and $b_2$, the sets of values $a$ in $R^H(a, b_1)$ and $R^H(a, b_2)$ may not be disjoint

  $\implies$ Enumerating all the values $a$ in $R^H(a, b_1)$ and $R^H(a, b_2)$ can lead to duplicates

Union Algorithm [CSL 2011]

- The distinct values $a$ can be enumerated with $\mathcal{O}(N^{1-\varepsilon})$ delay
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay

$S_1$ $a_3$ $a_4$ $a_1$ $a_2$ EOF $S_2$ $a_5$ $a_6$ $a_2$ $a_4$ EOF $S_1 \cup S_2$
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay

\begin{align*}
S_1 &= \{a_3, a_4, a_1, a_2\} \quad \text{EOF} \\
S_2 &= \{a_5, a_6, a_2, a_4\} \quad \text{EOF}
\end{align*}

$S_1 \cup S_2$
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\Rightarrow$ The union of the sets can be enumerated with $O(\ell + d)$ delay

S\textsubscript{1} \text{\ \ \ } a_{3} \text{ a_{4} a_{1} a_{2} EOF \ \ \ \ } S\textsubscript{2} \text{\ \ \ a_{5} a_{6} a_{2} a_{4} EOF \ \ \ \ a_{3} \text{ } S_{1} \cup S_{2}$
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay

---

$S_1 \cup S_2$

$S_1 = \{a_1, a_2, a_3, a_4\}$

$S_2 = \{a_5, a_6, a_2, a_4\}$
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$
- The union of the sets can be enumerated with $O(\ell + d)$ delay
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay

\[
\begin{align*}
S_1 & \quad S_2 & \quad S_1 \cup S_2 \\
a_3 & a_4 & a_1 & a_2 & EOF & \uparrow a_5 & a_6 & a_2 & a_4 & EOF & \uparrow a_3 & a_5 & a_1 & a_6 & a_2
\end{align*}
\]
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay
The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time $\ell$ and enumeration delay $d$

$\implies$ The union of the sets can be enumerated with $O(\ell + d)$ delay

Generalization: Enumeration from the union of $n$ sets

- Each set allows lookup time $\ell$ and enumeration delay $d$
- The union of the sets can be enumerated with $O(n(\ell + d))$ delay
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

\[ R^H(a, b) \]

update → \(a_0 \ b_0\)
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

propagate update  \( \rightarrow \)  \( b_0 \)

update  \( \rightarrow \)  \( a_0 \ b_0 \)
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

propagate update \(-\rightarrow\) \(b_0\)

look up \(-\rightarrow\) \(b_0\)

update \(-\rightarrow\) \(a_0 b_0\)
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

update \( a_0 \), \( b_0 \) → \( R^H(a, b) \) → \( b_0 \)

propagate update \( b_0 \) → \( S(b) \)

look up \( b_0 \)
Updates in the Heavy Case

\[ V_{RS}(b) = V_{R}(b) \cdot S(b) \]

\[ V_{R}(b) = \sum_{a} R^{H}(a, b) \]

- Propagate update \( \rightarrow b_0 \)
- Look up \( \rightarrow b_0 \)
- Update \( \rightarrow a_0 \quad b_0 \)

- Updates to \( R^{H} \): \( \mathcal{O}(1) \)
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

- Updates to \( R^H \): \( \mathcal{O}(1) \)

\( R^H(a, b) \)

\( S(b) \)

\( b_0 \)

update - - →
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

- **Updates to** \( R^H \): \( \mathcal{O}(1) \)

- Look up \( b_0 \) from \( R^H(a, b) \)
- Update \( S(b) \) to \( b_0 \)

\( R^H(a, b) \)
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

\[ V_R(b) = \sum_a R^H(a, b) \]

\[ R^H(a, b) \]

\[ b_0 \]

\[ S(b) \]

- Updates to \( R^H \): \( \mathcal{O}(1) \)
Updates in the Heavy Case

\[ V_{RS}(b) = V_R(b) \cdot S(b) \]

- Updates to \( R^H \): \( \mathcal{O}(1) \)
- Updates to \( S \): \( \mathcal{O}(1) \)
Summing Up

\[ Q(a) = R(a, b) \cdot S(b) \]

