Introduction to Incremental View Maintenance

Dan Olteanu (University of Zurich)

fdbresearch.github.io

Berkeley CS 294-248: Topics in Database Theory
https://berkeley-cs294-248.github.io/
September 12 & 14, 2023

Short Biography

Since 2020: Professor of Computer Science, University of Zurich

Since 2017: Computer Scientist at RelationalAI

2007 - 2020: Professor of Computer Science, University of Oxford

2013 - 2014: Visiting professor at University of California, Berkeley

Taught CS 186 & worked for LogicBlox

Acknowledgments

Members of the DaST IVM team

Ahmet Kara

Milos Nikolic

Haozhe Zhang

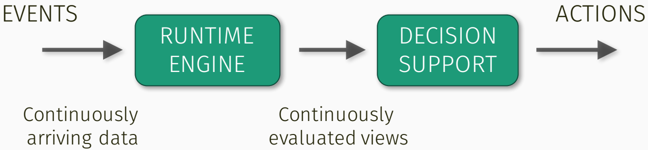
Real-Time Analytics

- Datasets continuously evolve over time
  - E.g.: data streams from sensors, social networks, apps
- Real-time analytics over streaming data
  - Users want fresh up-to-date computation results, e.g., models

Web Analytics

Sensor Networks

Cloud Monitoring



EVENTS

RUNTIME ENGINE

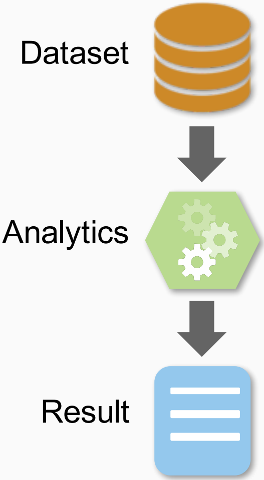
DECISION SUPPORT

ACTIONS

Continuously arriving data

Continuously evaluated views

Real-Time Analytics via Incremental Maintenance

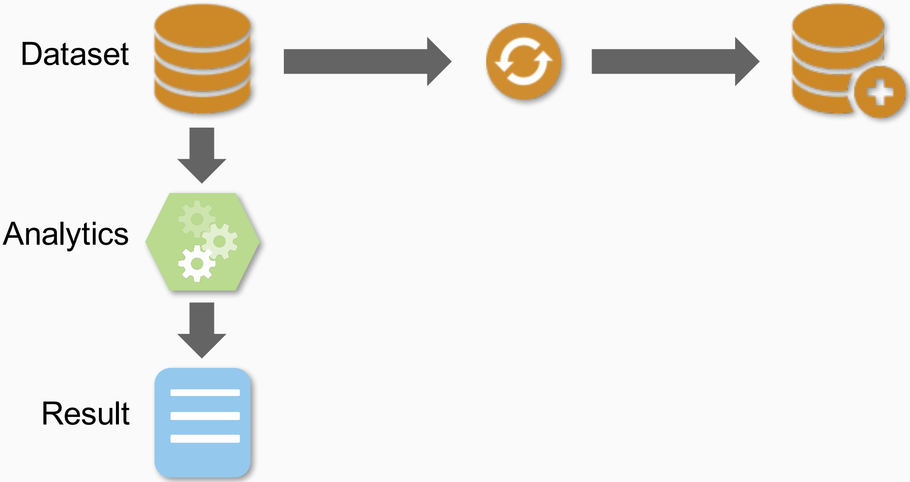


Dataset

Analytics

Result

Real-Time Analytics via Incremental Maintenance

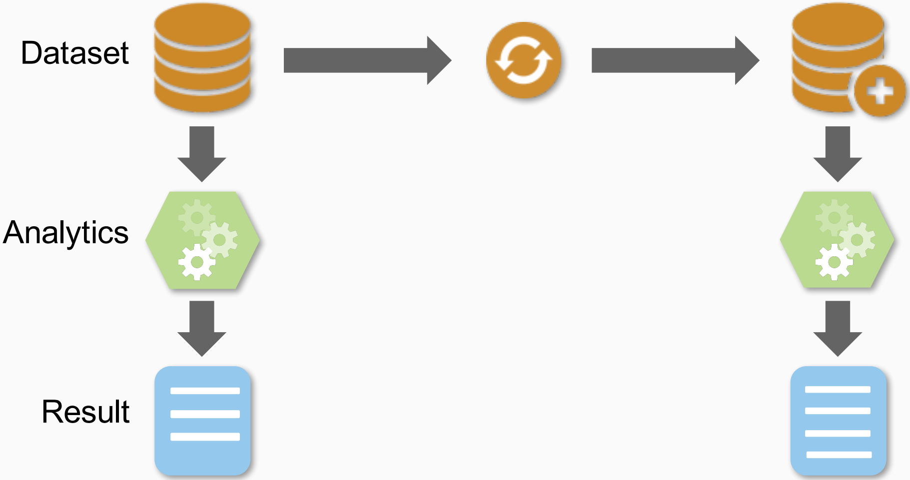


Dataset

Analytics

Result

Real-Time Analytics via Incremental Maintenance

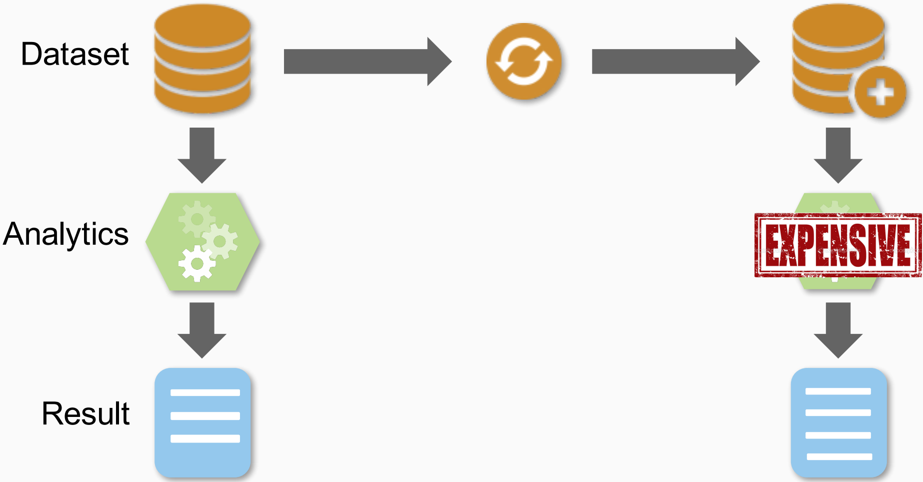


Dataset

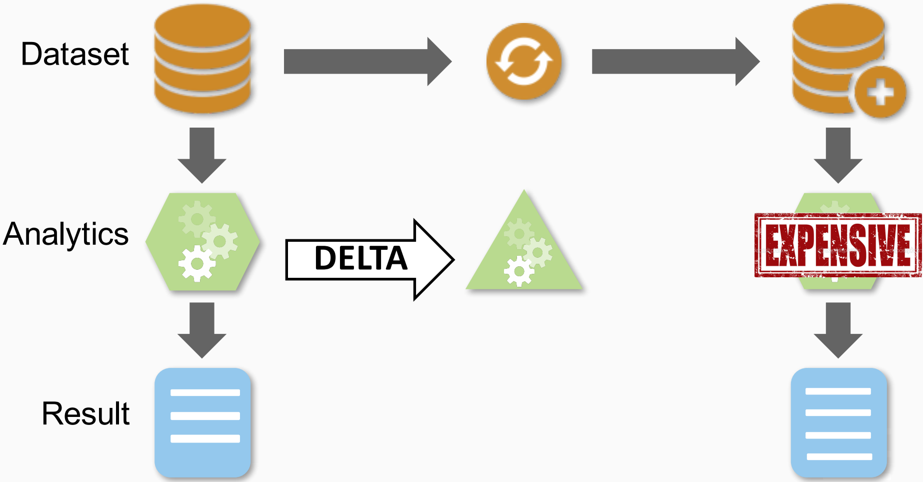
Analytics

Result

Real-Time Analytics via Incremental Maintenance



Real-Time Analytics via Incremental Maintenance



Dataset

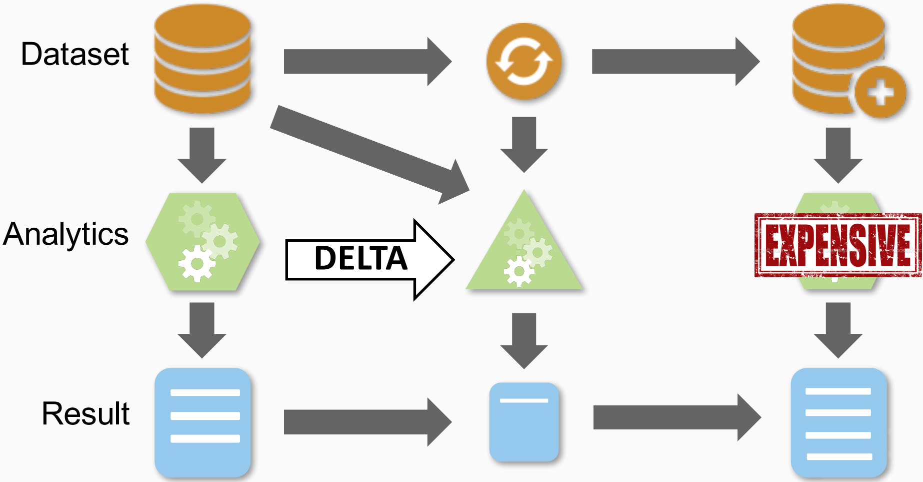
Analytics

Result

DELTA

EXPENSIVE

Real-Time Analytics via Incremental Maintenance



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*

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*

Setting

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

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*

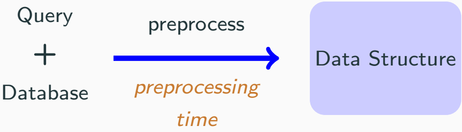
Setting

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

Objectives

- Main IVM approaches: First/higher order, adaptive
- Which queries can be maintained optimally?

The Incremental View Maintenance Problem



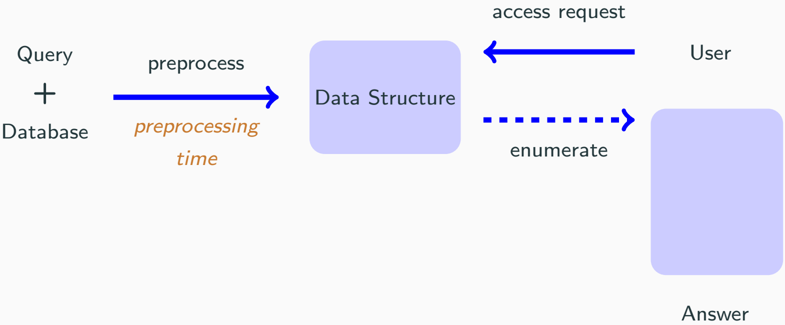
Query
+
Database

preprocess

preprocessing time

Data Structure

The Incremental View Maintenance Problem



Query

+

Database

preprocess

preprocessing time

Data Structure

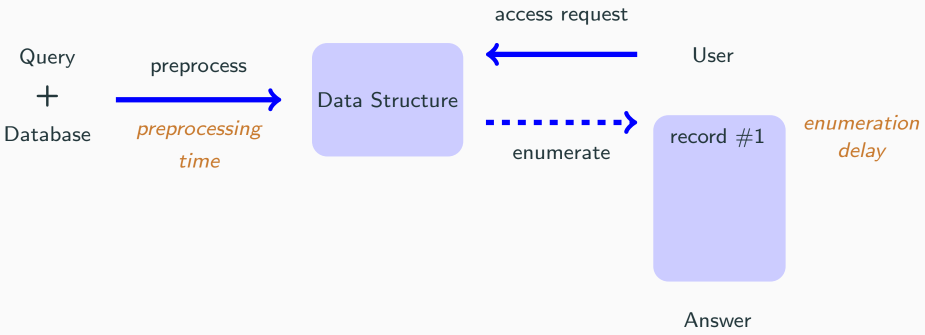
access request

User

enumerate

Answer

The Incremental View Maintenance Problem



access request

Query

+

Database

preprocess

preprocessing time

Data Structure

User

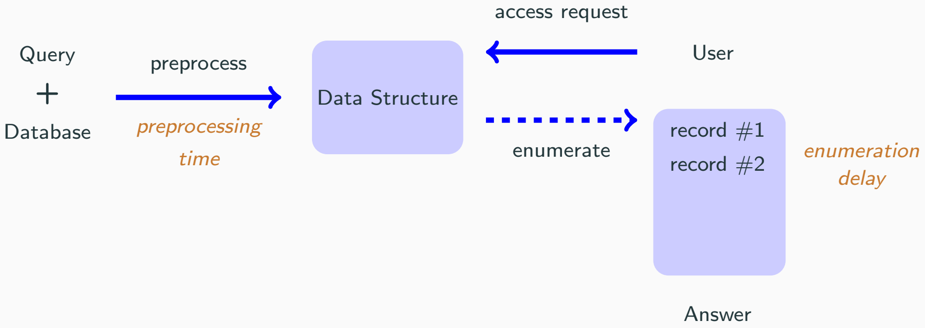
enumerate

record #1

enumeration delay

Answer

The Incremental View Maintenance Problem



access request

Query

$+$

Database

preprocess

preprocessing time

Data Structure

User

enumerate

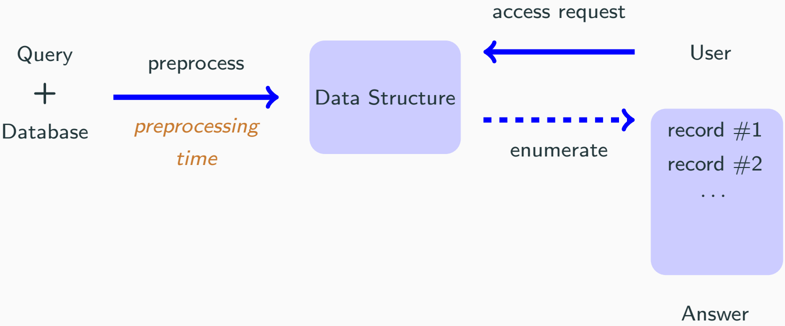
record #1

record #2

enumeration delay

Answer

The Incremental View Maintenance Problem



access request

Query
+
Database

preprocess

preprocessing time

Data Structure

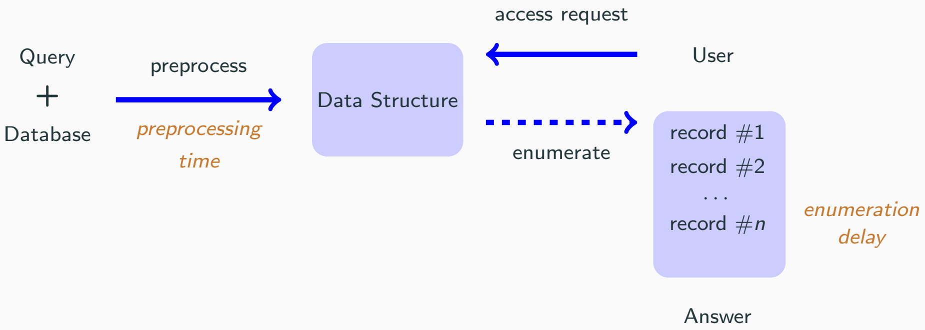
User

enumerate

record #1
record #2
...

Answer

The Incremental View Maintenance Problem



access request

Query

+

Database

preprocess

preprocessing time

Data Structure

User

enumerate

record $\#1$

record $\#2$

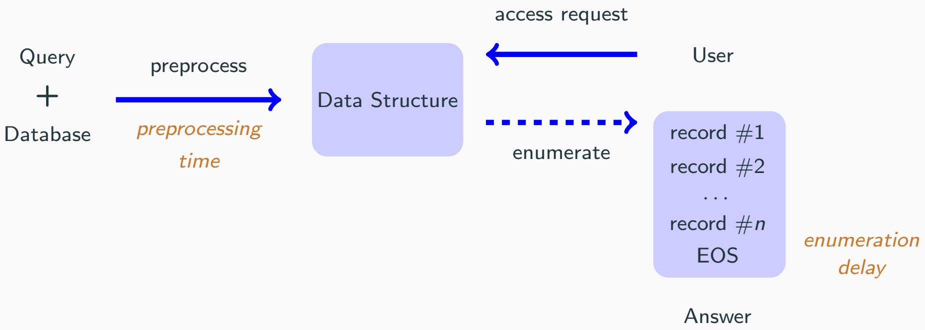
...

record $\#n$

enumeration delay

Answer

The Incremental View Maintenance Problem



access request

Query

preprocess

User

+

Data Structure

Database

preprocessing time

enumerate

record #1

record #2

. . .

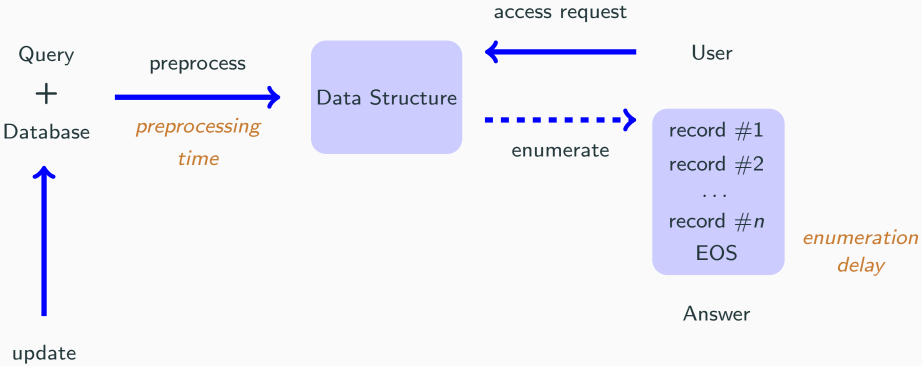
record #n

EOS

enumeration delay

Answer

The Incremental View Maintenance Problem



Query

+

Database

update

preprocess

preprocessing time

Data Structure

access request

User

enumerate

record #1

record #2

. . .

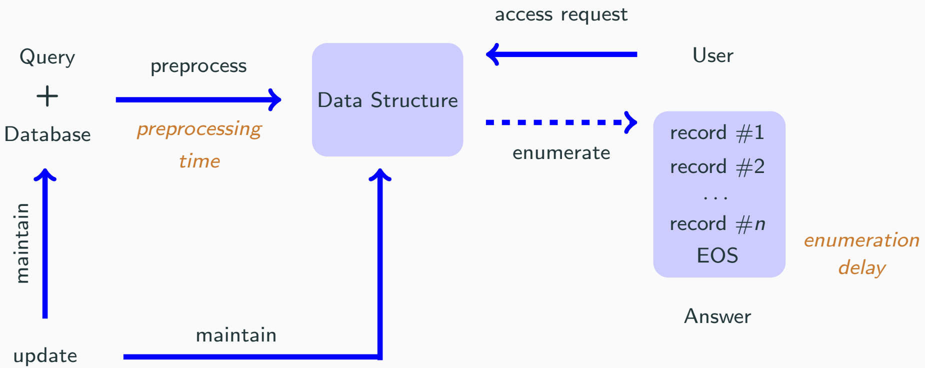
record #n

EOS

Answer

enumeration delay

The Incremental View Maintenance Problem



access request

User

Query

+

Database

preprocess

preprocessing time

Data Structure

enumerate

record #1

record #2

. . .

record #n

EOS

enumeration delay

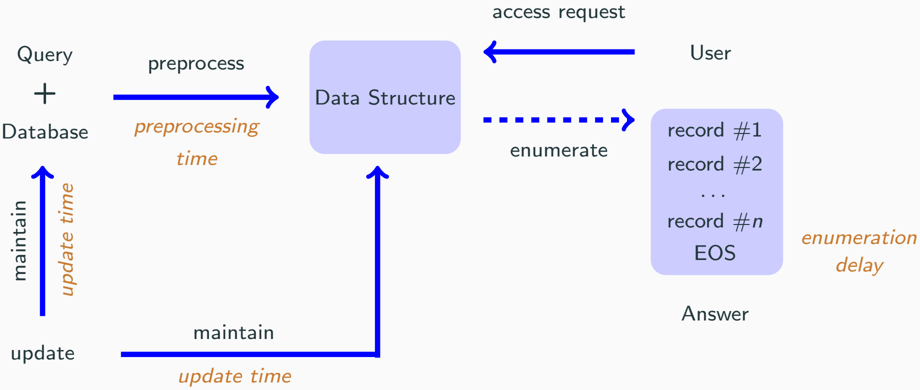
Answer

maintain

maintain

update

The Incremental View Maintenance Problem



access request

Query
+
Database

preprocess

preprocessing time

Data Structure

User

enumerate

record #1
record #2
. . .
record #n
EOS

Answer

enumeration delay

maintain

update time

update

maintain

update time

Agenda

Part 1. Data model and query language

Part 2. Main IVM techniques by example

- First-order IVM using delta queries
- Higher-order IVM using DBToaster and F-IVM
- Adaptive IVM using IVM$^\epsilon$

Part 3. Which queries can be maintained optimally?

- Constant update time and enumeration delay
- Beyond constant time

Part 1.