Preprocessing Time

<table>
<thead>
<tr>
<th></th>
<th>light case</th>
<th>heavy case</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( O(N) )</td>
<td>( O(N^{1-\varepsilon}) )</td>
<td>( O(N) )</td>
</tr>
</tbody>
</table>

Enumeration Delay

<table>
<thead>
<tr>
<th></th>
<th>light case</th>
<th>heavy case</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>( O(1) )</td>
<td>( O(N^{1-\varepsilon}) )</td>
<td>( O(N^{1-\varepsilon}) )</td>
</tr>
</tbody>
</table>

Update Time

<table>
<thead>
<tr>
<th></th>
<th>light case</th>
<th>heavy case</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( O(N^\varepsilon) )</td>
<td>( O(1) )</td>
<td>( O(N^\varepsilon) )</td>
</tr>
</tbody>
</table>
Are there more queries with the same weak Pareto optimality as our previous example?
For any bound variable $X$ and any atom $\alpha$ of $X$, there is at most one other atom $\beta$ so that all free variables dominated by $X$ are covered by $\alpha$ and $\beta$ together.

The query is hierarchical and not $q$-hierarchical.
\( \delta_1 \)-Hierarchical Queries

- For any bound variable \( X \) and any atom \( \alpha \) of \( X \), there is at most one other atom \( \beta \) so that all free variables dominated by \( X \) are covered by \( \alpha \) and \( \beta \) together.
- The query is hierarchical and not \( q \)-hierarchical.

\[
\delta_1\text{-hierarchical}
Q(a, d, e, g) = R(a, b, d) \cdot S(a, b, e) \cdot T(a, c, f) \cdot U(a, c, g)
\]

\[
hierarchical \text{ but not } \delta_1\text{-hierarchical}
Q(d, g) = R(a, b, d) \cdot S(a, b, e) \cdot T(A, C, F) \cdot U(a, c, g)
\]
For any $\delta_1$-hierarchical query, there is no algorithm that admits preprocessing time, update time, and enumeration delay that are arbitrary $O(N^{1/2-\gamma})$. For any $\gamma > 0$, unless the OMv Conjecture (*) fails, any $\delta_1$-hierarchical query can be maintained with preprocessing time, update time, and enumeration delay $O(N^{1+\epsilon})$, $O(N^{\epsilon})$, and $O(N^{1-\epsilon})$. Except for $\epsilon = 1/2$, this is weakly Pareto optimal, unless OMv Conjecture (*) fails.

(*) Online Matrix-Vector Multiplication cannot be solved in sub-cubic time.
Optimality for $\delta_1$-Hierarchical Queries

- For any $\delta_1$-hierarchical query, there is no algorithm that admits preprocessing time, update time, and enumeration delay arbitrary $O(N^{1/2-\gamma})$ for any $\gamma > 0$, unless the OMv Conjecture (*) fails.

- Any $\delta_1$-hierarchical query can be maintained with preprocessing time, update time, and enumeration delay $O(N^{1+\epsilon})$, $O(N^{\epsilon})$, and $O(N^{1-\epsilon})$.

(*) Online Matrix-Vector Multiplication cannot be solved in sub-cubic time.
For any $\delta_1$-hierarchical query, there is no algorithm that admits preprocessing time, update time, and enumeration delay that is arbitrary $O(N^{1/2-\gamma})$ for any $\gamma > 0$, unless the OMv Conjecture (*) fails.

Any $\delta_1$-hierarchical query can be maintained with preprocessing time, update time, and enumeration delay $O(N^{1+\varepsilon})$, $O(N^{\varepsilon})$, and $O(N^{1-\varepsilon})$.

For $\varepsilon = 1/2$, this is weakly Pareto optimal, unless OMv Conjecture fails.

(*) Online Matrix-Vector Multiplication cannot be solved in sub-cubic time.

---

**Optimality for $\delta_1$-Hierarchical Queries**
We can define syntactically classes of $\delta_i$-hierarchical queries ($i \in \mathbb{N}$)

- with $O(N^{i\varepsilon})$ update time and $O(N^{1-\varepsilon})$ enumeration delay.
- $\delta_0$-hierarchical $=$ $Q$-hierarchical

[LMCS 2023]
Any hierarchical query can be maintained with

preprocessing time \( \mathcal{O}(N^{1+(w-1)\varepsilon}) \)
update time \( \mathcal{O}(N^{\delta\varepsilon}) \)
enumeration delay \( \mathcal{O}(N^{1-\varepsilon}) \)

where

- static width \( w \) = the fractional hypertree width for CQs
- dynamic width \( \delta = \max_{\delta \text{ queries}} \text{static width} \)

[PODS 2020]
Any hierarchical query can be maintained with

\[
\begin{align*}
\text{preprocessing time} & = \mathcal{O}(N^{1+(w-1)\epsilon}) \\
\text{update time} & = \mathcal{O}(N^{\delta \epsilon}) \\
\text{enumeration delay} & = \mathcal{O}(N^{1-\epsilon})
\end{align*}
\]

where

- static width \( w \) = the fractional hypertree width for CQs
- dynamic width \( \delta = \max_{\text{delta queries}} \text{static width} \)

Open question: Lower bounds for hierarchical queries
Hierarchical queries admit sublinear update time and enumeration delay
Trade-Offs Beyond Hierarchical

- No nice closed-form expression for complexities seem possible
- For some $\alpha$-acyclic queries, trade-offs seem not possible
- First steps already made for $\alpha$-acyclic queries [CSL 2023]
conjunctive
\(\mathcal{O}(N^w)/\mathcal{O}(N^\delta)/\mathcal{O}(1)\) [SIGMOD ’18]
IVM Landscape (Partial)

Preprocessing time/Update time/Enumeration delay

conjunctive
\(O(N^w)/O(N^\delta)/O(1)\) [SIGMOD '18]

triangle join \(O(N^{1.5})/O(N^{0.5})/O(1)\) [TODS '20]
IVM Landscape (Partial)

Preprocessing time/Update time/Enumeration delay

**conjunctive**

\[ O(N^w) / O(N^\delta) / O(1) \quad \text{[SIGMOD '18]} \]

**triangle join**

\[ O(N^{1.5}) / O(N^{0.5}) / O(1) \quad \text{[TODS '20]} \]

**\(\alpha\)-acyclic**

**free-connex**

\[ O(N) / O(N) / O(1) \quad \text{[SIGMOD '17]} \]
conjunctive
\[ \mathcal{O}(N^w)/\mathcal{O}(N^\delta)/\mathcal{O}(1) \] [SIGMOD ’18]

triangle join
\[ \mathcal{O}(N^{1.5})/\mathcal{O}(N^{0.5})/\mathcal{O}(1) \] [TODS ’20]

\(\alpha\)-acyclic

hierarchical [PODS ’20]
\[ \mathcal{O}(N^{1+(w−1)\varepsilon})/\mathcal{O}(N^{\delta\varepsilon})/\mathcal{O}(N^{1−\varepsilon}) \]
\(\varepsilon \in [0, 1]\)

free-connex
\[ \mathcal{O}(N)/\mathcal{O}(N)/\mathcal{O}(1) \] [SIGMOD ’17]
Preprocessing time/Update time/Enumeration delay

**conjunctive**
\[ \mathcal{O}(N^w)/\mathcal{O}(N^\delta)/\mathcal{O}(1) \] [SIGMOD '18]

**triangle join**
\[ \mathcal{O}(N^{1.5})/\mathcal{O}(N^{0.5})/\mathcal{O}(1) \] [TODS '20]

**\(\alpha\)-acyclic**

**hierarchical** [PODS '20]
\[ \mathcal{O}(N^{1+(w-1)\varepsilon})/\mathcal{O}(N^{\delta\varepsilon})/\mathcal{O}(N^{1-\varepsilon}) \]
\[ \varepsilon \in [0, 1] \]

**q-hierarchical**
\[ \mathcal{O}(N)/\mathcal{O}(N)/\mathcal{O}(1) \] [PODS '17]

**\(\delta_0\)-hierarchical**
\[ w = 1, \delta = 0 \]

**free-connex**
[SIGMOD '17]
Preprocessing time/Update time/Enumeration delay

**conjunctive**
\[ O(N^w) / O(N^\delta) / O(1) \] [SIGMOD '18]

**triangle join**
\[ O(N^{1.5}) / O(N^{0.5}) / O(1) \] [TODS '20]

**\(\alpha\)-acyclic**

**hierarchical** [PODS '20]
\[ O(N^{1+(w-1)\varepsilon}) / O(N^{\delta\varepsilon}) / O(N^{1-\varepsilon}) \]
\(\varepsilon \in [0, 1]\)