Data Model & Query Language

Classical Representation of Relations and Updates

Insertions and deletions are represented as separate tables:

employee	age
Elise	35
Elise	35
Steve	40
Joe	30
Joe	30

relation

$\setminus$

employee	age
Steve	40
Joe	30
Joe	30

delete

$\cup$

employee	age
Steve	38
Mary	33
Mary	33

insert

$=$

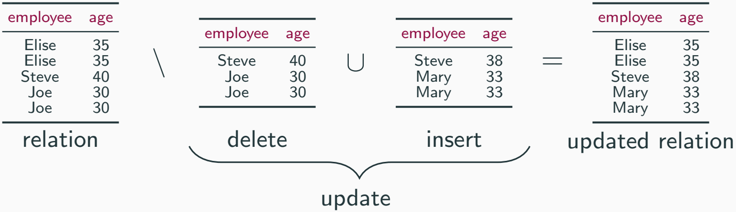
employee	age
Elise	35
Elise	35
Steve	38
Mary	33
Mary	33

updated relation

update

Classical Representation of Relations and Updates

Insertions and deletions are represented as separate tables:



employee	age
Elise	35
Elise	35
Steve	40
Joe	30
Joe	30

relation

$\setminus$

employee	age
Steve	40
Joe	30
Joe	30

delete

$\cup$

employee	age
Steve	38
Mary	33
Mary	33

insert

$=$

employee	age
Elise	35
Elise	35
Steve	38
Mary	33
Mary	33

updated relation

update

- Order of updates matters!

  This affects the parallel and distributed execution of updates

- Inserts and deletes are treated differently

  They rely on different execution mechanisms

Uniform Representation of Relations and Updates

Relations and updates are represented as factors mapping tuples to multiplicities:

employee	age	→	#
Elise	35	→	2
Steve	40	→	1
Joe	30	→	2

factor

$+$

employee	age	→	#
Steve	40	→	-1
Steve	38	→	1
Joe	30	→	-2
Mary	33	→	2

update

$=$

employee	age	→	#
Elise	35	→	2
Steve	38	→	1
Mary	33	→	2

updated factor

Uniform Representation of Relations and Updates

Relations and updates are represented as factors mapping tuples to multiplicities:

employee	age	→	#
Elise	35	→	2
Steve	40	→	1
Joe	30	→	2

factor

$+$

employee	age	→	#
Steve	40	→	−1
Steve	38	→	1
Joe	30	→	−2
Mary	33	→	2

update

$=$

employee	age	→	#
Elise	35	→	2
Steve	38	→	1
Mary	33	→	2

updated factor

- Order of updates does not matter: Addition is associative
- Database may be in inconsistent state: tuples with negative multiplicities
- Multiplicity:

  negative = tuple delete; positive = tuple insert; 0 = tuple not in database

Uniform Representation of Relations and Updates

Relations and updates are represented as factors mapping tuples to multiplicities:

employee	age	→	#
Elise	35	→	2
Steve	40	→	1
Joe	30	→	2

factor

+

employee	age	→	#
Steve	40	→	−1
Steve	38	→	1
Joe	30	→	−2
Mary	33	→	2

update

=

employee	age	→	#
Elise	35	→	2
Steve	38	→	1
Mary	33	→	2

updated factor

- Order of updates does not matter: Addition is associative
- Database may be in inconsistent state: tuples with negative multiplicities
- Multiplicity:

  negative = tuple delete; positive = tuple insert; 0 = tuple not in database

From $\mathbb{Z}$ to arbitrary rings (r_1 and r_2 are elements from a ring):

A	B	→	R(A, B)
a_1	b_1	→	r_1
a_2	b_1	→	r_2

Rings

- A ring $(\mathbf{D}, +, \cdot, \mathbf{0}, \mathbf{1})$ is a set $\mathbf{D}$ with two binary ops:

Additive commutativity	$a + b = b + a$
Additive associativity	$(a + b) + c = a + (b + c)$
Additive identity	$\mathbf{0} + a = a + \mathbf{0} = a$
Additive inverse	$\exists - a \in \mathbf{D} : a + (-a) = (-a) + a = \mathbf{0}$
Multiplicative associativity	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
Multiplicative identity	$a \cdot \mathbf{1} = \mathbf{1} \cdot a = a$
Left and right distributivity	$a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$

The ring is commutative if multiplication is also commutative

- Examples: $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$
- More on semi-rings later in the course
- We use the ring $(\mathbb{Z}, +, \cdot, 0, 1)$

Operations on Factors

R(a, b)

A	B	→	#
a_1	b_1	→	r_1
a_2	b_1	→	r_2

S(a, b)

A	B	→	#
a_2	b_1	→	s_1
a_3	b_2	→	s_2

T(b, c)

B	C	→	#
b_1	c_1	→	t_1
b_2	c_2	→	t_2

Operations on Factors

$R(a, b)$

A	B	$\rightarrow$	#
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	r_2

$S(a, b)$

A	B	$\rightarrow$	#
a_2	b_1	$\rightarrow$	s_1
a_3	b_2	$\rightarrow$	s_2

$T(b, c)$

B	C	$\rightarrow$	#
b_1	c_1	$\rightarrow$	t_1
b_2	c_2	$\rightarrow$	t_2

Union/Sum $V(a, b) = R(a, b) + S(a, b)$

A	B	$\rightarrow$	#
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	$r_2 + s_1$
a_3	b_2	$\rightarrow$	s_2

Operations on Factors

$R(a, b)$

A	B	$\rightarrow$	$\#$
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	r_2

$S(a, b)$

A	B	$\rightarrow$	$\#$
a_2	b_1	$\rightarrow$	s_1
a_3	b_2	$\rightarrow$	s_2

$T(b, c)$

B	C	$\rightarrow$	$\#$
b_1	c_1	$\rightarrow$	t_1
b_2	c_2	$\rightarrow$	t_2

Union/Sum $V(a, b) = R(a, b) + S(a, b)$

A	B	$\rightarrow$	$\#$
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	$r_2 + s_1$
a_3	b_2	$\rightarrow$	s_2

Join/Product

$W(a, b, c) = V(a, b) \cdot T(b, c)$

A	B	C	$\rightarrow$	$\#$
a_1	b_1	c_1	$\rightarrow$	$r_1 \cdot t_1$
a_2	b_1	c_1	$\rightarrow$	$(r_2 + s_1) \cdot t_1$
a_3	b_2	c_2	$\rightarrow$	$s_2 \cdot t_2$

Operations on Factors

$R(a, b)$

A	B	→	#
a_1	b_1	→	r_1
a_2	b_1	→	r_2

$S(a, b)$

A	B	→	#
a_2	b_1	→	s_1
a_3	b_2	→	s_2

$T(b, c)$

B	C	→	#
b_1	c_1	→	t_1
b_2	c_2	→	t_2

Union/Sum $V(a, b) = R(a, b) + S(a, b)$

A	B	→	#
a_1	b_1	→	r_1
a_2	b_1	→	$r_2 + s_1$
a_3	b_2	→	s_2

Join/Product

$W(a, b, c) = V(a, b) \cdot T(b, c)$

A	B	C	→	#
a_1	b_1	c_1	→	$r_1 \cdot t_1$
a_2	b_1	c_1	→	$(r_2 + s_1) \cdot t_1$
a_3	b_2	c_2	→	$s_2 \cdot t_2$

Project/Marginalization

$U(b, c) = \sum_a W(a, b, c)$

B	C	→	#
b_1	c_1	→	$r_1 * t_1 + (r_2 + s_1) \cdot t_1$
b_2	c_2	→	$s_2 \cdot t_2$

Query Language Considered in This Lecture

Language of queries with operations sum, product, and variable marginalization over factors. Examples:

- List the paths of length two in a graph with edge factor E:

$$Q(a, b, c) = E(a, b) \cdot E(b, c)$$

Query Language Considered in This Lecture

Language of queries with operations sum, product, and variable marginalization over factors. Examples:

- List the paths of length two in a graph with edge factor E:

  $$Q(a, b, c) = E(a, b) \cdot E(b, c)$$

- List the source-target nodes of the 2-length paths (b ranges over the vertices in the graph):

  $$Q(a, c) = \sum_b E(a, b) \cdot E(b, c)$$

Query Language Considered in This Lecture

Language of queries with operations sum, product, and variable marginalization over factors. Examples:

- List the paths of length two in a graph with edge factor E:

$$Q(a, b, c) = E(a, b) \cdot E(b, c)$$

- List the source-target nodes of the 2-length paths (b ranges over the vertices in the graph):

$$Q(a, c) = \sum_b E(a, b) \cdot E(b, c)$$

- List the triangles:

$$Q(a, b, c) = E(a, b) \cdot E(b, c) \cdot E(c, a)$$

Query Language Considered in This Lecture

Language of queries with operations sum, product, and variable marginalization over factors. Examples:

- List the paths of length two in a graph with edge factor E:

$$Q(a, b, c) = E(a, b) \cdot E(b, c)$$

- List the source-target nodes of the 2-length paths (b ranges over the vertices in the graph):

$$Q(a, c) = \sum_b E(a, b) \cdot E(b, c)$$

- List the triangles:

$$Q(a, b, c) = E(a, b) \cdot E(b, c) \cdot E(c, a)$$

- Count the triangles (a, b, c range over the set of vertices):

$$Q = \sum_{a,b,c} E(a, b) \cdot E(b, c) \cdot E(c, a)$$

Query Language Considered in This Lecture

Language of queries with operations sum, product, and variable marginalization over factors. Examples:

- List the paths of length two in a graph with edge factor E:

$$Q(a, b, c) = E(a, b) \cdot E(b, c)$$

- List the source-target nodes of the 2-length paths (b ranges over the vertices in the graph):

$$Q(a, c) = \textstyle\sum_b E(a, b) \cdot E(b, c)$$

- List the triangles:

$$Q(a, b, c) = E(a, b) \cdot E(b, c) \cdot E(c, a)$$

- Count the triangles (a, b, c range over the set of vertices):

$$Q = \textstyle\sum_{a,b,c} E(a, b) \cdot E(b, c) \cdot E(c, a)$$

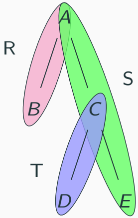
- Update a factor E with changes given by another factor δE:

$$E_{new}(a, b) = E(a, b) + \delta E(a, b)$$

Query Evaluation Example (1/2)

Compute COUNT over the natural join of factors R, S, and T (using Yannakakis)

Factors map tuples to multiplicity 1

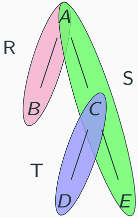


Join hypergraph

Query Evaluation Example (1/2)

Compute COUNT over the natural join of factors R, S, and T (using Yannakakis)

Factors map tuples to multiplicity 1



Join hypergraph

Naïve: compute the join and then COUNT

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

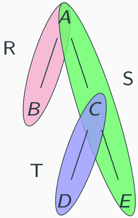
Takes $\mathcal{O}(N^3)$ time if each factor has size $\mathcal{O}(N)$!

There can be $\mathcal{O}(N^3)$ possible combinations of b, e, and d values

Query Evaluation Example (1/2)

Compute COUNT over the natural join of factors R, S, and T (using Yannakakis)

Factors map tuples to multiplicity 1



Join hypergraph

Naïve: compute the join and then COUNT

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

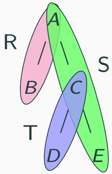
Takes $\mathcal{O}(N^3)$ time if each factor has size $\mathcal{O}(N)$!

There can be $\mathcal{O}(N^3)$ possible combinations of b, e, and d values

Can we compute COUNT in $\mathcal{O}(N)$ time?

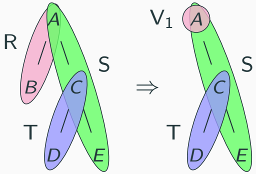
Query Evaluation Example (2/2)

Idea: Push COUNT past joins and marginalize variables



Query Evaluation Example (2/2)

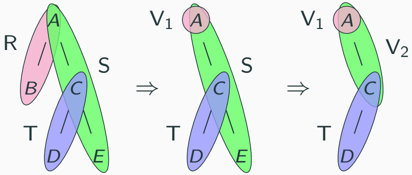
Idea: Push COUNT past joins and marginalize variables



$$V_1(a) = \sum_{b} R(a, b) \qquad \mathcal{O}(N)$$

Query Evaluation Example (2/2)

Idea: Push COUNT past joins and marginalize variables

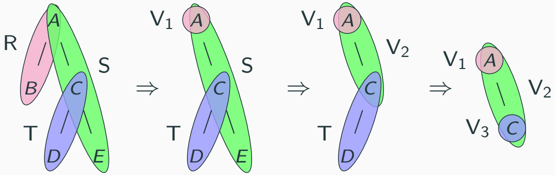


$$V_1(a) = \sum_{b} R(a, b) \qquad \mathcal{O}(N)$$

$$V_2(a, c) = \sum_{e} S(a, c, e) \qquad \mathcal{O}(N)$$

Query Evaluation Example (2/2)

Idea: Push COUNT past joins and marginalize variables



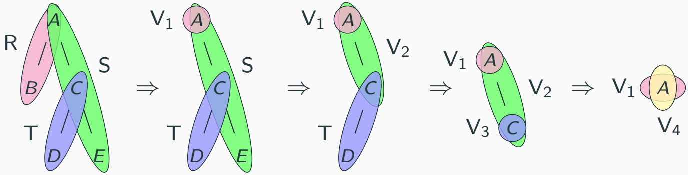
$$V_1(a) = \sum_b R(a,b) \qquad \mathcal{O}(N)$$

$$V_2(a,c) = \sum_e S(a,c,e) \qquad \mathcal{O}(N)$$

$$V_3(c) = \sum_d T(c,d) \qquad \mathcal{O}(N)$$

Query Evaluation Example (2/2)

Idea: Push COUNT past joins and marginalize variables



$$V_1(a) = \sum_b R(a, b) \qquad \mathcal{O}(N)$$

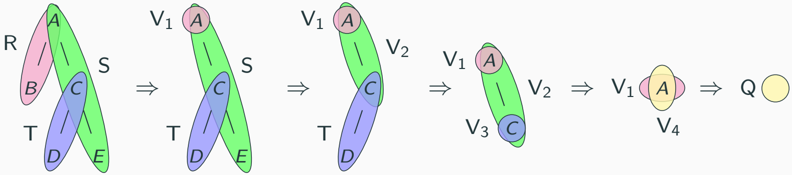
$$V_2(a, c) = \sum_e S(a, c, e) \qquad \mathcal{O}(N)$$

$$V_3(c) = \sum_d T(c, d) \qquad \mathcal{O}(N)$$

$$V_4(a) = \sum_c (V_2(a, c) \cdot V_3(c)) \qquad \mathcal{O}(N)$$

Query Evaluation Example (2/2)

Idea: Push COUNT past joins and marginalize variables



$$V_1(a) = \sum_{b} R(a, b) \qquad \mathcal{O}(N)$$

$$V_2(a, c) = \sum_{e} S(a, c, e) \qquad \mathcal{O}(N)$$

$$V_3(c) = \sum_{d} T(c, d) \qquad \mathcal{O}(N)$$

$$V_4(a) = \sum_{c} (V_2(a, c) \cdot V_3(c)) \qquad \mathcal{O}(N)$$

$$Q = \sum_{a} (V_1(a) \cdot V_4(a)) \qquad \mathcal{O}(N)$$

Factor Payloads Enable Different Aggregates (1/2)

Task: Compute $SUM(C \cdot D)$ over the join of R, S, and T

Factor Payloads Enable Different Aggregates (1/2)

Task: Compute $SUM(C \cdot D)$ over the join of R, S, and T

- Use the same query:

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

Factor Payloads Enable Different Aggregates (1/2)

Task: Compute $SUM(C \cdot D)$ over the join of R, S, and T

- Use the same query:

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

- Use factors with different payloads, e.g.,:
  - $\mathrm{R}(\mathrm{a}, \mathrm{b}) = 1$ if $(a, b) \in R$ and 0 otherwise
  - $\mathrm{S}(\mathrm{a}, \mathrm{c}, \mathrm{e}) = c$ if $(a, c, e) \in S$ and 0 otherwise
  - $\mathrm{T}(\mathrm{c}, \mathrm{d}) = d$ if $(c, d) \in T$ and 0 otherwise

- For every tuple (a, b, c, d, e),
  we sum up the products $\mathrm{R}(\mathrm{a}, \mathrm{b}) \cdot \mathrm{S}(\mathrm{a}, \mathrm{c}, \mathrm{e}) \cdot \mathrm{T}(\mathrm{c}, \mathrm{d})$

Factor Payloads Enable Different Aggregates (2/2)

- Factors R, S, and T with payloads from a ring $(\mathbf{D}, +, *, \mathbf{0}, \mathbf{1})$
  - Each tuple has the payload $\mathbf{1} \in \mathbf{D}$

- Lift factors Λ_A, Λ_B, Λ_C, Λ_D, Λ_E map the domain of a variable to $\mathbf{D}$

COUNT	all lift factors map to $1 \in \mathbb{Z}$
SUM(C*D)	$\Lambda_C[c] = c \in \mathbb{R}$ and $\Lambda_D[d] = d \in \mathbb{R}$; all others map to $1 \in \mathbb{R}$

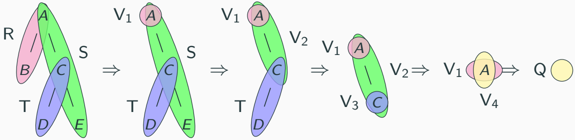
Factor Payloads Enable Different Aggregates (2/2)

- Factors R, S, and T with payloads from a ring $(\mathbf{D}, +, *, \mathbf{0}, \mathbf{1})$
  - Each tuple has the payload $\mathbf{1} \in \mathbf{D}$

- Lift factors Λ_A, Λ_B, Λ_C, Λ_D, Λ_E map the domain of a variable to **D**

COUNT	all lift factors map to $1 \in \mathbb{Z}$
SUM(C*D)	$\Lambda_C[c] = c \in \mathbb{R}$ and $\Lambda_D[d] = d \in \mathbb{R}$; all others map to $1 \in \mathbb{R}$

- Lift values of a variable just before its marginalization



$V_1(a) = \sum_b (R(a,b) \cdot \Lambda_B(b))$ $\quad$ $V_2(a,c) = \sum_e (S(a,c,e) \cdot \Lambda_E(e))$

$V_3(c) = \sum_d (T(c,d) \cdot \Lambda_D(d))$ $\quad$ $V_4(a) = \sum_c (V_2(a,c) \cdot V_3(c) \cdot \Lambda_C(c))$

$Q = \sum_a (V_1(a) \cdot V_4(a) \cdot \Lambda_A(a))$

Update Example

- Database of three factors R, S, T

R

A	B	#
a_1	b_1	2
a_2	b_1	3

S

B	C	#
b_1	c_1	2
b_1	c_2	1

T

C	A	#
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

Update Example

- Database of three factors R, S, T

R

A	B	$\#$
a_1	b_1	2
a_2	b_1	3

S

B	C	$\#$
b_1	c_1	2
b_1	c_2	1

T

C	A	$\#$
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	$\#$
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$

Update Example

- Database of three factors R, S, T

R

A	B	#
a_1	b_1	2
a_2	b_1	3

S

B	C	#
b_1	c_1	2
b_1	c_2	1

T

C	A	#
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	#
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

Update Example

- Database of three factors R, S, T
- Triangle Count Query: $Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$

R

A	B	$\#$
a_1	b_1	2
a_2	b_1	3

S

B	C	$\#$
b_1	c_1	2
b_1	c_2	1

T

C	A	$\#$
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	$\#$
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

$\downarrow$

Q

$\emptyset$	$\#$
$()$	$4 + 6 + 9 = 19$

Update Example

- Database of three factors R, S, T
- 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 factor mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

R

A	B	#
a_1	b_1	2
a_2	b_1	3

S

B	C	#
b_1	c_1	2
b_1	c_2	1

T

C	A	#
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	#
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↑

↓

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

A	B	#
a_2	b_1	-2

Q

$\emptyset$	#
$()$	$4 + 6 + 9 = 19$

Update Example

- Database of three factors R, S, T
- 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 factor mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

R

A	B	$\#$
a_1	b_1	2
a_2	b_1	3

S

B	C	$\#$
b_1	c_1	2
b_1	c_2	1

T

C	A	$\#$
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	$\#$
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↑

↓

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

A	B	$\#$
a_2	b_1	-2

Q

$\emptyset$	$\#$
$()$	$4 + 6 + 9 = 19$

Update Example

- Database of three factors R, S, T
- 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 factor mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

R

A	B	$\#$
a_1	b_1	2
~~a_2~~	~~b_1~~	~~3~~
a_2	b_1	1

S

B	C	$\#$
b_1	c_1	2
b_1	c_2	1

T

C	A	$\#$
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	$\#$
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↑

↓

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

A	B	$\#$
a_2	b_1	-2

Q

$\emptyset$	$\#$
$()$	$4 + 6 + 9 = 19$

Update Example

- Database of three factors R, S, T
- 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 factor mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

R

A	B	$\#$
a_1	b_1	2
~~a_2~~	~~b_1~~	~~3~~
a_2	b_1	1

S

B	C	$\#$
b_1	c_1	2
b_1	c_2	1

T

C	A	$\#$
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	$\#$
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↑

↓

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

A	B	$\#$
a_2	b_1	-2

Q

$\emptyset$	$\#$
$(\,)$	$4 + 6 + 9 = 19$

Update Example

- Database of three factors R, S, T
- 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 factor mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

R

A	B	$\#$
a_1	b_1	2
~~a_2~~	~~b_1~~	~~3~~
a_2	b_1	1

S

B	C	$\#$
b_1	c_1	2
b_1	c_2	1

T

C	A	$\#$
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	$\#$
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
~~a_2~~	~~b_1~~	~~c_2~~	~~$3 \cdot 1 \cdot 3 = 9$~~
a_2	b_1	c_2	$1 \cdot 1 \cdot 3 = 3$

↑

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

A	B	$\#$
a_2	b_1	-2

↓

Q

$\emptyset$	$\#$
$(\,)$	$4 + 6 + 9 = 19$

Update Example

- Database of three factors R, S, T
- 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 factor mapping a tuple to a non-zero value (positive for insertions, negative for deletions)

R

A	B	#
a_1	b_1	2
~~a_2~~	~~b_1~~	~~3~~
a_2	b_1	1

S

B	C	#
b_1	c_1	2
b_1	c_2	1

T

C	A	#
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \cdot S \cdot T$

A	B	C	#
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
~~a_2~~	~~b_1~~	~~c_2~~	~~$3 \cdot 1 \cdot 3 = 9$~~
a_2	b_1	c_2	$1 \cdot 1 \cdot 3 = 3$

↑

↓

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

A	B	#
a_2	b_1	-2

Q

$\emptyset$	#
~~( )~~	~~$4+6+9=19$~~
( )	$4+6+3=13$

Part 2.