**q-hierarchical**
\[ \delta_0\text{-hierarchical} = \]
\(w = 1, \delta = 0\)

**\(\delta_1\text{-hierarchical}**
\(w \in \{1, 2\}, \delta = 1\)

**free-connex**
\[ O(N) / O(N) / O(1) \] [SIGMOD '17]
Recovery of Prior Results

- Preprocessing time: $O(N^{1+(w-1)\varepsilon})$
- Update time: $O(N^{\delta\varepsilon})$
- Enumeration delay: $O(N^{1-\varepsilon})$
Recovery of Prior Results

**Preprocessing time:** \( O(N^{1+(w-1)\varepsilon}) \)

**Update time:** \( O(N^{\delta\varepsilon}) \)

**Enumeration delay:** \( O(N^{1-\varepsilon}) \)
Recovery of Prior Results

\[ \log_N \text{update time} = O(N^{1+(w-1)\varepsilon}) \]

\[ \log_N \text{preprocessing time} = O(N^{\delta_0}) \]

\[ \log_N \text{delay} = O(N^{1-\varepsilon}) \]

\[ \text{conjunctive} \]

\[ \delta_0\text{-hierarchical} \]

\[ (w = 1, \delta = 0) \]

\[ \delta_1\text{-hierarchical} \]

\[ (w = 1, \delta = 1) \]

\[ \text{free-connex} \]

\[ \log N \text{ time} = O(N^{\delta_0}) \]

\[ \log N \text{ time} = O(N^{1-\varepsilon}) \]


4. Maintaining ML Models over Evolving Relational Data
Maintain Models under Updates

1. Polynomial Regression: Find parameters $\Theta$ best satisfying

<table>
<thead>
<tr>
<th>Size (m²)</th>
<th>#beds</th>
<th>Year</th>
<th>Region 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>403</td>
<td>7</td>
<td>1925</td>
<td>1</td>
</tr>
<tr>
<td>189</td>
<td>6</td>
<td>1948</td>
<td>1</td>
</tr>
<tr>
<td>568</td>
<td>8</td>
<td>1935</td>
<td>0</td>
</tr>
<tr>
<td>420</td>
<td>4</td>
<td>1908</td>
<td>0</td>
</tr>
<tr>
<td>246</td>
<td>5</td>
<td>1928</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price (CHF)</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,450,000</td>
<td>3</td>
</tr>
<tr>
<td>2,750,000</td>
<td>2</td>
</tr>
<tr>
<td>6,000,000</td>
<td>4</td>
</tr>
<tr>
<td>4,600,000</td>
<td>1</td>
</tr>
<tr>
<td>3,250,000</td>
<td>2</td>
</tr>
</tbody>
</table>

- Features $X$ and labels $Y$ are given by database joins
Maintain Models under Updates

1. Polynomial Regression: Find parameters $\Theta$ best satisfying

\[
\begin{array}{c|c|c|c|c}
\text{Size (m}^2\text{)} & \#\text{beds} & \text{Year} & \text{Region 1} \\
403 & 7 & 1925 & 1 \\
189 & 6 & 1948 & 1 \\
568 & 8 & 1935 & 0 \\
420 & 4 & 1908 & 0 \\
246 & 5 & 1928 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{Price (CHF)} & \text{Rating} \\
3,450,000 & 3 \\
2,750,000 & 2 \\
6,000,000 & 4 \\
4,600,000 & 1 \\
3,250,000 & 2 \\
\end{array}
\]

- Features $X$ and labels $Y$ are given by database joins
- Solved using iterative gradient computation:
  \[
  \Theta_{i+1} = \Theta_i - \alpha X^T (X \Theta_i - Y) \quad \text{(repeat until convergence)}
  \]