Main IVM techniques:

First-Order IVM

Incremental Computation for Queries

- Input: Query Q, database D, updates δD to D
- Task: Maintain the query result $Q(D)$ under changes δD

$$Q(D + \delta D) = Q(D) + \delta Q(D, \delta D)$$

IVM is faster than re-computation when:

- The "merge" operation of the two query results is fast

  Upserts into hashmaps, appends to lists

- Delta query δQ is faster

  Lower computational complexity, less data input and output

Incremental Computation for Queries

- Input: Query Q, database D, updates δD to D
- Task: Maintain the query result $Q(D)$ under changes δD

$$Q(D + \delta D) = Q(D) + \delta Q(D, \delta D)$$

IVM is faster than re-computation when:

- The "merge" operation of the two query results is fast

  Upserts into hashmaps, appends to lists

- Delta query δQ is faster

  Lower computational complexity, less data input and output

Question: How can we express the delta queries?

Query Language Closed under Taking Deltas

Given:

- Query Q
- Update $\delta\mathsf{T}$ for factor T in Q

The delta query is defined on the structure of Q:

$$\delta(\mathsf{R} + \mathsf{S}) = \delta\mathsf{R} + \delta\mathsf{S}$$

$$\delta(\mathsf{R} \cdot \mathsf{S}) = (\delta\mathsf{R} \cdot \mathsf{S}) + (\mathsf{R} \cdot \delta\mathsf{S}) + (\delta\mathsf{R} \cdot \delta\mathsf{S})$$

$$\delta(\sum_{a} \mathsf{R}) = \sum_{a} \delta\mathsf{R}$$

$$\delta\mathsf{R} = \begin{cases} \delta\mathsf{T} & \text{if } \mathsf{R} = \mathsf{T} \\ 0 & \text{otherwise // Factor 0 maps empty tuple to value 0} \end{cases}$$

Example: Delta for Simple Join Query

Given: Factors R and S of size N

- Query:

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

Evaluation time: $\mathcal{O}(N)$

Example: Delta for Simple Join Query

Given: Factors R and S of size N

- Query:

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

Evaluation time: $\mathcal{O}(N)$

- Delta queries:

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

Single-tuple update: $\mathcal{O}(1)$ for $\delta R(a,b)$ and $\mathcal{O}(N)$ for $\delta S(b)$
Enumeration delay: $\mathcal{O}(1)$

Example: Delta for Simple Join Query

Given: Factors R and S of size N

- Query:

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

  Evaluation time: $\mathcal{O}(N)$

- Delta queries:

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

  Single-tuple update: $\mathcal{O}(1)$ for $\delta R(a, b)$ and $\mathcal{O}(N)$ for $\delta S(b)$
  Enumeration delay: $\mathcal{O}(1)$

  Optimal: $\mathcal{O}(1)$ single-tuple update time and enumeration delay

Example: Delta for Simple Project-Join Query

Given: Factors R and S of size N

- Query:

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

Evaluation time: $\mathcal{O}(N)$

Example: Delta for Simple Project-Join Query

Given: Factors R and S of size N

- Query:

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

Evaluation time: $\mathcal{O}(N)$

- Delta queries:

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

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

Single-tuple update time: $\mathcal{O}(1)$ for δR and $\mathcal{O}(N)$ for δS

Enumeration delay: $\mathcal{O}(1)$

Example: Delta for Simple Project-Join Query

Given: Factors R and S of size N

- Query:

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

Evaluation time: $\mathcal{O}(N)$

- Delta queries:

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

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

Single-tuple update time: $\mathcal{O}(1)$ for δR and $\mathcal{O}(N)$ for δS

Enumeration delay: $\mathcal{O}(1)$

Optimal: $\mathcal{O}(\sqrt{N})$ single-tuple update time and enumeration delay

Example: Delta for Triangle Join Query (1/3)

Given: Graph with edge factor E, update δE

- Query:

$$Q(a, b, c) = E(a, b) \cdot E(b, c) \cdot E(c, a)$$

Evaluation time: $\mathcal{O}(|E|^{1.5})$ (clarified later in the course)

Example: Delta for Triangle Join Query (1/3)

Given: Graph with edge factor E, update δE

- Query:

$$Q(a,b,c) = E(a,b) \cdot E(b,c) \cdot E(c,a)$$

Evaluation time: $\mathcal{O}(|E|^{1.5})$ (clarified later in the course)

- Delta query:

$$\begin{aligned}\delta Q(a,b,c) &= \delta\left(E(a,b) \cdot E(b,c) \cdot E(c,a)\right)\\ &= \delta E(a,b) \cdot E(b,c) \cdot E(c,a) + E(a,b) \cdot \delta\left(E(b,c) \cdot E(c,a)\right)\\ &+ \delta E(a,b) \cdot \delta\left(E(b,c) \cdot E(c,a)\right)\end{aligned}$$

We next expand:

$$\delta\left(E(b,c) \cdot E(c,a)\right) = \delta E(b,c) \cdot E(c,a) + E(b,c) \cdot \delta E(c,a) + \delta E(b,c) \cdot \delta E(c,a)$$

Example: Delta for Triangle Join Query (2/3)

$$
\begin{aligned}
\delta Q(a,b,c) = {} & \delta E(a,b) \cdot E(b,c) \cdot E(c,a) + E(a,b) \cdot \delta E(b,c) \cdot E(c,a) \\
& + E(a,b) \cdot E(b,c) \cdot \delta E(c,a) + E(a,b) \cdot \delta E(b,c) \cdot \delta E(c,a) \\
& + \delta E(a,b) \cdot \delta E(b,c) \cdot E(c,a) + \delta E(a,b) \cdot E(b,c) \cdot \delta E(c,a) \\
& + \delta E(a,b) \cdot \delta E(b,c) \cdot \delta E(c,a)
\end{aligned}
$$

Example: Delta for Triangle Join Query (2/3)

$$\begin{aligned}\delta Q(a,b,c) = {} & \delta E(a,b) \cdot E(b,c) \cdot E(c,a) + E(a,b) \cdot \delta E(b,c) \cdot E(c,a) \\ & + E(a,b) \cdot E(b,c) \cdot \delta E(c,a) + E(a,b) \cdot \delta E(b,c) \cdot \delta E(c,a) \\ & + \delta E(a,b) \cdot \delta E(b,c) \cdot E(c,a) + \delta E(a,b) \cdot E(b,c) \cdot \delta E(c,a) \\ & + \delta E(a,b) \cdot \delta E(b,c) \cdot \delta E(c,a)\end{aligned}$$

Can we simplify this query?

Example: Delta for Triangle Join Query (2/3)

$$\begin{aligned}\delta Q(a,b,c) = {} & \delta E(a,b)\cdot E(b,c)\cdot E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot E(c,a)\\ & + E(a,b)\cdot E(b,c)\cdot \delta E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot E(c,a) + \delta E(a,b)\cdot E(b,c)\cdot \delta E(c,a)\\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Can we simplify this query?

Yes! Idea: Q is symmetric, so

$$\begin{aligned}\delta E(a,b)\cdot E(b,c)\cdot E(c,a) &= E(a,b)\cdot \delta E(b,c)\cdot E(c,a)\\ &= E(a,b)\cdot E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Example: Delta for Triangle Join Query (3/3)

$$\begin{aligned}\delta Q(a,b,c) = {} & \delta E(a,b)\cdot E(b,c)\cdot E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot E(c,a) \\ & + E(a,b)\cdot E(b,c)\cdot \delta E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a) \\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot E(c,a) + \delta E(a,b)\cdot E(b,c)\cdot \delta E(c,a) \\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Let 3 be a factor mapping the empty tuple to multiplicity 3. Then,

$$\begin{aligned}\delta Q(a,b,c) = {} & 3\cdot \delta E(a,b)\cdot E(b,c)\cdot E(c,a) \\ & + 3\cdot E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a) \\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Example: Delta for Triangle Join Query (3/3)

$$\begin{aligned}\delta Q(a,b,c) = {} & \delta E(a,b)\cdot E(b,c)\cdot E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot E(c,a)\\ & + E(a,b)\cdot E(b,c)\cdot \delta E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot E(c,a) + \delta E(a,b)\cdot E(b,c)\cdot \delta E(c,a)\\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Let 3 be a factor mapping the empty tuple to multiplicity 3. Then,

$$\begin{aligned}\delta Q(a,b,c) = {} & 3\cdot \delta E(a,b)\cdot E(b,c)\cdot E(c,a)\\ & + 3\cdot E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Single-tuple update time: $\mathcal{O}(|E|)$ Enumeration delay: $\mathcal{O}(1)$

Example: Delta for Triangle Join Query (3/3)

$$\begin{aligned}\delta Q(a,b,c) = {} & \delta E(a,b)\cdot E(b,c)\cdot E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot E(c,a) \\ & + E(a,b)\cdot E(b,c)\cdot \delta E(c,a) + E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a) \\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot E(c,a) + \delta E(a,b)\cdot E(b,c)\cdot \delta E(c,a) \\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Let 3 be a factor mapping the empty tuple to multiplicity 3. Then,

$$\begin{aligned}\delta Q(a,b,c) = {} & 3\cdot \delta E(a,b)\cdot E(b,c)\cdot E(c,a) \\ & + 3\cdot E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a) \\ & + \delta E(a,b)\cdot \delta E(b,c)\cdot \delta E(c,a)\end{aligned}$$

Single-tuple update time: $\mathcal{O}(|E|)$ Enumeration delay: $\mathcal{O}(1)$

Optimal: $\mathcal{O}(\sqrt{|E|})$ single-tuple update time and $\mathcal{O}(1)$ delay

Example: Delta for Triangle Count Query

Graph with edge factor E, update δE

- Query:

$$Q = \sum_{a,b,c} E(a,b) \cdot E(b,c) \cdot E(c,a)$$

Evaluation time: $\mathcal{O}(|E|^{1.5})$. Strassen-like algorithm: $\mathcal{O}(|E|^{1.41})$

Example: Delta for Triangle Count Query

Graph with edge factor E, update δE

- Query:

$$Q = \sum_{a,b,c} E(a,b) \cdot E(b,c) \cdot E(c,a)$$

Evaluation time: $\mathcal{O}(|E|^{1.5})$. Strassen-like algorithm: $\mathcal{O}(|E|^{1.41})$

- Delta query:

$$\delta Q = \delta \left(\sum_{a,b,c} E(a,b) \cdot E(b,c) \cdot E(c,a) \right)$$

$$= \sum_{a,b,c} \delta \left(E(a,b) \cdot E(b,c) \cdot E(c,a) \right) = \ldots \text{ (as for the triangle join)}$$

Single-tuple update time: $\mathcal{O}(|E|)$ Enumeration delay: $\mathcal{O}(1)$

Example: Delta for Triangle Count Query

Graph with edge factor E, update δE

- Query:

$$Q = \sum_{a,b,c} E(a, b) \cdot E(b, c) \cdot E(c, a)$$

Evaluation time: $\mathcal{O}(|E|^{1.5})$. Strassen-like algorithm: $\mathcal{O}(|E|^{1.41})$

- Delta query:

$$\delta Q = \delta \left(\sum_{a,b,c} E(a, b) \cdot E(b, c) \cdot E(c, a) \right)$$

$$= \sum_{a,b,c} \delta \left(E(a, b) \cdot E(b, c) \cdot E(c, a) \right) = \ldots \text{ (as for the triangle join)}$$

Single-tuple update time: $\mathcal{O}(|E|)$ Enumeration delay: $\mathcal{O}(1)$

Optimal: $\mathcal{O}(\sqrt{|E|})$ single-tuple update time and $\mathcal{O}(1)$ delay

Example: Delta for Acyclic Query

Given: Factors R, S, T of size N

- Query: $Q(a, b, c) = \sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d)$

  Evaluation time: $\mathcal{O}(N^2)$

Example: Delta for Acyclic Query

Given: Factors R, S, T of size N

- Query: $Q(a,b,c) = \sum_{d,e} R(a,b) \cdot S(a,c,e) \cdot T(a,c,d)$

  Evaluation time: $\mathcal{O}(N^2)$

- Delta query for update δR (similar for δS and δT):

$$\begin{aligned}
\delta Q(a,b,c) &= \delta \left(\sum_{d,e} R(a,b) \cdot S(a,c,e) \cdot T(a,c,d) \right) \\
&= \sum_{d,e} \delta \left(R(a,b) \cdot S(a,c,e) \cdot T(a,c,d) \right) \\
&= \delta R(a,b) \cdot \left(\sum_{d} S(a,c,e) \right) \cdot \left(\sum_{c} T(a,c,d) \right)
\end{aligned}$$

Single-tuple update time: $\mathcal{O}(N)$. Enumeration delay: $\mathcal{O}(1)$

Example: Delta for Acyclic Query

Given: Factors R, S, T of size N

- Query: $Q(a, b, c) = \sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d)$

  Evaluation time: $\mathcal{O}(N^2)$

- Delta query for update δR (similar for δS and δT):

$$\begin{aligned} \delta Q(a, b, c) &= \delta \left(\sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d) \right) \\ &= \sum_{d,e} \delta \left(R(a, b) \cdot S(a, c, e) \cdot T(a, c, d) \right) \\ &= \delta R(a, b) \cdot \left(\sum_{d} S(a, c, e) \right) \cdot \left(\sum_{c} T(a, c, d) \right) \end{aligned}$$

Single-tuple update time: $\mathcal{O}(N)$. Enumeration delay: $\mathcal{O}(1)$

Optimal: $\mathcal{O}(1)$ single-tuple update time and delay

Beyond Delta Queries

Can we achieve
the aforementioned optimal update times
by using additional storage?

Part 2.

Main IVM techniques:

Higher-Order IVM

Using Materialized Views To Lower Update Time

We reconsider the following query with factors R and S of size N

- Query:

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

Using Materialized Views To Lower Update Time

We reconsider the following query with factors R and S of size N

- Query:

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

- Delta query for update $\delta S(b_0)$:

$$\delta Q(a, b_0) = R(a, b_0) \cdot \delta S(b_0)$$

Single-tuple update: $\mathcal{O}(N)$, Enumeration delay: $\mathcal{O}(1)$

Using Materialized Views To Lower Update Time

We reconsider the following query with factors R and S of size N

- Query:

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

- Delta query for update $\delta S(b_0)$:

$$\delta Q(a, b_0) = R(a, b_0) \cdot \delta S(b_0)$$

Single-tuple update: $\mathcal{O}(N)$, Enumeration delay: $\mathcal{O}(1)$

Using the views

$$V_R(b) = \sum_a R(a, b) \quad \text{and} \quad V_{RS}(b) = V_R(b) \cdot S(b)$$

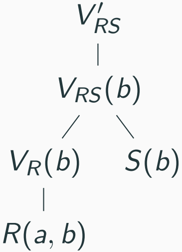
we achieve optimality:

Single-tuple update: $\mathcal{O}(1)$, Enumeration delay: $\mathcal{O}(1)$

Tree of Materialized Views

We organize the input factors and the extra views in a tree

View tree



V'_{RS}

$V_{RS}(b)$

$V_R(b)$ $S(b)$

$R(a, b)$

Views

$$V'_{RS} = \sum_b V_{RS}(b)$$

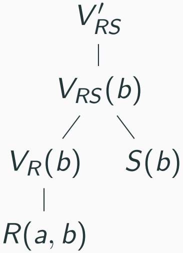
$$V_{RS}(b) = V_R(b) \cdot S(b)$$

$$V_R(b) = \sum_a R(a, b)$$

Computation time: $\mathcal{O}(N)$

Maintaining View Tree Under Updates (1/2)

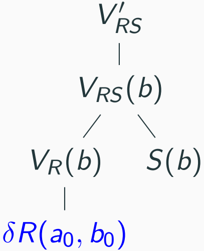
View tree



Single-tuple update $\delta R(a_0, b_0)$

Maintaining View Tree Under Updates (1/2)

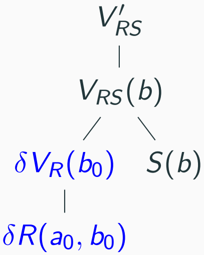
View tree



Single-tuple update $\delta R(a_0, b_0)$

Maintaining View Tree Under Updates (1/2)

View tree



V'_{RS}

$V_{RS}(b)$

$\delta V_R(b_0)$ $\quad S(b)$

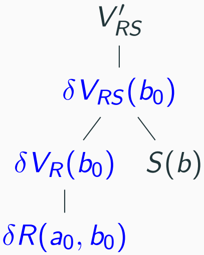
$\delta R(a_0, b_0)$

Single-tuple update $\delta R(a_0, b_0)$

$$\delta V_R(b_0) = \sum_{a_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

Maintaining View Tree Under Updates (1/2)

View tree



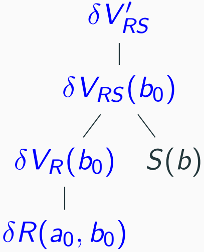
Single-tuple update $\delta R(a_0, b_0)$

$$\delta V_{RS}(b_0) = \delta V_R(b_0) \cdot S(b_0)$$

$$\delta V_R(b_0) = \sum_{a_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

Maintaining View Tree Under Updates (1/2)

View tree



Single-tuple update $\delta R(a_0, b_0)$

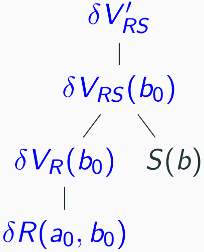
$$\delta V'_{RS} = \sum_{b_0} \delta V_{RS}(b_0) = V_{RS}(b_0)$$

$$\delta V_{RS}(b_0) = \delta V_R(b_0) \cdot S(b_0)$$

$$\delta V_R(b_0) = \sum_{a_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

Maintaining View Tree Under Updates (1/2)

View tree



$\delta V'_{RS}$

$\delta V_{RS}(b_0)$

$\delta V_R(b_0)$ $S(b)$

$\delta R(a_0, b_0)$

Single-tuple update $\delta R(a_0, b_0)$

$$\delta V'_{RS} = \sum_{b_0} \delta V_{RS}(b_0) = V_{RS}(b_0)$$

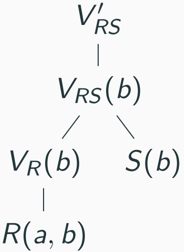
$$\delta V_{RS}(b_0) = \delta V_R(b_0) \cdot S(b_0)$$

$$\delta V_R(b_0) = \sum_{a_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

Update time: O(1)

Maintaining View Tree Under Updates (2/2)

View tree



V'_{RS}

$V_{RS}(b)$

$V_R(b)$

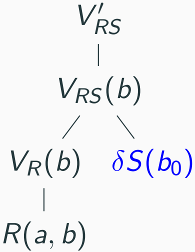
$S(b)$

$R(a,b)$

Single-tuple update $\delta S(b_0)$

Maintaining View Tree Under Updates (2/2)

View tree



V'_{RS}

$V_{RS}(b)$

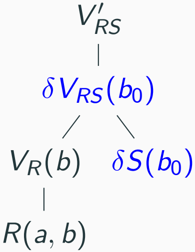
$V_R(b)$ $\delta S(b_0)$

$R(a, b)$

Single-tuple update $\delta S(b_0)$

Maintaining View Tree Under Updates (2/2)

View tree



V'_{RS}

$\delta V_{RS}(b_0)$

$V_R(b)$ $\delta S(b_0)$

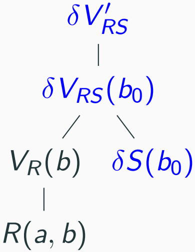
$R(a, b)$

Single-tuple update $\delta S(b_0)$

$$\delta V_{RS}(b_0) = \delta V_R(b_0) \cdot S(b_0)$$

Maintaining View Tree Under Updates (2/2)

View tree



$\delta V'_{RS}$

$\delta V_{RS}(b_0)$

$V_R(b)$ $\delta S(b_0)$

$R(a, b)$

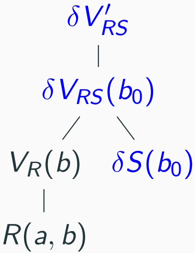
Single-tuple update $\delta S(b_0)$

$$\delta V'_{RS} = \sum_{b_0} \delta V_{RS}(b_0) = V_{RS}(b_0)$$

$$\delta V_{RS}(b_0) = \delta V_R(b_0) \cdot S(b_0)$$

Maintaining View Tree Under Updates (2/2)

View tree



Single-tuple update $\delta S(b_0)$

$$\delta V'_{RS} = \sum_{b_0} \delta V_{RS}(b_0) = V_{RS}(b_0)$$

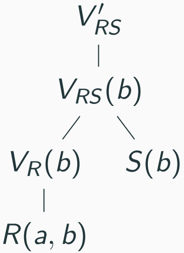
$$\delta V_{RS}(b_0) = \delta V_R(b_0) \cdot S(b_0)$$

Update time: O(1)

Enumeration from View Tree

Enumeration

View tree

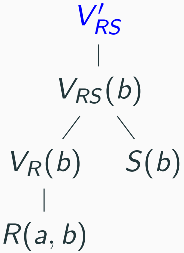


Enumeration from View Tree

Enumeration

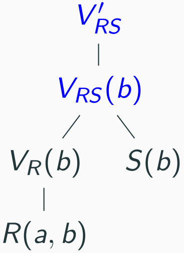
- Is V'_{RS} empty? If yes, stop!

View tree



Enumeration from View Tree

View tree

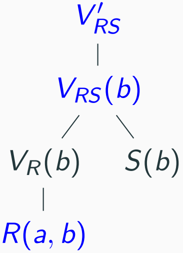


Enumeration

- Is V'_{RS} empty? If yes, stop!
- Iterate over (distinct) b's in V_{RS}

Enumeration from View Tree

View tree



V'_{RS}

$V_{RS}(b)$

$V_R(b)$ $S(b)$

$R(a,b)$

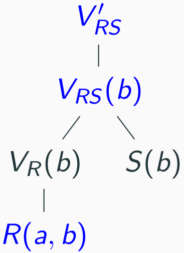
Enumeration

- Is V'_{RS} empty? If yes, stop!
- Iterate over (distinct) b's in V_{RS}
- For each b, iterate over (distinct) a's in $R(a,b)$

  Requires an index $b \mapsto a$ over R

Enumeration from View Tree

View tree



V'_{RS}

$V_{RS}(b)$

$V_R(b)$ $S(b)$

$R(a, b)$

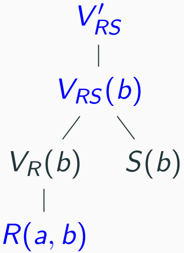
Enumeration

- Is V'_{RS} empty? If yes, stop!
- Iterate over (distinct) b's in V_{RS}
- For each b, iterate over (distinct) a's in $R(a, b)$

  Requires an index $b \mapsto a$ over R
- Output (b, a)

Enumeration from View Tree

View tree



Enumeration

- Is V'_{RS} empty? If yes, stop!
- Iterate over (distinct) b's in V_{RS}
- For each b, iterate over (distinct) a's in $R(a, b)$

  Requires an index $b \mapsto a$ over R
- Output (b, a)

Enumeration delay: $\mathcal{O}(1)$

How to Generalize This Example?

We need to construct view trees for the query

- Similar to query plans for static query evaluation
- Goal: Minimize the propagation time for each update

We next sketch two main approaches

- DBToaster `https://dbtoaster.github.io`
- F-IVM `https://github.com/fdbresearch/FIVM`

How to Generalize This Example?

We need to construct view trees for the query

- Similar to query plans for static query evaluation
- Goal: Minimize the propagation time for each update

We next sketch two main approaches

- DBToaster `https://dbtoaster.github.io`
    1. One view tree per updatable factor, which is child of root
    2. Materialized view over each connected component of the remaining factors
    3. This view exports the join and head variables
    4. This view is treated like a query (Go to 1)
- F-IVM `https://github.com/fdbresearch/FIVM`

How to Generalize This Example?

We need to construct view trees for the query

- Similar to query plans for static query evaluation
- Goal: Minimize the propagation time for each update

We next sketch two main approaches

- DBToaster `https://dbtoaster.github.io`
    1. One view tree per updatable factor, which is child of root
    2. Materialized view over each connected component of the remaining factors
    3. This view exports the join and head variables
    4. This view is treated like a query (Go to 1)
- F-IVM `https://github.com/fdbresearch/FIVM`
    1. One view tree for all updatable factors
    2. Structure of view tree follows a partial order of query variables
    3. Query result possibly distributed over several views (previous example)

Higher-Order IVM using DBToaster (1/10)

Consider again the factors R, S, T of size N and the query

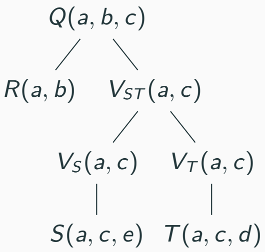
$$Q(a, b, c) = \sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d)$$

Higher-Order IVM using DBToaster (1/10)

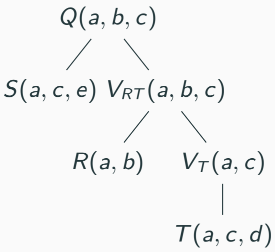
Consider again the factors R, S, T of size N and the query

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

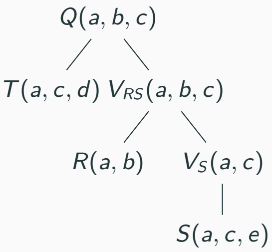
DBToaster constructs a view tree for each factor:



For update δR to R



For update δS to S



For update δT to T

Higher-Order IVM using DBToaster (2/10)

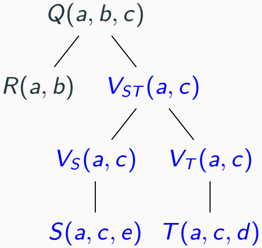
$$Q(a,b,c) = \sum_{d,e} R(a,b) \cdot S(a,c,e) \cdot T(a,c,d)$$

For update δR to R:

DBToaster creates the materialized view V_{ST}

$$\begin{aligned} V_{ST}(a,c) &= \sum_{d,e} S(a,c,e) \cdot T(a,c,d) \\ &= \sum_{e} S(a,c,e) \cdot \sum_{d} T(a,c,d) \\ &= V_S(a,c) \cdot V_T(a,c) \end{aligned}$$

Computation time: $\mathcal{O}(N)$



$Q(a,b,c)$

$R(a,b)$ $V_{ST}(a,c)$

$V_S(a,c)$ $V_T(a,c)$

$S(a,c,e)$ $T(a,c,d)$

Higher-Order IVM using DBToaster (2/10)

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

For update δR to R:

DBToaster creates the materialized view V_{ST}

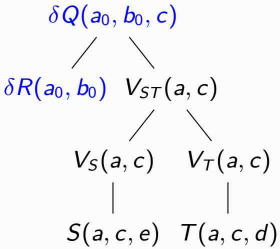
$$\begin{aligned} V_{ST}(a,c) &= \sum_{d,e} S(a,c,e) \cdot T(a,c,d) \\ &= \sum_{e} S(a,c,e) \cdot \sum_{d} T(a,c,d) \\ &= V_S(a,c) \cdot V_T(a,c) \end{aligned}$$

Computation time: $\mathcal{O}(N)$

The delta query for update $\delta R(a_0, b_0)$ using V_{ST} is then:

$$\delta Q(a,b,c) = \delta R(a_0,b_0) \cdot V_{ST}(a_0,c)$$

Single-tuple update time: $\mathcal{O}(N)$



Higher-Order IVM using DBToaster (3/10)

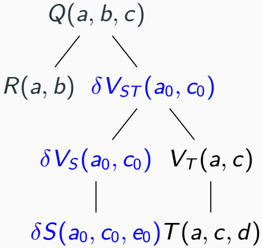
$$Q(a,b,c) = \sum_{d,e} R(a,b) \cdot S(a,c,e) \cdot T(a,c,d)$$

The materialized view V_{ST} needs to be updated, too

Delta query for update $\delta S(a_0, c_0, e_0)$:

$$\delta V_S(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V_{ST}(a_0, c_0) = \delta V_S(a_0, c_0) \cdot V_T(a_0, c_0)$$



$Q(a,b,c)$

$R(a,b)$ $\delta V_{ST}(a_0, c_0)$

$\delta V_S(a_0, c_0)$ $V_T(a,c)$

$\delta S(a_0, c_0, e_0)$ $T(a,c,d)$

Higher-Order IVM using DBToaster (3/10)

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

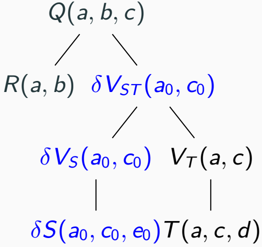
The materialized view V_{ST} needs to be updated, too

Delta query for update $\delta S(a_0, c_0, e_0)$:

$$\delta V_S(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V_{ST}(a_0, c_0) = \delta V_S(a_0, c_0) \cdot V_T(a_0, c_0)$$

Single-tuple update time: $\mathcal{O}(1)$



$Q(a, b, c)$

$R(a, b)$ $\delta V_{ST}(a_0, c_0)$

$\delta V_S(a_0, c_0)$ $V_T(a, c)$

$\delta S(a_0, c_0, e_0)$ $T(a, c, d)$

Higher-Order IVM using DBToaster (4/10)

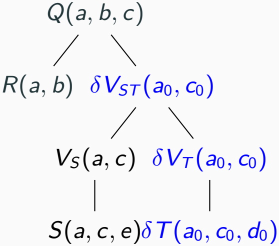
$$Q(a,b,c) = \sum_{d,e} R(a,b) \cdot S(a,c,e) \cdot T(a,c,d)$$

The materialized view V_{ST} needs to be updated, too

Delta query for update $\delta T(a_0, c_0, d_0)$:

$$\delta V_T(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V_{ST}(a_0, c_0) = V_S(a_0, c_0) \cdot \delta V_T(a_0, c_0)$$



$Q(a,b,c)$

$R(a,b)$ $\delta V_{ST}(a_0, c_0)$

$V_S(a,c)$ $\delta V_T(a_0, c_0)$

$S(a,c,e)$ $\delta T(a_0, c_0, d_0)$

Higher-Order IVM using DBToaster (4/10)

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

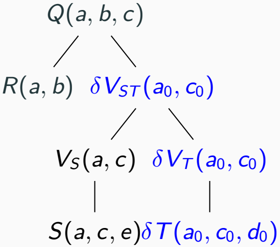
The materialized view V_{ST} needs to be updated, too

Delta query for update $\delta T(a_0, c_0, d_0)$:

$$\delta V_T(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V_{ST}(a_0, c_0) = V_S(a_0, c_0) \cdot \delta V_T(a_0, c_0)$$

Single-tuple update time: $\mathcal{O}(1)$



Higher-Order IVM using DBToaster (5/10)

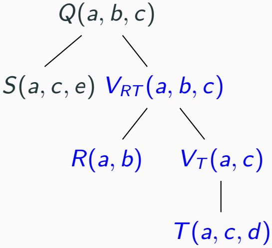
$$Q(a,b,c) = \sum_{d,e} R(a,b) \cdot S(a,c,e) \cdot T(a,c,d)$$

For update δS to S:

DBToaster creates the materialized view V_{RT}

$$\begin{aligned} V_{RT}(a,b,c) &= \sum_{d} R(a,b) \cdot T(a,c,d) \\ &= R(a,b) \cdot \sum_{d} T(a,c,d) \\ &= R(a,b) \cdot V_{T}(a,c) \end{aligned}$$

Computation time: $\mathcal{O}(N^2)$



$Q(a,b,c)$

$S(a,c,e)$ $V_{RT}(a,b,c)$

$R(a,b)$ $V_{T}(a,c)$

$T(a,c,d)$

Higher-Order IVM using DBToaster (5/10)

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

For update δS to S:

DBToaster creates the materialized view V_{RT}

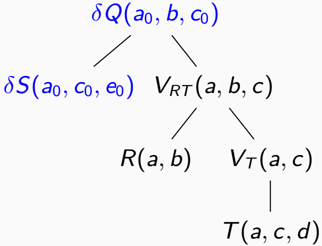
$$\begin{aligned} V_{RT}(a, b, c) &= \sum_{d} R(a, b) \cdot T(a, c, d) \\ &= R(a, b) \cdot \sum_{d} T(a, c, d) \\ &= R(a, b) \cdot V_T(a, c) \end{aligned}$$

Computation time: $\mathcal{O}(N^2)$

The delta query for update $\delta S(a_0, c_0, e_0)$ using V_{RT} is:

$$\begin{aligned} \delta Q(a_0, b, c_0) &= \sum_{e_0} \delta S(a_0, c_0, e_0) \cdot V_{RT}(a_0, b, c_0) \\ &= \delta S(a_0, c_0, e_0) \cdot V_{RT}(a_0, b, c_0) \end{aligned}$$

Single-tuple update time: $\mathcal{O}(N)$



$\delta Q(a_0, b, c_0)$

$\delta S(a_0, c_0, e_0)$ $V_{RT}(a, b, c)$

$R(a, b)$ $V_T(a, c)$

$T(a, c, d)$

Higher-Order IVM using DBToaster (6/10)

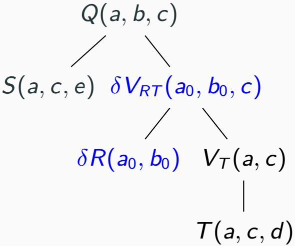
$$Q(a, b, c) = \sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d)$$

The materialized view V_{RT} needs to be updated, too

Delta query for update $\delta R(a_0, b_0)$:

$$\delta V_{RT}(a_0, b_0, c) = \delta R(a_0, b_0) \cdot V_T(a_0, c)$$

Single-tuple update time: $\mathcal{O}(N)$



$Q(a, b, c)$

$S(a, c, e)$ $\delta V_{RT}(a_0, b_0, c)$

$\delta R(a_0, b_0)$ $V_T(a, c)$

$T(a, c, d)$

Higher-Order IVM using DBToaster (7/10)

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

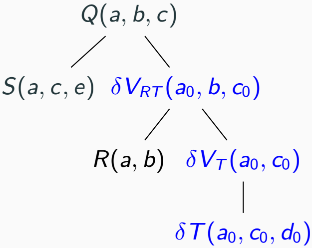
The materialized view V_{RT} needs to be updated, too

Delta queries for update $\delta T(a_0, c_0, d_0)$:

$$\delta V_T(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V_{RT}(a_0, b, c_0) = R(a_0, b) \cdot \delta V_T(a_0, c_0)$$

Single-tuple update time: $\mathcal{O}(1)$ and $\mathcal{O}(N)$, respectively



Higher-Order IVM using DBToaster (8/10)

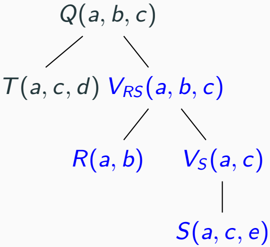
$$Q(a, b, c) = \sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d)$$

For update δT to T (analogous to update to S):

DBToaster creates the materialized view V_{RS}

$$\begin{aligned} V_{RS}(a, b, c) &= \sum_{e} R(a, b) \cdot S(a, c, e) \\ &= R(a, b) \cdot \sum_{e} S(a, c, e) \\ &= R(a, b) \cdot V_S(a, c) \end{aligned}$$

Computation time: $\mathcal{O}(N^2)$



$Q(a, b, c)$

$T(a, c, d)$ $V_{RS}(a, b, c)$

$R(a, b)$ $V_S(a, c)$

$S(a, c, e)$

Higher-Order IVM using DBToaster (8/10)

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

For update δT to T (analogous to update to S):

DBToaster creates the materialized view V_{RS}

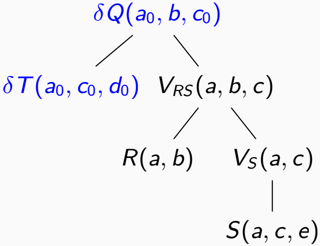
$$\begin{aligned} V_{RS}(a, b, c) &= \sum_{e} R(a, b) \cdot S(a, c, e) \\ &= R(a, b) \cdot \sum_{e} S(a, c, e) \\ &= R(a, b) \cdot V_S(a, c) \end{aligned}$$

Computation time: $\mathcal{O}(N^2)$

The delta query for update $\delta T(a_0, c_0, d_0)$ using V_{RS}:

$$\begin{aligned} \delta Q(a_0, b, c_0) &= \sum_{d_0} \delta T(a_0, c_0, d_0) \cdot V_{RS}(a_0, b, c_0) \\ &= \delta T(a_0, c_0, d_0) \cdot V_{RS}(a_0, b, c_0) \end{aligned}$$

Single-tuple update time: $\mathcal{O}(N)$



Higher-Order IVM using DBToaster (9/10)

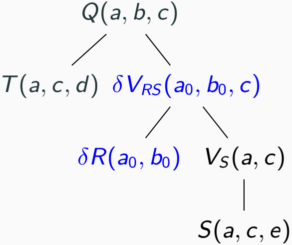
$$Q(a, b, c) = \sum_{d,e} R(a, b) \cdot S(a, c, e) \cdot T(a, c, d)$$

The materialized view V_{RS} needs to be updated, too

Delta query for update $\delta R(a_0, b_0)$:

$$\delta V_{RS}(a_0, b_0, c) = \delta R(a_0, b_0) \cdot V_S(a_0, c)$$

Single-tuple update time: $\mathcal{O}(N)$



$Q(a, b, c)$

$T(a, c, d)$ $\delta V_{RS}(a_0, b_0, c)$

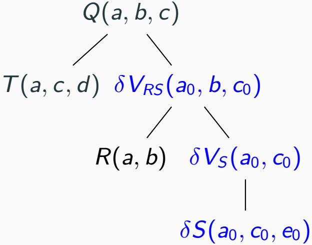
$\delta R(a_0, b_0)$ $V_S(a, c)$

$S(a, c, e)$

Higher-Order IVM using DBToaster (10/10)

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

The materialized view V_{RS} needs to be updated, too



Delta queries for update $\delta S(a_0, c_0, e_0)$:

$$\delta V_S(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V_{RS}(a_0, b, c_0) = R(a_0, b) \cdot \delta V_S(a_0, c_0)$$

Single-tuple update time: $\mathcal{O}(1)$ and $\mathcal{O}(N)$, respectively

DBToaster's Update Time Can Be Sub-Optimal

Two sources of sub-optimality (among others)

1. Too many views and view trees to maintain
2. The entire (super-linear) query result available at root views

DBToaster's Update Time Can Be Sub-Optimal

Two sources of sub-optimality (among others)

1. Too many views and view trees to maintain
2. The entire (super-linear) query result available at root views

How to overcome them?

1. One view tree for all updatable factors

   ⇒ Share views across updates

2. Keep the query result distributed over several views

   ⇒ Factorized representation of the query result

Both features are supported by F-IVM

Higher-Order IVM using F-IVM

F-IVM uses a partial order on the query variables:

- The variables of a factor are on the same path

Challenge: Find variable order that guarantees asymptotically minimal update time (among all variable orders)

- Hard problem already for static query evaluation, more in the FAQ lecture

Higher-Order IVM using F-IVM

F-IVM uses a partial order on the query variables:

- The variables of a factor are on the same path

Challenge: Find variable order that guarantees asymptotically minimal update time (among all variable orders)

- Hard problem already for static query evaluation, more in the FAQ lecture

F-IVM creates a view tree for a given variable order:

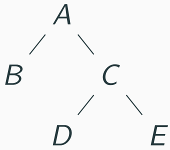
- Each variable induces extra views for join and marginalization
- The view joins the children views & marginalizes the variables
- The input factors are the leaves

Higher-Order IVM using F-IVM: Variable Order

Consider again the factors R, S, T of size N and the query

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

A possible variable order:

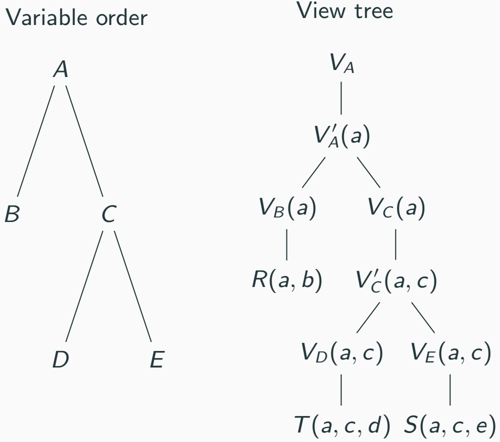


- A as root as it dominates all other variables
  - Dominate: It appears in all factors of the other variables
  - B is independent of the others given A
- D and E are under C as they are dominated by C
  - D and E are independent given C

Higher-Order IVM using F-IVM: View Tree

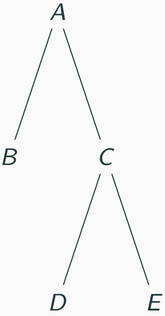
View tree construction:

- Place factors at leaves

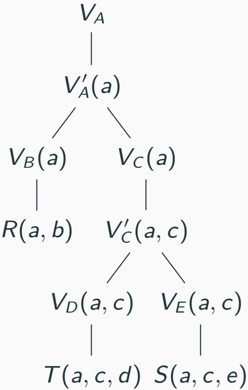


Higher-Order IVM using F-IVM: View Tree

Variable order



View tree



View tree construction:

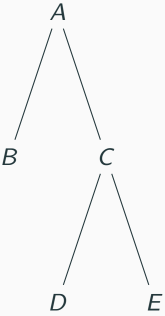
- Place factors at leaves
- Create parent view to join children

$$V'_C(a,c) = V_D(a,c) \cdot V_E(a,c)$$

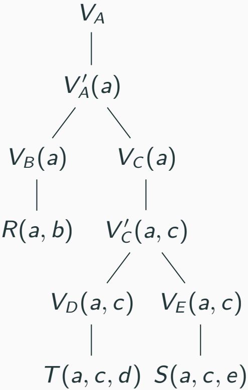
$$V'_A(a) = V_B(a) \cdot V_C(a)$$

Higher-Order IVM using F-IVM: View Tree

Variable order



View tree



View tree construction:

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

$$V'_C(a, c) = V_D(a, c) \cdot V_E(a, c)$$

$$V'_A(a) = V_B(a) \cdot V_C(a)$$

- Aggregate away variables not needed for further joins

$$V_A = \sum_a V'_A(a)$$

$$V_B(a) = \sum_b R(a, b)$$

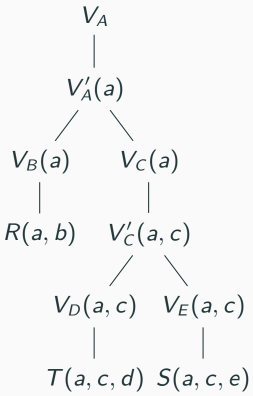
$$V_C(a) = \sum_c V'_C(a, c)$$

$$V_D(a, c) = \sum_d T(a, c, d)$$

$$V_E(a, c) = \sum_e S(a, c, e)$$

Higher-Order IVM using F-IVM: Updates (1/3)