2. Chow-Liu Trees: based on pairwise mutual information

Approach for both: Maintain the Covariance Matrix $[X \ Y]^T [X \ Y]$

[SIGMOD 2018 & 2020, VLDB J 2023]
Covariance Matrix Defined by Queries

Covariance matrix \([X \ Y]^T [X \ Y]\) can be expressed in SQL

\[
Q = \text{SELECT } \text{SUM}(1 \times 1), \text{SUM}(1 \times X), \ldots \text{SUM}(1 \times X_n), \text{SUM}(1 \times Y), \\
\text{SUM}(X \times 1), \text{SUM}(X \times X), \ldots \text{SUM}(X \times X_n), \text{SUM}(X \times Y), \\
\ldots \\
\text{SUM}(X_n \times 1), \text{SUM}(X_n \times X), \ldots \text{SUM}(X_n \times X_n), \text{SUM}(X_n \times Y) \\
\text{SUM}(Y \times 1), \text{SUM}(Y \times X), \ldots \text{SUM}(Y \times X_n), \text{SUM}(Y \times Y) \\
\text{FROM } R1 \text{ JOIN } R2 \text{ JOIN } \ldots \text{ JOIN } Rn
\]
Covariance Matrix Defined by Queries

Covariance matrix \([\mathbf{X} \mathbf{Y}]^T [\mathbf{X} \mathbf{Y}]\) can be expressed in SQL

\[
Q = \text{SELECT} \begin{bmatrix}
\text{SUM}(1 \ast 1), & \text{SUM}(1 \ast X_1), & \ldots & \text{SUM}(1 \ast X_n), & \text{SUM}(1 \ast Y), \\
\text{SUM}(X_1 \ast 1), & \text{SUM}(X_1 \ast X_1), & \ldots & \text{SUM}(X_1 \ast X_n), & \text{SUM}(X_1 \ast Y), \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\text{SUM}(X_n \ast 1), & \text{SUM}(X_n \ast X_1), & \ldots & \text{SUM}(X_n \ast X_n), & \text{SUM}(X_n \ast Y), \\
\text{SUM}(Y \ast 1), & \text{SUM}(Y \ast X_1), & \ldots & \text{SUM}(Y \ast X_n), & \text{SUM}(Y \ast Y)
\end{bmatrix}
\]

\[
\text{FROM} \ R1 \ JOIN \ R2 \ JOIN \ldots \ JOIN \ R_n
\]

We compute and maintain under data updates:

- \(\text{COUNT} = \text{SUM}(1) = \text{database join size}\)
- vector of \(\text{SUM}(\mathbf{X}_i)\) for feature/label \(\mathbf{X}_i\)
- matrix of \(\text{SUM}(\mathbf{X}_i \cdot \mathbf{X}_j)\) for features/label \(\mathbf{X}_i\) and \(\mathbf{X}_j\)
The Covariance Ring

**Covariance Ring** has the support:

- Set of triples \((\mathbb{Z}, \mathbb{R}^m, \mathbb{R}^{m \times m})\)

\[
\left( \text{COUNT}, \text{ vector of } \sum(X_i), \text{ matrix of } \sum(X_i \cdot X_j) \right)
\]

- Neutral elements for sum and product operations:

\[
0 = (0, 0_{m \times 1}, 0_{m \times m})
\]
\[
1 = (1, 0_{m \times 1}, 0_{m \times m})
\]
Covariance Ring has the sum and product operations:

\[ a = (\square, \begin{bmatrix} \square \\ \square \end{bmatrix}, \begin{bmatrix} \square \\ \square \end{bmatrix}) \quad b = (\square, \begin{bmatrix} \square \\ \square \end{bmatrix}, \begin{bmatrix} \square \\ \square \end{bmatrix}) \]
The Covariance Ring

Covariance Ring has the sum and product operations:

\[
a = \left( \begin{array}{c}
\text{ },
\begin{bmatrix}
\text{ }, & \text{ }, & \text{ },
\end{bmatrix}
\end{array}
\right)
\quad b = \left( \begin{array}{c}
\text{ },
\begin{bmatrix}
\text{ }, & \text{ }, & \text{ },
\end{bmatrix}
\end{array}
\right)
\]

\[
a + b = \left( \begin{array}{c}
\text{ },
\begin{bmatrix}
\text{ }, & \text{ }, & \text{ },
\end{bmatrix}
\end{array}
\right)
\]
The Covariance Ring

Covariance Ring has the sum and product operations:

\[ a = \left( \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right), \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right) \]

\[ b = \left( \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right), \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right) \]

\[ a + b = \left( \begin{array}{c} \text{ } + \\
\text{ } \end{array} \right), \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right), \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right) \]

\[ a \ast b = \left( \begin{array}{c} \text{ } \cdot \\
\text{ } \end{array} \right), \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right), \begin{array}{c} \text{ }, \\
\text{ }, \\
\text{ } \end{array} \right) \]


Thank You!