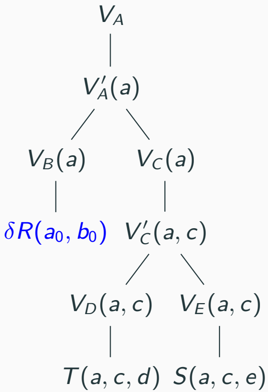
View tree



Single-tuple update $\delta R(a_0, b_0)$ to R:

Higher-Order IVM using F-IVM: Updates (1/3)

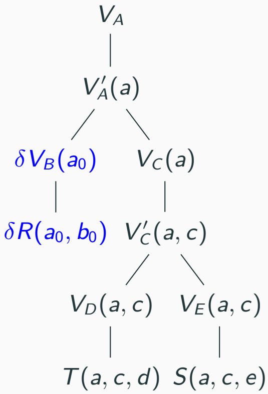
View tree



Single-tuple update $\delta R(a_0, b_0)$ to R:

Higher-Order IVM using F-IVM: Updates (1/3)

View tree

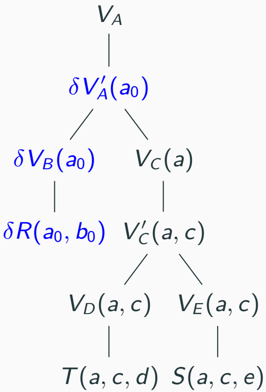


Single-tuple update $\delta R(a_0, b_0)$ to R:

$$\delta V_B(a_0) = \sum_{b_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

Higher-Order IVM using F-IVM: Updates (1/3)

View tree



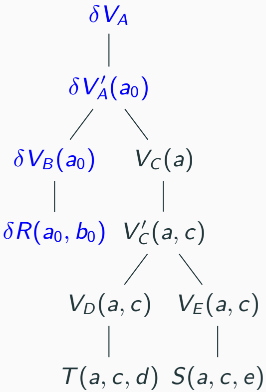
Single-tuple update $\delta R(a_0, b_0)$ to R:

$$\delta V_B(a_0) = \sum_{b_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

$$\delta V'_A(a_0) = \delta V_B(a_0) \cdot V_C(a_0)$$

Higher-Order IVM using F-IVM: Updates (1/3)

View tree



δV_A

$\delta V'_A(a_0)$

$\delta V_B(a_0)$ $V_C(a)$

$\delta R(a_0, b_0)$ $V'_C(a, c)$

$V_D(a, c)$ $V_E(a, c)$

$T(a, c, d)$ $S(a, c, e)$

Single-tuple update $\delta R(a_0, b_0)$ to R:

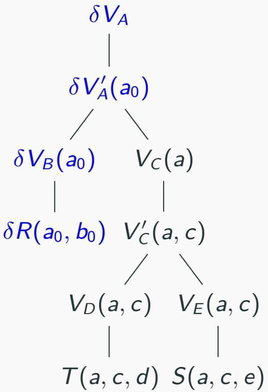
$$\delta V_B(a_0) = \sum_{b_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

$$\delta V'_A(a_0) = \delta V_B(a_0) \cdot V_C(a_0)$$

$$\delta V_A = \sum_{a_0} \delta V'_A(a_0) = \delta V'_A(a_0)$$

Higher-Order IVM using F-IVM: Updates (1/3)

View tree



Single-tuple update $\delta R(a_0, b_0)$ to R:

$$\delta V_B(a_0) = \sum_{b_0} \delta R(a_0, b_0) = \delta R(a_0, b_0)$$

$$\delta V'_A(a_0) = \delta V_B(a_0) \cdot V_C(a_0)$$

$$\delta V_A = \sum_{a_0} \delta V'_A(a_0) = \delta V'_A(a_0)$$

For each updated view/factor A:

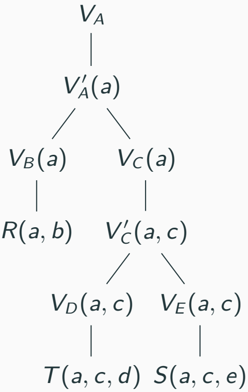
$$A := A + \delta A$$

Each view update takes $\mathcal{O}(1)$ time

Higher-Order IVM using F-IVM: Updates (2/3)

Single-tuple update $\delta S(a_0, c_0, e_0)$ **to** S**:**

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

$R(a, b)$ $V'_C(a, c)$

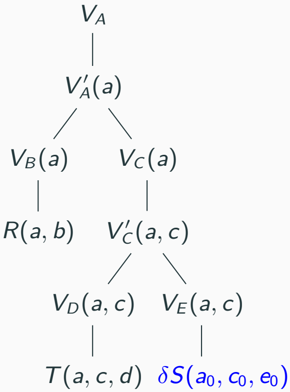
$V_D(a, c)$ $V_E(a, c)$

$T(a, c, d)$ $S(a, c, e)$

Higher-Order IVM using F-IVM: Updates (2/3)

Single-tuple update $\delta S(a_0, c_0, e_0)$ **to** S**:**

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

$R(a, b)$ $V'_C(a, c)$

$V_D(a, c)$ $V_E(a, c)$

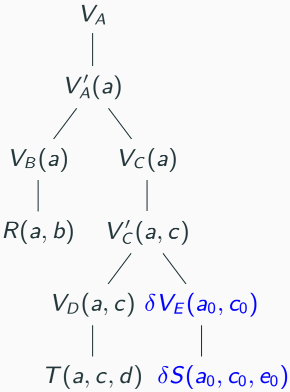
$T(a, c, d)$ $\delta S(a_0, c_0, e_0)$

Higher-Order IVM using F-IVM: Updates (2/3)

Single-tuple update $\delta S(a_0, c_0, e_0)$ to S:

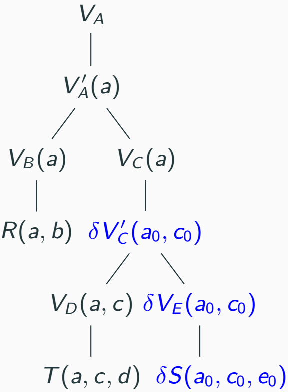
$$\delta V_E(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

View tree



Higher-Order IVM using F-IVM: Updates (2/3)

View tree



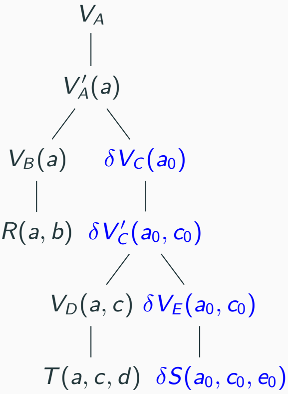
Single-tuple update $\delta S(a_0, c_0, e_0)$ to S:

$$\delta V_E(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V'_C(a_0, c_0) = \delta V_E(a_0, c_0) \cdot V_D(a_0, c_0)$$

Higher-Order IVM using F-IVM: Updates (2/3)

View tree



V_A

$V'_A(a)$

$V_B(a)$ $\delta V_C(a_0)$

$R(a,b)$ $\delta V'_C(a_0,c_0)$

$V_D(a,c)$ $\delta V_E(a_0,c_0)$

$T(a,c,d)$ $\delta S(a_0,c_0,e_0)$

Single-tuple update $\delta S(a_0, c_0, e_0)$ to S:

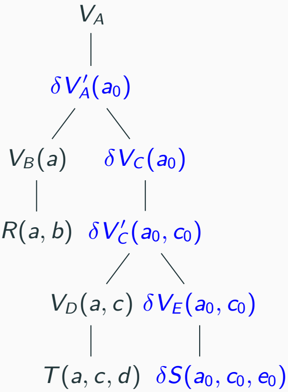
$$\delta V_E(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V'_C(a_0, c_0) = \delta V_E(a_0, c_0) \cdot V_D(a_0, c_0)$$

$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

Higher-Order IVM using F-IVM: Updates (2/3)

View tree



V_A

$\delta V'_A(a_0)$

$V_B(a)$

$\delta V_C(a_0)$

$R(a,b)$

$\delta V'_C(a_0,c_0)$

$V_D(a,c)$

$\delta V_E(a_0,c_0)$

$T(a,c,d)$

$\delta S(a_0,c_0,e_0)$

Single-tuple update $\delta S(a_0, c_0, e_0)$ to S:

$$\delta V_E(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

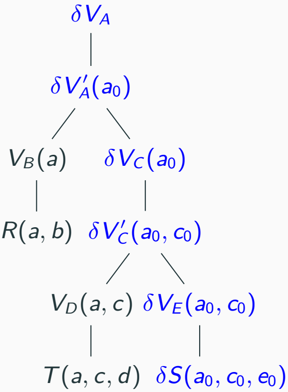
$$\delta V'_C(a_0, c_0) = \delta V_E(a_0, c_0) \cdot V_D(a_0, c_0)$$

$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

$$\delta V'_A(a_0) = \delta V_C(a_0) \cdot V_B(a_0)$$

Higher-Order IVM using F-IVM: Updates (2/3)

View tree



δV_A

$\delta V'_A(a_0)$

$V_B(a)$ $\delta V_C(a_0)$

$R(a, b)$ $\delta V'_C(a_0, c_0)$

$V_D(a, c)$ $\delta V_E(a_0, c_0)$

$T(a, c, d)$ $\delta S(a_0, c_0, e_0)$

Single-tuple update $\delta S(a_0, c_0, e_0)$ to S:

$$\delta V_E(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V'_C(a_0, c_0) = \delta V_E(a_0, c_0) \cdot V_D(a_0, c_0)$$

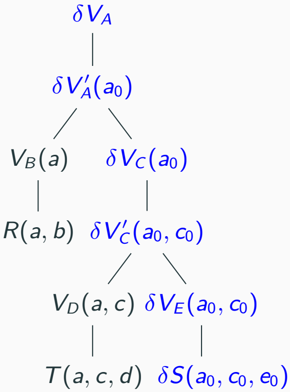
$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

$$\delta V'_A(a_0) = \delta V_C(a_0) \cdot V_B(a_0)$$

$$\delta V_A = \sum_{a_0} \delta V'_A(a_0) = \delta V'_A(a_0)$$

Higher-Order IVM using F-IVM: Updates (2/3)

View tree



Single-tuple update $\delta S(a_0, c_0, e_0)$ to S:

$$\delta V_E(a_0, c_0) = \sum_{e_0} \delta S(a_0, c_0, e_0) = \delta S(a_0, c_0, e_0)$$

$$\delta V'_C(a_0, c_0) = \delta V_E(a_0, c_0) \cdot V_D(a_0, c_0)$$

$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

$$\delta V'_A(a_0) = \delta V_C(a_0) \cdot V_B(a_0)$$

$$\delta V_A = \sum_{a_0} \delta V'_A(a_0) = \delta V'_A(a_0)$$

For each updated view/factor A:

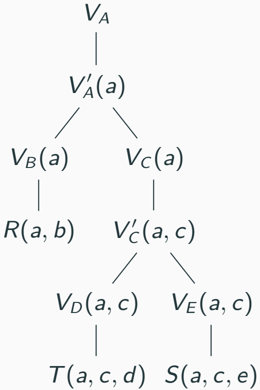
$$A := A + \delta A$$

Each view update takes $\mathcal{O}(1)$ time

Higher-Order IVM using F-IVM: Updates (3/3)

Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

$R(a, b)$ $V'_C(a, c)$

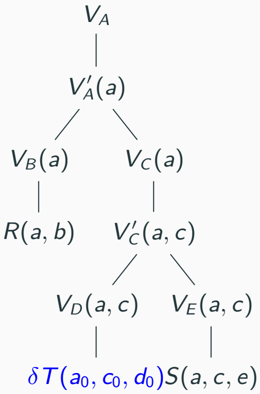
$V_D(a, c)$ $V_E(a, c)$

$T(a, c, d)$ $S(a, c, e)$

Higher-Order IVM using F-IVM: Updates (3/3)

Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

$R(a,b)$ $V'_C(a,c)$

$V_D(a,c)$ $V_E(a,c)$

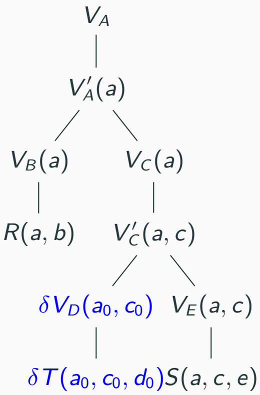
$\delta T(a_0, c_0, d_0)$ $S(a,c,e)$

Higher-Order IVM using F-IVM: Updates (3/3)

Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

$$\delta V_D(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

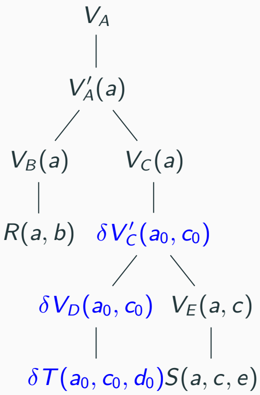
$R(a, b)$ $V'_C(a, c)$

$\delta V_D(a_0, c_0)$ $V_E(a, c)$

$\delta T(a_0, c_0, d_0)$ $S(a, c, e)$

Higher-Order IVM using F-IVM: Updates (3/3)

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

$R(a,b)$ $\delta V'_C(a_0, c_0)$

$\delta V_D(a_0, c_0)$ $V_E(a,c)$

$\delta T(a_0, c_0, d_0)$ $S(a,c,e)$

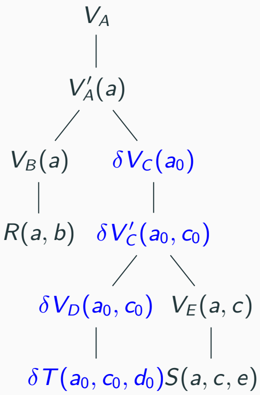
Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

$$\delta V_D(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V'_C(a_0, c_0) = V_E(a_0, c_0) \cdot \delta V_D(a_0, c_0)$$

Higher-Order IVM using F-IVM: Updates (3/3)

View tree



Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

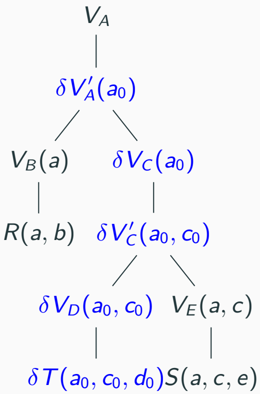
$$\delta V_D(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V'_C(a_0, c_0) = V_E(a_0, c_0) \cdot \delta V_D(a_0, c_0)$$

$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

Higher-Order IVM using F-IVM: Updates (3/3)

View tree



V_A

$\delta V'_A(a_0)$

$V_B(a)$ $\delta V_C(a_0)$

$R(a, b)$ $\delta V'_C(a_0, c_0)$

$\delta V_D(a_0, c_0)$ $V_E(a, c)$

$\delta T(a_0, c_0, d_0)$ $S(a, c, e)$

Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

$$\delta V_D(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

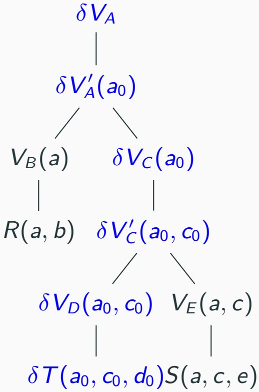
$$\delta V'_C(a_0, c_0) = V_E(a_0, c_0) \cdot \delta V_D(a_0, c_0)$$

$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

$$\delta V'_A(a_0) = \delta V_B(a_0) \cdot \delta V_C(a_0)$$

Higher-Order IVM using F-IVM: Updates (3/3)

View tree



Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

$$\delta V_D(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V'_C(a_0, c_0) = V_E(a_0, c_0) \cdot \delta V_D(a_0, c_0)$$

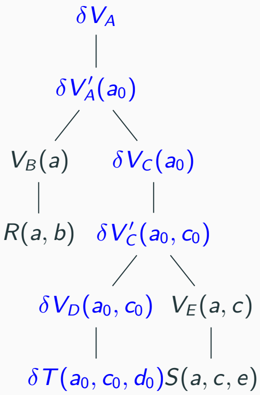
$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

$$\delta V'_A(a_0) = \delta V_B(a_0) \cdot \delta V_C(a_0)$$

$$\delta V_A = \sum_{a_0} \delta V'_A(a_0) = \delta V'_A(a_0)$$

Higher-Order IVM using F-IVM: Updates (3/3)

View tree



δV_A

$\delta V'_A(a_0)$

$V_B(a)$ $\delta V_C(a_0)$

$R(a,b)$ $\delta V'_C(a_0,c_0)$

$\delta V_D(a_0,c_0)$ $V_E(a,c)$

$\delta T(a_0,c_0,d_0)$ $S(a,c,e)$

Single-tuple update $\delta T(a_0, c_0, d_0)$ to T:

$$\delta V_D(a_0, c_0) = \sum_{d_0} \delta T(a_0, c_0, d_0) = \delta T(a_0, c_0, d_0)$$

$$\delta V'_C(a_0, c_0) = V_E(a_0, c_0) \cdot \delta V_D(a_0, c_0)$$

$$\delta V_C(a_0) = \sum_{c_0} \delta V'_C(a_0, c_0) = \delta V'_C(a_0, c_0)$$

$$\delta V'_A(a_0) = \delta V_B(a_0) \cdot \delta V_C(a_0)$$

$$\delta V_A = \sum_{a_0} \delta V'_A(a_0) = \delta V'_A(a_0)$$

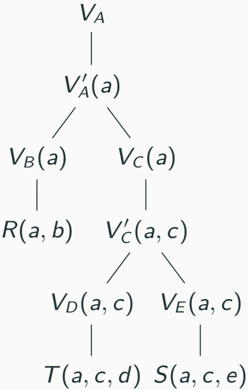
For each updated view/factor A:

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

Each view update takes $\mathcal{O}(1)$ time

Higher-Order IVM using F-IVM: Enumeration

View tree



V_A

$V'_A(a)$

$V_B(a)$ $V_C(a)$

$R(a,b)$ $V'_C(a,c)$

$V_D(a,c)$ $V_E(a,c)$

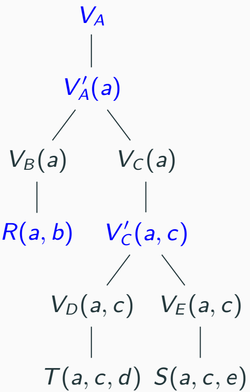
$T(a,c,d)$ $S(a,c,e)$

Enumeration for $Q(a,b,c)$ with constant delay

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

Higher-Order IVM using F-IVM: Enumeration

View tree



Enumeration for $Q(a,b,c)$ with constant delay

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

- Is V_A empty? If yes, stop.
- Iterate over a's in $V'_A(a)$
- For each a, iterate over b's in $R(a,b)$

  Requires index $a \mapsto b$ in R
- For each a, b, iterate over c's in $V'_C(a,c)$

  Requires index $a \mapsto c$ in V'_C
- Output (a,b,c)

Yet DBToaster & F-IVM remain Sub-Optimal

DBToaster & F-IVM are not optimal

for a variety of queries,

including the triangle count

F-IVM for the Triangle Count Query (1/2)

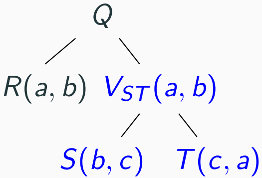
$$Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$$

A single-tuple update δR to R takes time $\mathcal{O}(1)$:

F-IVM creates the materialized view V_{ST}

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

Computation time: $\mathcal{O}(N^2)$



F-IVM for the Triangle Count Query (1/2)

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

A single-tuple update δR to R takes time $\mathcal{O}(1)$:

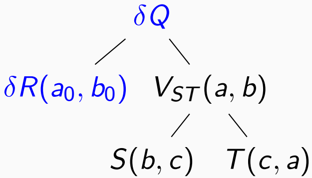
F-IVM creates the materialized view V_{ST}

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

The delta query for update δR using V_{ST} is:

$$\delta Q = \sum_{a_0,b_0} \delta R(a_0,b_0) \cdot V_{ST}(a_0,b_0)$$

$$= \delta R(a_0,b_0) \cdot V_{ST}(a_0,b_0)$$



F-IVM for the Triangle Count Query (2/2)

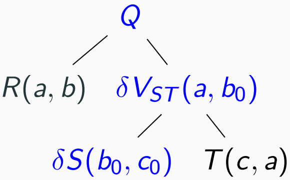
$$Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$$

Q and the view V_{ST} need $\mathcal{O}(N)$ for single-tuple updates:

Delta queries for updates δS and δT:

$$\delta V_{ST}(a, b_0) = \delta S(b_0, c_0) \cdot T(c_0, a)$$

$$\delta Q = \delta V_{ST}(a, b_0) \cdot R(a, b_0)$$



Q

$R(a,b)$ $\delta V_{ST}(a, b_0)$

$\delta S(b_0, c_0)$ $T(c,a)$

F-IVM for the Triangle Count Query (2/2)

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

Q and the view V_{ST} need $\mathcal{O}(N)$ for single-tuple updates:

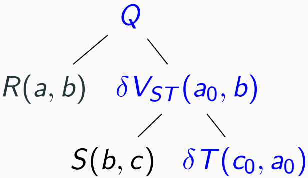
Delta queries for updates δS and δT:

$$\delta V_{ST}(a, b_0) = \delta S(b_0, c_0) \cdot T(c_0, a)$$

$$\delta Q = \delta V_{ST}(a, b_0) \cdot R(a, b_0)$$

$$\delta V_{ST}(a_0, b) = \delta T(c_0, a_0) \cdot S(b, c_0)$$

$$\delta Q = \delta V_{ST}(a_0, b) \cdot R(a_0, b)$$



Q

$R(a,b)$ $\delta V_{ST}(a_0, b)$

$S(b,c)$ $\delta T(c_0, a_0)$

DBToaster for the Triangle Count Query

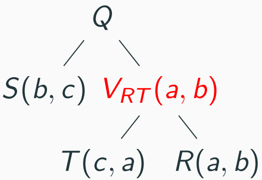
$$Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$$

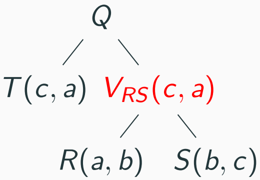
DBToaster also creates the materialized views V_{RT} and V_{RS} to support updates to S and T:

DBToaster for the Triangle Count Query

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

DBToaster also creates the materialized views V_{RT} and V_{RS} to support updates to S and T:



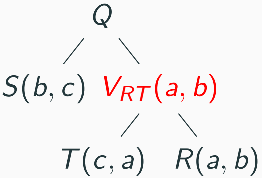


They again require $\mathcal{O}(N)$ update time

DBToaster for the Triangle Count Query

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

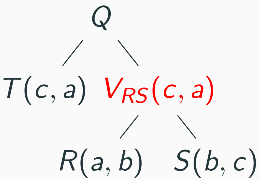
DBToaster also creates the materialized views V_{RT} and V_{RS} to support updates to S and T:



Q

$S(b,c)$ $V_{RT}(a,b)$

$T(c,a)$ $R(a,b)$



Q

$T(c,a)$ $V_{RS}(c,a)$

$R(a,b)$ $S(b,c)$

They again require $\mathcal{O}(N)$ update time

Single-tuple updates to each of the three factors require $\mathcal{O}(N)$ time

IVM for the Triangle Count Query: Summing Up

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

- First-order IVM: $\mathcal{O}(N)$ time for update to any factor

IVM for the Triangle Count Query: Summing Up

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

- First-order IVM: $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using DBToaster
  - creates three view trees
  - $\mathcal{O}(N)$ time for update to any factor

IVM for the Triangle Count Query: Summing Up

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

- First-order IVM: $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using DBToaster
    - creates three view trees
    - $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using F-IVM
    - creates a single view tree
    - $\mathcal{O}(1)$ time for update to one factor
    - $\mathcal{O}(N)$ time for update to the other two factors

IVM for the Triangle Count Query: Summing Up

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

- First-order IVM: $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using DBToaster
  - creates three view trees
  - $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using F-IVM
  - creates a single view tree
  - $\mathcal{O}(1)$ time for update to one factor
  - $\mathcal{O}(N)$ time for update to the other two factors

Can we achieve the optimal update time of $\mathcal{O}(N^{1/2})$?

IVM for the Triangle Count Query: Summing Up

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

- First-order IVM: $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using DBToaster
  - creates three view trees
  - $\mathcal{O}(N)$ time for update to any factor
- Higher-order IVM using F-IVM
  - creates a single view tree
  - $\mathcal{O}(1)$ time for update to one factor
  - $\mathcal{O}(N)$ time for update to the other two factors

Can we achieve the optimal update time of $\mathcal{O}(N^{1/2})$?

Yes! Adaptive IVM

Part 2.

Main IVM techniques:

Adaptive IVM

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
  [SODA 2002, COCOON 2005, PODS 2006, PODS 2016, Theor. Comput. Sci. 2017]

Answering Queries under Updates

- Theoretical developments *[PODS 2017, ICDT 2018]*
- Systems developments *[F. & T. DB 2012, VLDB J. 2014, SIGMOD 2017, 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]*
- Randomized approximation in data streams *[SODA 2002, COCOON 2005, PODS 2006, PODS 2016, Theor. Comput. Sci. 2017]*

Answering Queries under Updates

- Theoretical developments *[PODS 2017, ICDT 2018]*
- Systems developments *[F. & T. DB 2012, VLDB J. 2014, SIGMOD 2017, 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

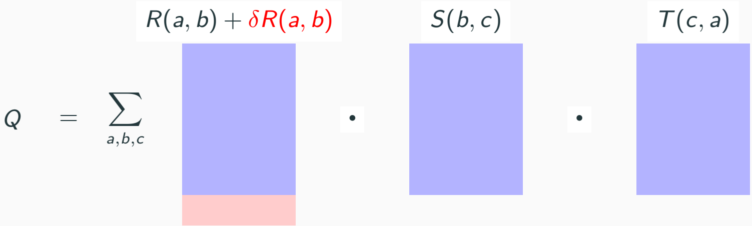
“Recompute from scratch”

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

Naïve Maintenance

"Recompute from scratch"

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

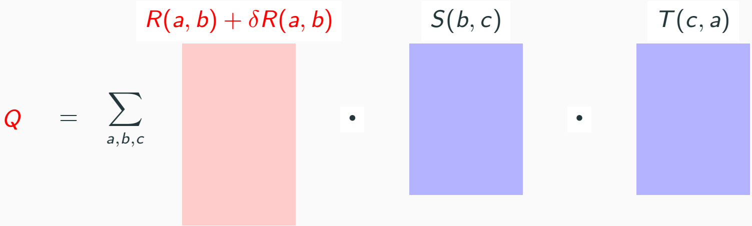


$$Q = \sum_{a,b,c} R(a,b) + \delta R(a,b) \cdot S(b,c) \cdot T(c,a)$$

Naïve Maintenance

"Recompute from scratch"

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



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

Naïve Maintenance

"Recompute from scratch"

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

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

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

First-Order Incremental View Maintenance

"Compute the delta" *[Found. & Trends DB 2018]*

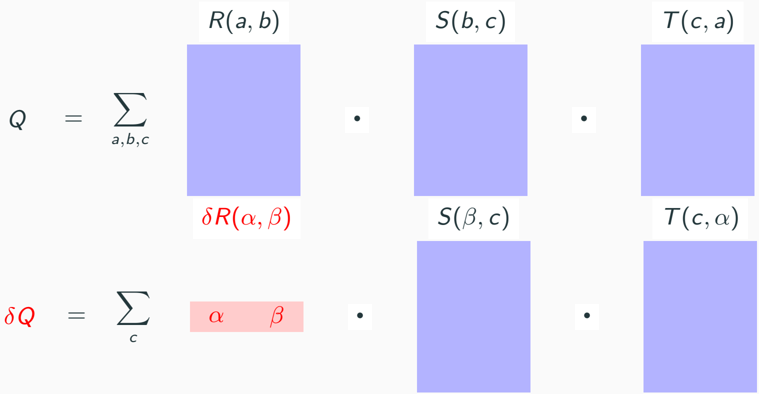
$$Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$$

First-Order Incremental View Maintenance

“Compute the delta”

[Found. & Trends DB 2018]

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



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

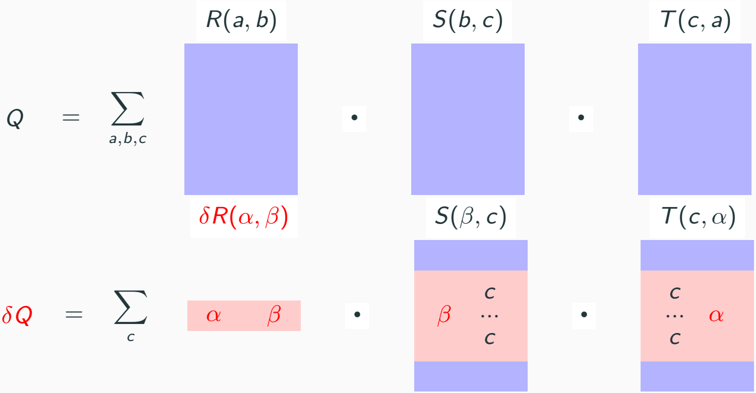
$$\delta Q = \sum_{c} \delta R(\alpha,\beta) \cdot S(\beta,c) \cdot T(c,\alpha)$$

First-Order Incremental View Maintenance

"Compute the delta"

[Found. & Trends DB 2018]

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



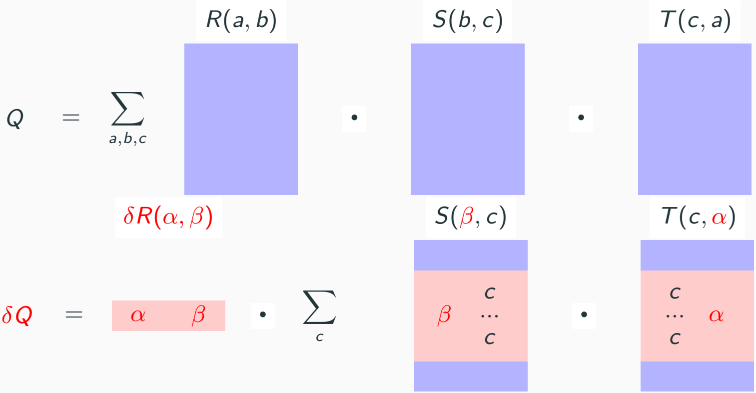
$$Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$$

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

First-Order Incremental View Maintenance

"Compute the delta" *[Found. & Trends DB 2018]*

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



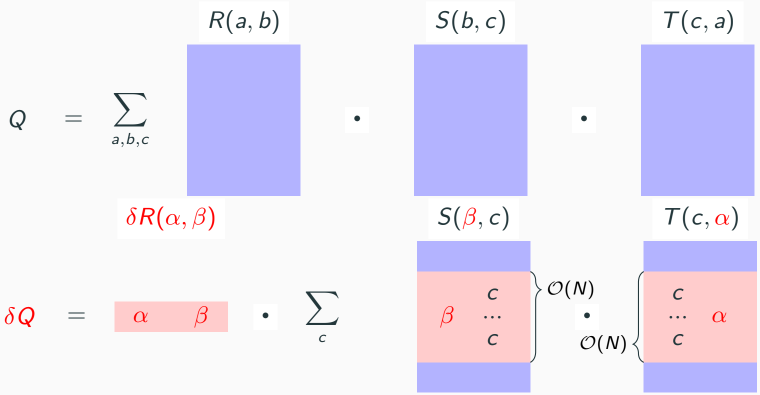
$$Q = \sum_{a,b,c} R(a,b) \cdot S(b,c) \cdot T(c,a)$$

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

First-Order Incremental View Maintenance

"Compute the delta" *[Found. & Trends DB 2018]*

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



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

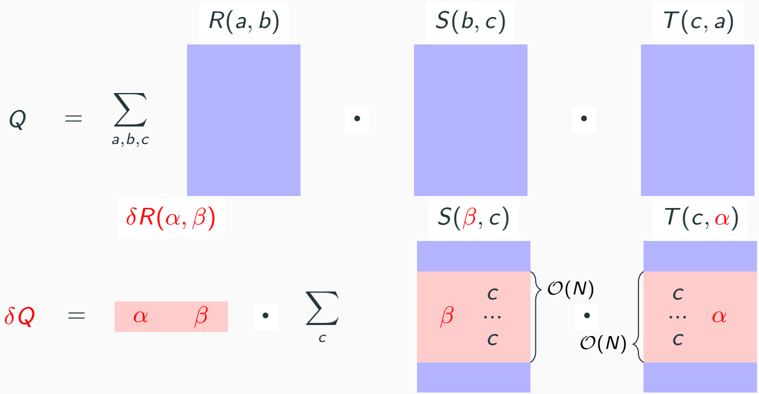
$$\delta Q = \delta R(\alpha,\beta) \cdot \sum_{c} S(\beta,c) \cdot T(c,\alpha)$$

First-Order Incremental View Maintenance

"Compute the delta"

[Found. & Trends DB 2018]

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



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

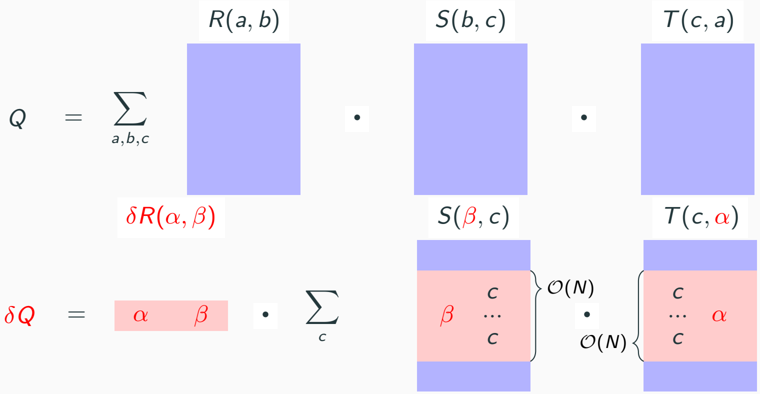
$$\delta Q = \delta R(\alpha,\beta) \cdot \sum_{c} S(\beta,c) \cdot T(c,\alpha)$$

$Q = Q + \delta Q$

First-Order Incremental View Maintenance

"Compute the delta" *[Found. & Trends DB 2018]*

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



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

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

$$Q = Q + \delta Q$$

- Update time: $\mathcal{O}(N)$ to intersect C-values from S and T
- Space: $\mathcal{O}(N)$ to store input factors

Higher-Order Incremental View Maintenance

“Compute the delta using materialized views” *[VLDB J 2014]*

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

Higher-Order Incremental View Maintenance

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

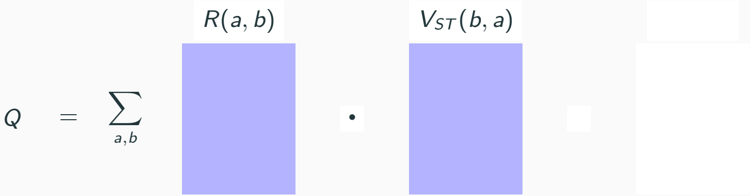
$$\delta R = \{(\alpha, \beta) \mapsto m\}$$

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

Higher-Order Incremental View Maintenance

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

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

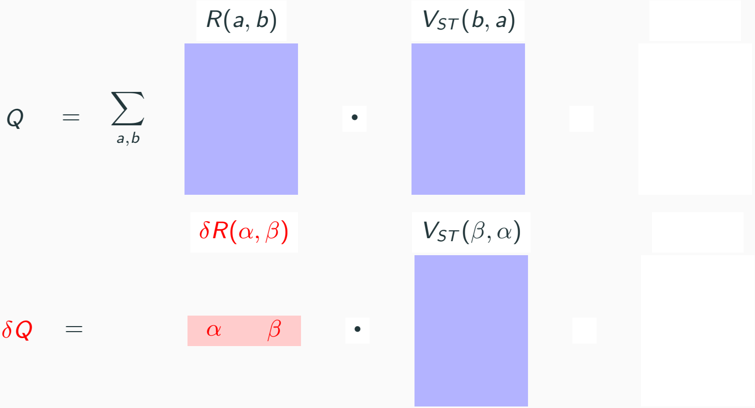


$$Q = \sum_{a,b} R(a,b) \cdot V_{ST}(b,a)$$

Higher-Order Incremental View Maintenance

“Compute the delta using materialized views” *[VLDB J 2014]*

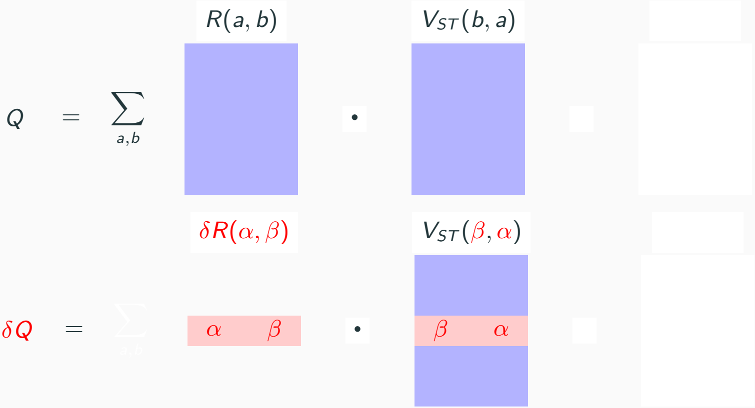
$$\delta R = \{(\alpha, \beta) \mapsto m\}$$



Higher-Order Incremental View Maintenance

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

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



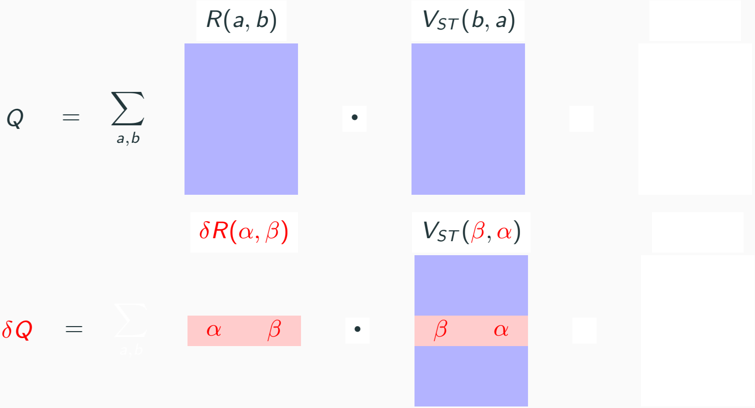
$$Q = \sum_{a,b} R(a, b) \cdot V_{ST}(b, a)$$

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

Higher-Order Incremental View Maintenance

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

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



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

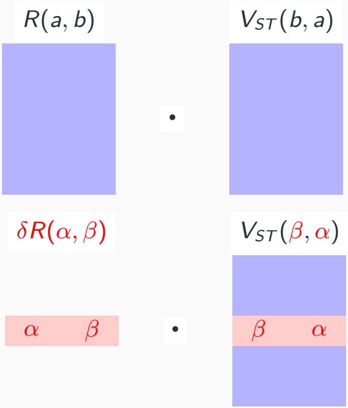
$$\delta Q = \sum_{a,b} \delta R(\alpha, \beta) \cdot V_{ST}(\beta, \alpha)$$

$$Q = Q + \delta Q$$

Higher-Order Incremental View Maintenance

“Compute the delta using materialized views” *[VLDB J 2014]*

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



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

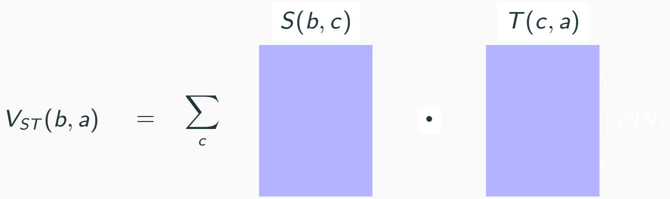
$$\delta Q = \sum_{a,b} \delta R(\alpha, \beta) \cdot V_{ST}(\beta, \alpha)$$

$$Q = Q + \delta Q$$

- Time for updates to R: $\mathcal{O}(1)$ to look up in V_{ST}

Higher-Order Incremental View Maintenance

Maintain V_{ST} under updates

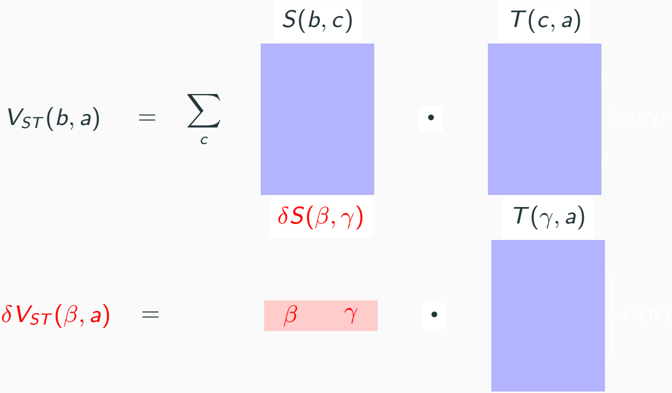


$$V_{ST}(b, a) = \sum_{c} S(b, c) \cdot T(c, a)$$

Higher-Order Incremental View Maintenance

Maintain V_{ST} under updates

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



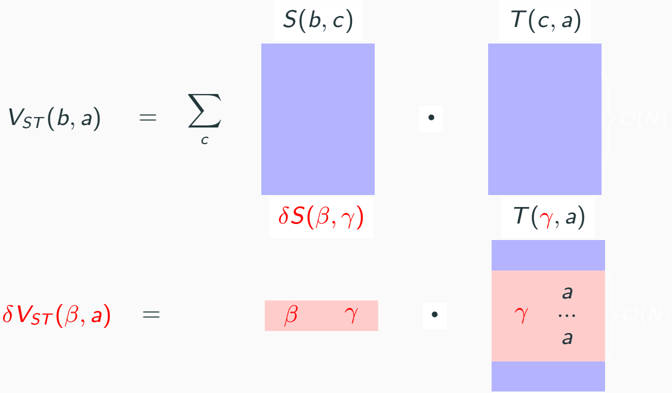
$$V_{ST}(b,a) = \sum_{c} S(b,c) \cdot T(c,a)$$

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

Higher-Order Incremental View Maintenance

Maintain V_{ST} under updates

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



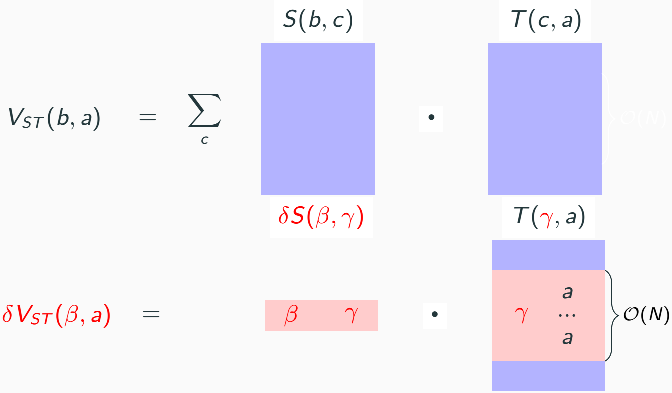
$$V_{ST}(b, a) = \sum_{c} S(b, c) \cdot T(c, a)$$

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

Higher-Order Incremental View Maintenance

Maintain V_{ST} under updates

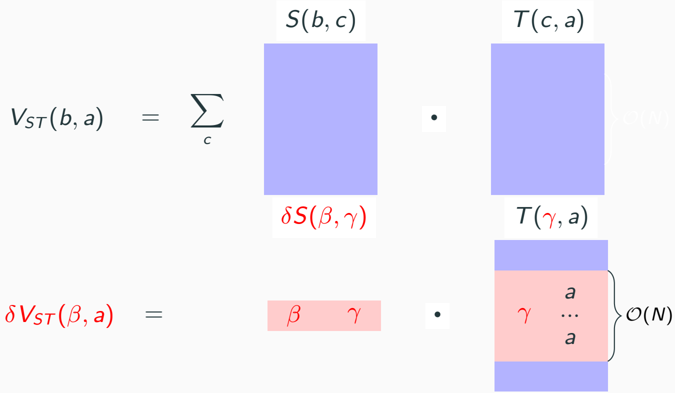
$\delta S = \{(\beta, \gamma) \mapsto m\}$



Higher-Order Incremental View Maintenance

Maintain V_{ST} under updates

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



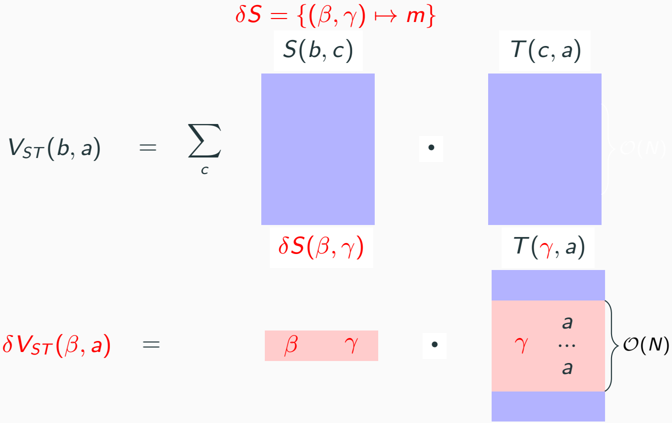
$$V_{ST}(b, a) = \sum_{c} S(b, c) \bullet T(c, a)$$

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

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

Higher-Order Incremental View Maintenance

Maintain V_{ST} under updates



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

- Time for updates to S and T: $\mathcal{O}(N)$ to maintain V_{ST}
- Space: $\mathcal{O}(N^2)$ to store input factors and V_{ST}

Lower Bound for
Maintaining the Triangle Count

The Boolean Triangle Detection Problem

Boolean Triangle Detection Query

$$Q_b = \bigvee_{a,b,c} R(a,b) \wedge S(b,c) \wedge T(c,a)$$

The Boolean Triangle Detection Problem

Boolean Triangle Detection Query

$$Q_b = \bigvee_{a,b,c} R(a,b) \wedge S(b,c) \wedge T(c,a)$$

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

update time	enumeration delay
$\mathcal{O}(N^{\frac{1}{2}-\gamma})$	$\mathcal{O}(N^{1-\gamma})$

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

Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- Input: An $n \times n$ Boolean matrix $\mathbf{M}$ and n pairs $(\mathbf{u}_1, \mathbf{v}_1), \dots, (\mathbf{u}_n, \mathbf{v}_n)$ of Boolean column-vectors of size n arriving one after the other.
- Goal: After seeing each pair $(\mathbf{u}_r, \mathbf{v}_r)$, output $\mathbf{u}_r^\top \mathbf{M} \mathbf{v}_r$

Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- Input: An $n \times n$ Boolean matrix $\mathbf{M}$ and n pairs $(\mathbf{u}_1, \mathbf{v}_1), \ldots, (\mathbf{u}_n, \mathbf{v}_n)$ of Boolean column-vectors of size n arriving one after the other.
- Goal: After seeing each pair $(\mathbf{u}_r, \mathbf{v}_r)$, output $\mathbf{u}_r^\top \mathbf{M} \mathbf{v}_r$

The OuMv Conjecture [*STOC 2015*]

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

Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- Input: An $n \times n$ Boolean matrix $\mathbf{M}$ and n pairs $(\mathbf{u}_1, \mathbf{v}_1), \ldots, (\mathbf{u}_n, \mathbf{v}_n)$ of Boolean column-vectors of size n arriving one after the other.
- Goal: After seeing each pair $(\mathbf{u}_r, \mathbf{v}_r)$, output $\mathbf{u}_r^\top \mathbf{M} \mathbf{v}_r$

The OuMv Conjecture [*STOC 2015*]

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

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

The OMv problem:

- Input: An $n \times n$ Boolean matrix $\mathbf{M}$ and n Boolean column-vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of size n arriving one after the other
- Goal: After seeing each vector $\mathbf{v}_r$, output $\mathbf{M}\mathbf{v}_r$

Online Vector-Matrix-Vector Multiplication

The OuMv problem:

- Input: An $n \times n$ Boolean matrix $\mathbf{M}$ and n pairs $(\mathbf{u}_1, \mathbf{v}_1), \ldots, (\mathbf{u}_n, \mathbf{v}_n)$ of Boolean column-vectors of size n arriving one after the other.
- Goal: After seeing each pair $(\mathbf{u}_r, \mathbf{v}_r)$, output $\mathbf{u}_r^\top \mathbf{M} \mathbf{v}_r$

The OuMv Conjecture [*STOC 2015*]

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

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

The OMv problem:

- Input: An $n \times n$ Boolean matrix $\mathbf{M}$ and n Boolean column-vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of size n arriving one after the other
- Goal: After seeing each vector $\mathbf{v}_r$, output $\mathbf{M}\mathbf{v}_r$

The OMv Conjecture

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

Proof Idea

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

amortized update time	enumeration delay
$\mathcal{O}(N^{\frac{1}{2}-\gamma})$	$\mathcal{O}(N^{1-\gamma})$

  for some $\gamma > 0$.

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

Proof Idea

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

amortized update time	enumeration delay
$\mathcal{O}(N^{\frac{1}{2}-\gamma})$	$\mathcal{O}(N^{1-\gamma})$

  for some $\gamma > 0$.

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

Algorithm $\mathcal{B}$

- Factor S encodes the matrix $\mathbf{M}$: $S(i,j) = \mathbf{M}[i,j]$
- In each round $r \in [n]$:
  - Factor R encodes the vector $\mathbf{u}_r$: $R(\mathrm{a},i) = \mathbf{u}_r[i]$, for constant a
  - Factor T encodes the vector $\mathbf{v}_r$: $T(j,\mathrm{a}) = \mathbf{v}_r[j]$, for constant a
  - Then $\mathbf{u}_r^\top \mathbf{M} \mathbf{v}_r = Q_b$
  - Check whether $Q_b = 1$ using algorithm $\mathcal{A}$.

Example Encoding for u, M, and v

$\mathbf{u}^\top$

0	1	0

$\mathbf{M}$

0	1	0
1	1	0
1	0	1

$\mathbf{v}$

1
0
0

$\mathbf{u}^\top\mathbf{M}\mathbf{v}$

1

R

A	B	val
a	2	1

S

B	C	val
2	1	1
3	1	1
1	2	1
2	2	1
3	3	1

T

C	A	val
1	a	1

Q_b

$\emptyset$	val
( )	1

Proof Sketch: Algorithm $\mathcal{B}$

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

Proof Sketch: Algorithm $\mathcal{B}$

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

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

- Delete all tuples in R and T ($\leq 2n$ deletions)

Proof Sketch: Algorithm $\mathcal{B}$

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

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

- Delete all tuples in R and T ($\leq 2n$ deletions)
- Insert into R and T:
  For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$ ($\leq 2n$ insertions)

Proof Sketch: Algorithm $\mathcal{B}$

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

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

- Delete all tuples in R and T ($\leq 2n$ deletions)
- Insert into R and T:
  For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$ ($\leq 2n$ insertions)
- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

$$\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1 \Leftrightarrow \exists i, j \in [n] : \mathbf{u}_r[i] = 1, \mathbf{M}[i,j] = 1, \mathbf{v}_r[j] = 1$$

Proof Sketch: Algorithm $\mathcal{B}$

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

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

- Delete all tuples in R and T ($\leq 2n$ deletions)
- Insert into R and T:
  For $i, j \in [n]$: $R(\mathrm{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathrm{a}) = \mathbf{v}_r[j]$ ($\leq 2n$ insertions)
- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

$$\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1 \Leftrightarrow \exists i, j \in [n] : \mathbf{u}_r[i] = 1, \mathbf{M}[i, j] = 1, \mathbf{v}_r[j] = 1$$

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

Proof Sketch: Time Analysis

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

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

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

- Delete all tuples in R and T
- Insert into R and T: For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$

- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

Proof Sketch: Time Analysis

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

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

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

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

- Delete all tuples in R and T
- Insert into R and T: For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$

- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

Proof Sketch: Time Analysis

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

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

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

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

- Delete all tuples in R and T
- Insert into R and T: For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$

$$\mathcal{O}(\underbrace{4n}_{\#\text{updates}} \cdot \underbrace{(n^2)^{\frac{1}{2}-\gamma}}_{\text{update time}}) = \mathcal{O}(n^{2-2\gamma})$$

- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

Proof Sketch: Time Analysis

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

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

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

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

- Delete all tuples in R and T
- Insert into R and T: For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$

$$\mathcal{O}(\underbrace{4n}_{\#\text{updates}} \cdot \underbrace{(n^2)^{\frac{1}{2}-\gamma}}_{\text{update time}}) = \mathcal{O}(n^{2-2\gamma})$$

- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

$$\mathcal{O}(\underbrace{(n^2)^{1-\gamma}}_{\text{delay}}) = \mathcal{O}(n^{2-2\gamma})$$

Proof Sketch: Time Analysis

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

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

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

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

- Delete all tuples in R and T
- Insert into R and T: For $i, j \in [n]$: $R(\mathtt{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathtt{a}) = \mathbf{v}_r[j]$

$$\mathcal{O}(\underbrace{4n}_{\#\text{updates}} \cdot \underbrace{(n^2)^{\frac{1}{2}-\gamma}}_{\text{update time}}) = \mathcal{O}(n^{2-2\gamma})$$

- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

$$\mathcal{O}(\underbrace{(n^2)^{1-\gamma}}_{\text{delay}}) = \mathcal{O}(n^{2-2\gamma})$$

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

Proof Sketch: Time Analysis

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

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

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

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

- Delete all tuples in R and T
- Insert into R and T: For $i, j \in [n]$: $R(\mathrm{a}, i) = \mathbf{u}_r[i]$ and $T(j, \mathrm{a}) = \mathbf{v}_r[j]$

$$\mathcal{O}(\underbrace{4n}_{\#\text{updates}} \cdot \underbrace{(n^2)^{\frac{1}{2}-\gamma}}_{\text{update time}}) = \mathcal{O}(n^{2-2\gamma})$$

- Check $Q_b = 1$: This holds if and only if $\mathbf{u}_r^T \mathbf{M} \mathbf{v}_r = 1$

$$\mathcal{O}(\underbrace{(n^2)^{1-\gamma}}_{\text{delay}}) = \mathcal{O}(n^{2-2\gamma})$$

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

Overall time: $\mathcal{O}(n^{3-2\gamma} + n^{3-2\gamma}) = \mathcal{O}(n^{3-2\gamma}) \Rightarrow$ 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: $\mathcal{O}(N)$
Space: $\mathcal{O}(N)$

Known Lower Bound

Update time: not $\mathcal{O}(N^{\frac{1}{2}-\gamma})$ 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: $\mathcal{O}(N)$
Space: $\mathcal{O}(N)$

Can the triangle count be maintained with sublinear update time?

Known Lower Bound

Update time: not $\mathcal{O}(N^{\frac{1}{2}-\gamma})$ 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: $\mathcal{O}(N)$
Space: $\mathcal{O}(N)$

Question	Answer
Can the triangle count be maintained with sublinear update time?	Yes: IVM$^{\varepsilon}$ Amortized update time: $\mathcal{O}(N^{\frac{1}{2}})$ This is worst-case optimal

Known Lower Bound

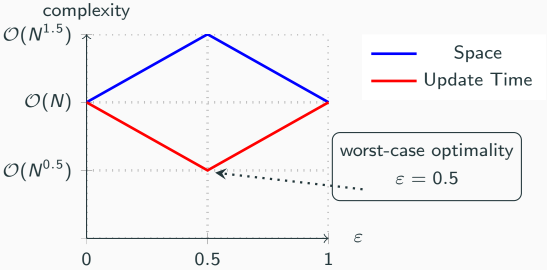
Update time: not $\mathcal{O}(N^{\frac{1}{2}-\gamma})$ for any $\gamma > 0$

under the OuMv Conjecture

IVM$^\varepsilon$ Exhibits a Time-Space Tradeoff

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

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



(Linear space possible with a slightly more involved argument)

Inside IVM$^{\varepsilon}$

Main Techniques used in IVM$^{\varepsilon}$

- Compute the delta like in first-order IVM

- Materialize views like in higher-order IVM

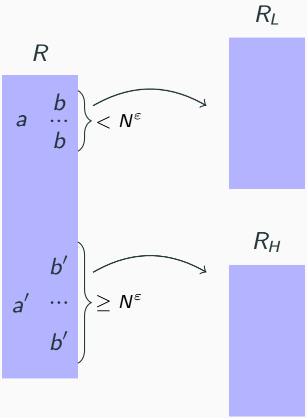
- New ingredient: Use adaptive processing based on data skew

  $\Longrightarrow$ Treat *heavy* values differently from *light* values

Heavy/Light Partitioning of Factors

Partition R based on A into

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



Derived Bounds

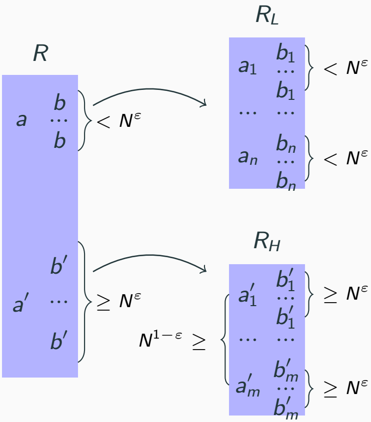
from light part:
for all A-values a, $|\sigma_{A=a} R_L| < N^{\varepsilon}$

from heavy part:
for all A-values a, $|\sigma_{A=a} R_H| \geq N^{\varepsilon}$
and $|\pi_A R_H| \cdot N^{\varepsilon} \leq N$
$\Longrightarrow |\pi_A R_H| \leq N^{1-\varepsilon}$

Heavy/Light Partitioning of Factors

Partition R based on A into

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



Derived Bounds

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

from heavy part:
$|\pi_A R_H| \leq N^{1-\varepsilon}$, since
for all A-values a, $|\sigma_{A=a} R_H| \geq N^\varepsilon$
and $|\pi_A R_H| \cdot N^\varepsilon \leq N$

Heavy/Light Partitioning of Factors

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

$$Q = \sum_{a,b,c} R_U(a,b) \cdot S_V(b,c) \cdot T_W(c,a), \text{ for } U, V, W \in \{L, H\}.$$

Adaptive Maintenance Strategy

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

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

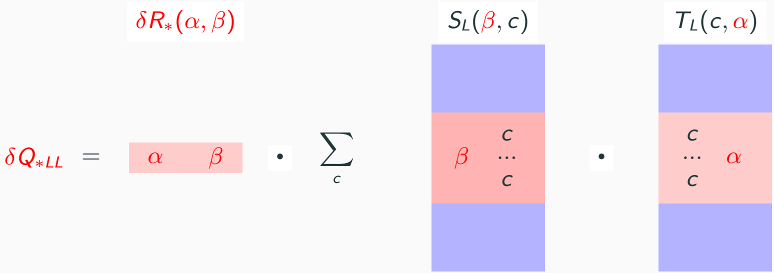
$$\delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_H(c, \alpha)$$

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

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

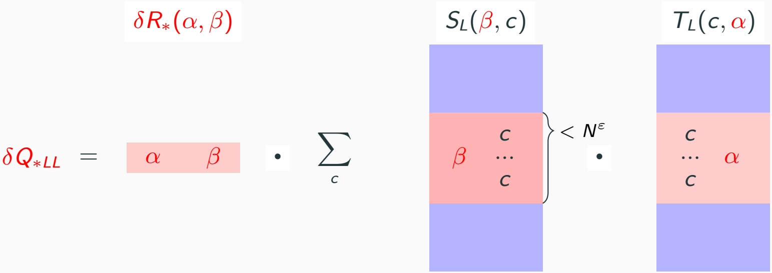
Adaptive Maintenance Strategy

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



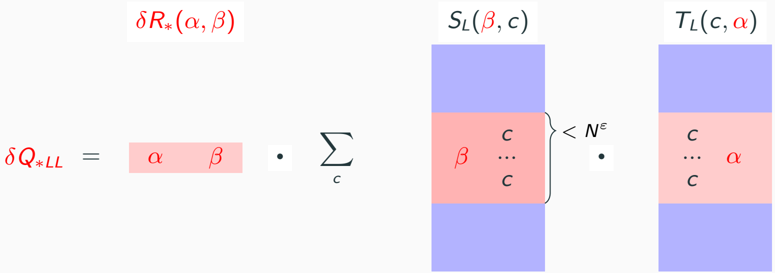
Adaptive Maintenance Strategy

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



Adaptive Maintenance Strategy

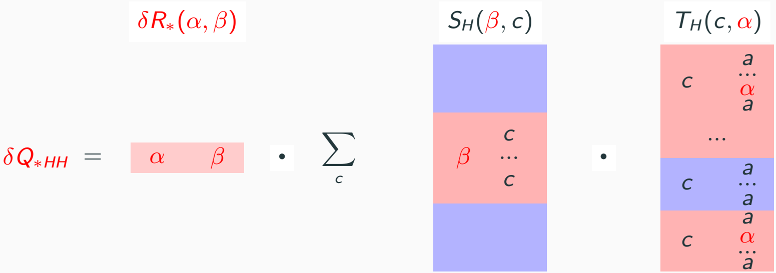
$$\delta Q_{*LL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_L(c, \alpha)$$



Update time: $\mathcal{O}(N^\varepsilon)$ to intersect the lists of C-values from S_L and T_L

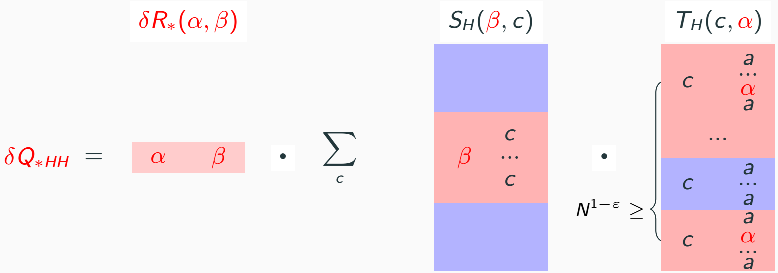
Adaptive Maintenance Strategy

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



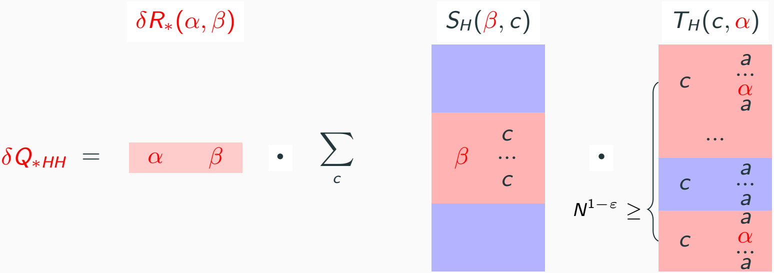
Adaptive Maintenance Strategy

$$\delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_{c} S_H(\beta, c) \cdot T_H(c, \alpha)$$



Adaptive Maintenance Strategy

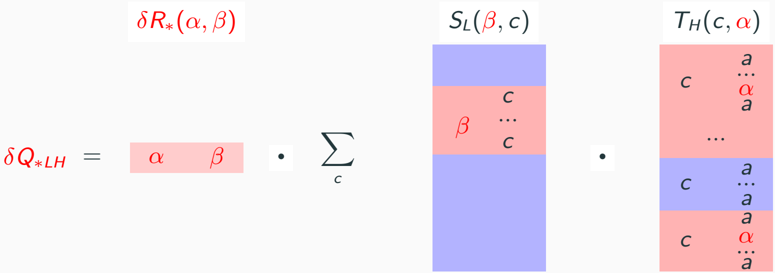
$$\delta Q_{*HH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_H(c, \alpha)$$



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

Adaptive Maintenance Strategy

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



$\delta R_*(\alpha,\beta)$

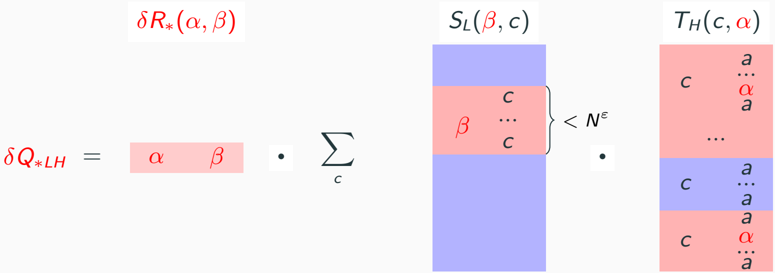
$S_L(\beta,c)$

$T_H(c,\alpha)$

$\delta Q_{*LH} = \begin{array}{cc} \alpha & \beta \end{array} \bullet \sum_c$

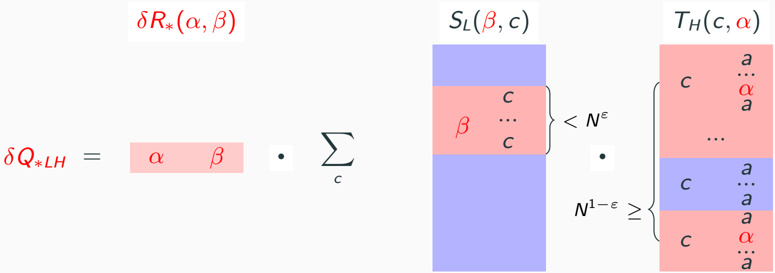
Adaptive Maintenance Strategy

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



Adaptive Maintenance Strategy

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



$\delta R_*(\alpha, \beta)$

$S_L(\beta, c)$

$T_H(c, \alpha)$

$\delta Q_{*LH} =$ α β $\bullet$ $\sum_c$

β c $\cdots$ c

$< N^{\varepsilon}$

$\bullet$

c a $\cdots$ α a

$\cdots$

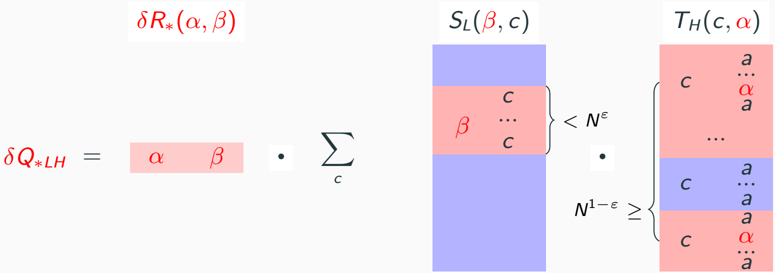
c a $\cdots$ a

c a α $\cdots$ a

$N^{1-\varepsilon} \geq$

Adaptive Maintenance Strategy

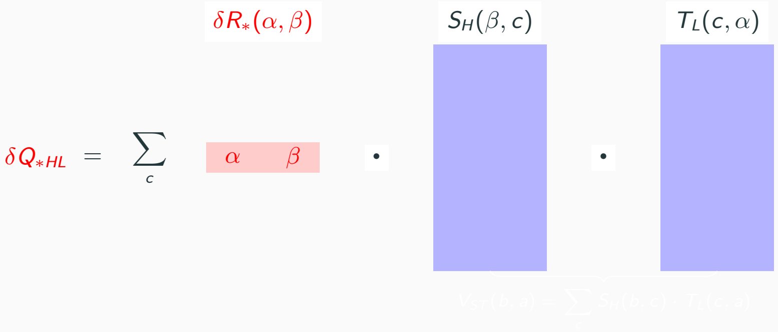
$$\delta Q_{*LH} = \delta R_*(\alpha, \beta) \cdot \sum_c S_L(\beta, c) \cdot T_H(c, \alpha)$$



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

Adaptive Maintenance Strategy

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

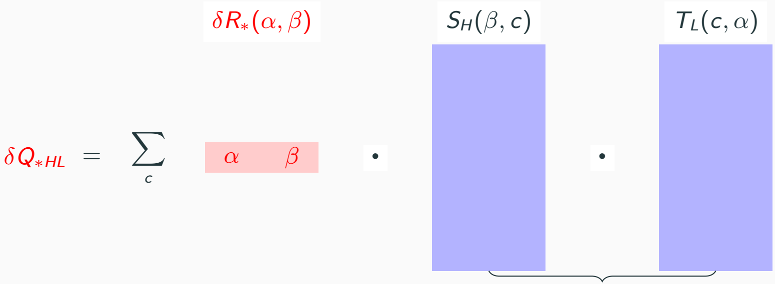


$\delta R_*(\alpha, \beta)$ $S_H(\beta, c)$ $T_L(c, \alpha)$

$$\delta Q_{*HL} = \sum_c \boxed{\alpha \quad \beta} \cdot \underbrace{\boxed{} \cdot \boxed{}}_{V_{ST}(b,a) = \sum_c S_H(b,c) \cdot T_L(c,a)}$$

Adaptive Maintenance Strategy

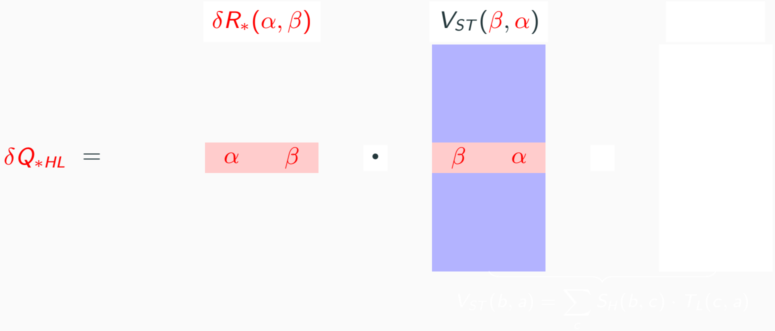
$$\delta Q_{*HL} = \delta R_*(\alpha, \beta) \cdot \sum_c S_H(\beta, c) \cdot T_L(c, \alpha)$$



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

Adaptive Maintenance Strategy

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



$\delta R_*(\alpha, \beta)$

$V_{ST}(\beta, \alpha)$

$\delta Q_{*HL} =$

α β

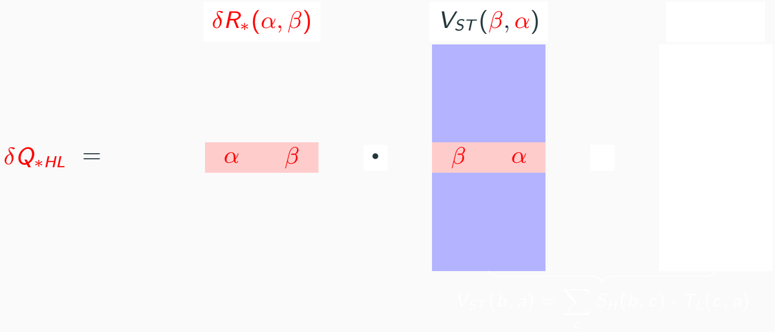
•

β α

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

Adaptive Maintenance Strategy

$$\delta Q_{*HL} = \delta R_{*}(\alpha, \beta) \cdot \sum_{c} S_{H}(\beta, c) \cdot T_{L}(c, \alpha)$$



$$\underbrace{\qquad\qquad}_{V_{ST}(b, a) = \sum_{c} S_{H}(b, c) \cdot T_{L}(c, a)}$$

Update time: $\mathcal{O}(1)$ to look up in V_{ST}, assuming V_{ST} is already materialized

Summary of Adaptive Maintenance Strategies

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

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

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

Auxiliary Materialized Views

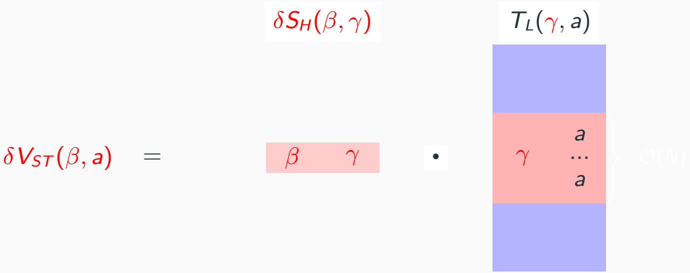
$$V_{RS}(a, c) = \sum_{b} R_H(a, b) \cdot S_L(b, c)$$

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

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

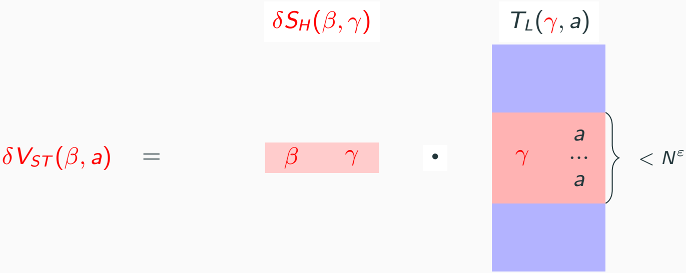
Maintenance of Auxiliary Views

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



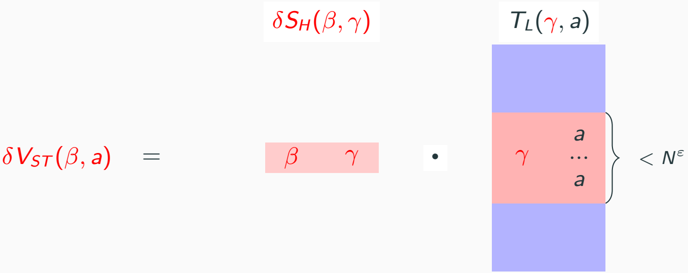
Maintenance of Auxiliary Views

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



Maintenance of Auxiliary Views

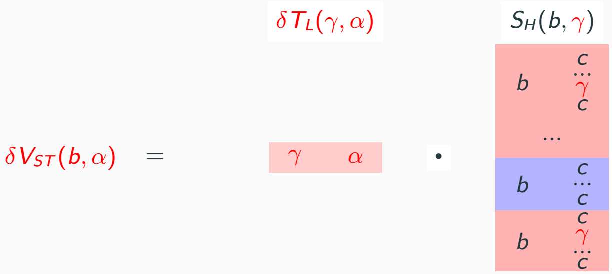
Maintain $V_{ST}(b,a) = \sum_{c} S_H(b,c) \cdot T_L(c,a)$ under update $\delta S_H = \{(\beta,\gamma) \mapsto m\}$



Update time: $\mathcal{O}(N^{\varepsilon})$ to iterate over a-values paired with γ from T_L

Maintenance of Auxiliary Views

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



$\delta V_{ST}(b,\alpha) \quad =$

$\delta T_L(\gamma,\alpha)$

γ	α

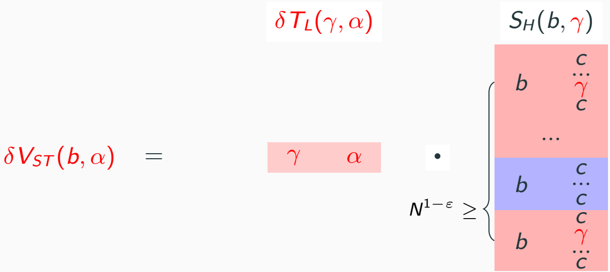
$\bullet$

$S_H(b,\gamma)$

b	c $\dots$ γ c
$\dots$	
b	c $\dots$ c
b	c γ $\dots$ c

Maintenance of Auxiliary Views

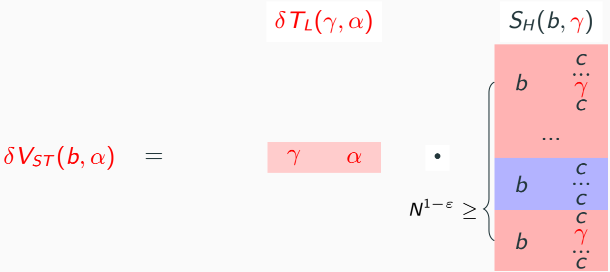
Maintain $V_{ST}(b, a) = \sum_{c} S_H(b, c) \cdot T_L(c, a)$ under update $\delta T_L = \{(\gamma, \alpha) \mapsto m\}$



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

Maintenance of Auxiliary Views

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



Update time: $\mathcal{O}(N^{1-\varepsilon})$ to iterate over b-values paired with γ from S_H

Maintenance of Auxiliary Views: Summary

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

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

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

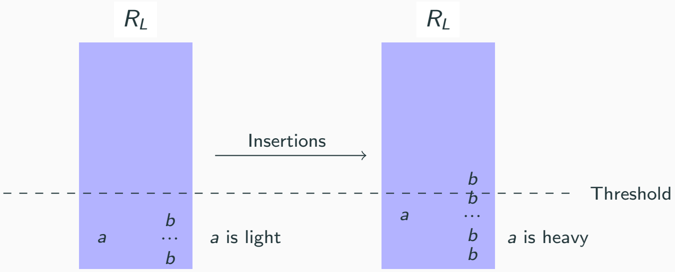
Maintenance Complexity

- 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 factor parts

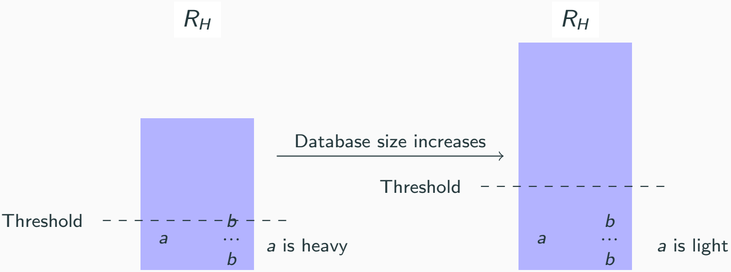


Minor Rebalancing

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

Rebalancing Partitions

Updates can change the heavy-light threshold!



Major Rebalancing

- Recompute partitions and views from scratch
- Time complexity: $\mathcal{O}(N^{1+\max\{\varepsilon,1-\varepsilon\}})$

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: $\mathcal{O}(N^{\max\{\varepsilon,1-\varepsilon\}})$

$$\ldots\ \textit{update}\quad \overset{\mathcal{O}(N^{\varepsilon+\max\{\varepsilon,1-\varepsilon\}})}{\textit{minor}}\quad \underbrace{\textit{update}\ \ldots\ \textit{update}}_{\Omega(N^{\varepsilon})}\quad \overset{\mathcal{O}(N^{\varepsilon+\max\{\varepsilon,1-\varepsilon\}})}{\textit{minor}}\quad \textit{update}\ \ldots$$

$$\ldots\ \textit{update}\quad \overset{\mathcal{O}(N^{1+\max\{\varepsilon,1-\varepsilon\}})}{\textit{major}}\quad \underbrace{\textit{update}\ \ldots\ \textit{update}}_{\Omega(N)}\quad \overset{\mathcal{O}(N^{1+\max\{\varepsilon,1-\varepsilon\}})}{\textit{major}}\quad \textit{update}\ \ldots$$

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: $\mathcal{O}(N^{\max\{\varepsilon,1-\varepsilon\}})$
- Overall amortized rebalancing time: $\mathcal{O}(N^{\max\{\varepsilon,1-\varepsilon\}})$

$$\ldots \quad \mathit{update} \quad \overset{\mathcal{O}(N^{\varepsilon+\max\{\varepsilon,1-\varepsilon\}})}{\mathit{minor}} \quad \underbrace{\mathit{update} \;\ldots\; \mathit{update}}_{\Omega(N^{\varepsilon})} \quad \overset{\mathcal{O}(N^{\varepsilon+\max\{\varepsilon,1-\varepsilon\}})}{\mathit{minor}} \quad \mathit{update} \quad \ldots$$

$$\ldots \quad \mathit{update} \quad \overset{\mathcal{O}(N^{1+\max\{\varepsilon,1-\varepsilon\}})}{\mathit{major}} \quad \underbrace{\mathit{update} \;\ldots\; \mathit{update}}_{\Omega(N)} \quad \overset{\mathcal{O}(N^{1+\max\{\varepsilon,1-\varepsilon\}})}{\mathit{major}} \quad \mathit{update} \quad \ldots$$

References i

[Algorithmica 1997] Noga Alon, Raphael Yuster, and Uri Zwick. *Finding and counting given length cycles.*

[SODA 2002] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. *Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs.*

[COCOON 2005] Hossein Jowhari and Mohammad Ghodsi. *New Streaming Algorithms for Counting Triangles in Graphs.*

[PODS 2006] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. *Counting Triangles in Data Streams.*

[Found. & Trends DB 2012] Rada Chirkova and Jun Yang. *Materialized Views.*

[SIGMOD R. 2013] Hung Q Ngo, Christopher Ré, and Atri Rudra. *Skew strikes back: new developments in the theory of join algorithms.*

References ii

[ICDT 2014] Todd L. Veldhuizen. *Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm.*

[VLDB J. 2014] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. *DBToaster: Higher-Order Delta Processing for Dynamic, Frequently Fresh Views.*

[FOCS 2015] Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. *Approximately Counting Triangles in Sublinear Time.*

[STOC 2015] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. *Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture.*

[PODS 2016] Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. *Better Algorithms for Counting Triangles in Data Streams.*

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

References iii

[SIGMOD 2017] Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. *The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates.*

[Theor. Comput. Sci. 2017] Graham Cormode and Hossein Jowhari. *A Second Look at Counting Triangles in Graph Streams (Corrected).*

[Found. & Trends DB 2018] Paraschos Koutris, Semih Salihoglu, and Dan Suciu. *Algorithmic Aspects of Parallel Data Processing.*

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

[ICM 2018] Virginia Vassilevska Williams. *On Some Fine-Grained Questions in Algorithms and Complexity.*

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

References iv

[ICDT 2019] Ahmet Kara, Milos Nikolic, Hung Q. Ngo, Dan Olteanu, and Haozhe Zhang. *Counting Triangles under Updates in Worst-Case Optimal Time.*

[APOCS 2021] Laxman Dhulipala, Quanquan C. Liu, Julian Shun, Shangdi Yu. *Parallel Batch-Dynamic k-Clique Counting.*

[ICDT 2021] Shangqi Lu, Yufei Tao. *Towards Optimal Dynamic Indexes for Approximate (and Exact) Triangle Counting.*

Part 3.

IVM Optimality

IVM Optimality

Two query classes with known worst-case optimal IVM algorithms

- Q-hierarchical queries
  - The only queries that admit constant update time and delay
- δ_1-hierarchical queries
  - They admit $\mathcal{O}(\sqrt{N})$ update time and delay

Query examples shown in Part 2

Part 3.

IVM Optimality:

Constant Update Time & Delay

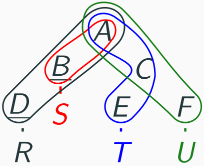
Hierarchical Queries

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

[VLDB 2004]

hierarchical

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



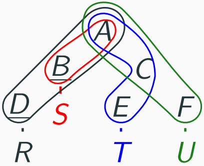
Hierarchical Queries

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

[VLDB 2004]

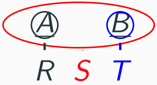
hierarchical

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



not hierarchical

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



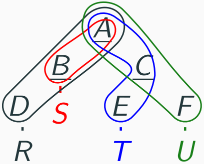
Q-Hierarchical Queries

A query is *q-hierarchical* if it is hierarchical and the free variables dominate the bound variables

[PODS 2017]

q-hierarchical

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



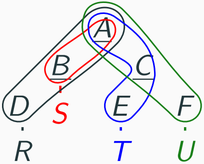
Q-Hierarchical Queries

A query is *q*-hierarchical if it is hierarchical and
the free variables dominate the bound variables

[PODS 2017]

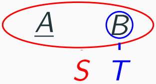
q-hierarchical

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



hierarchical but not *q*-hierarchical

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



Dichotomy for Q-Hierarchical Queries

Let Q be any conjunctive query without self-joins and D a database.

- If Q is **q-hierarchical**, then the query answer admits **O(1)** single-tuple updates and enumeration delay.
- If Q is **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]

Queries under Functional Dependencies

Rewriting queries under functional dependencies [ICDE 2009]

- Given: Query Q and set Σ of functional dependencies
- Replace the set of variables of each atom in Q by its closure under Σ called Σ-reduct

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

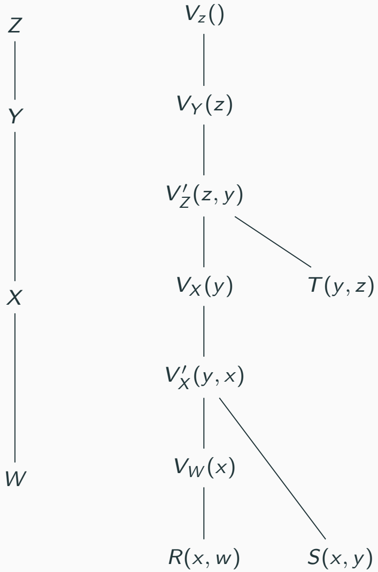
$$Q'(w,x,y,z) = R'(w,x,y,z) \cdot S'(x,y,z) \cdot T'(y,z)$$

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:

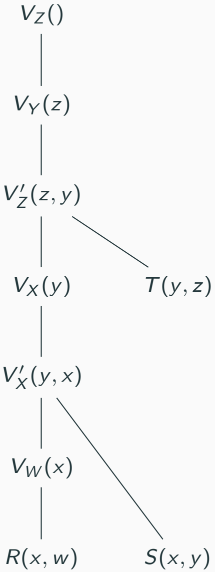
- Place factors at leaves
- Create parent view to join children

$$V'_Z(z,y) = T(y,z) \cdot V_X(y)$$
$$V'_X(y,x) = S(x,y) \cdot V_W(x)$$

- Aggregate away variables not needed for further joins

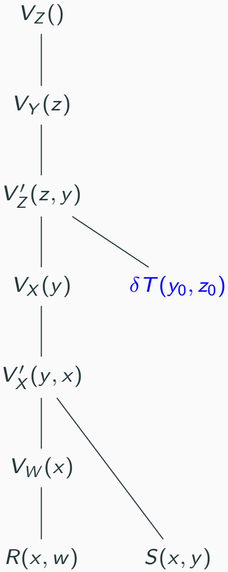
$$V_Z() = \sum_z V_Y(z)$$
$$V_Y(z) = \sum_y V'_Z(y,z)$$
$$V_X(y) = \sum_x V'_X(x,y)$$
$$V_W(x) = \sum_w R'(x,w)$$

Example: Single-Tuple Update to T



Single-tuple update to T

Example: Single-Tuple Update to T



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V_X(y)$

$\delta T(y_0, z_0)$

$V'_X(y, x)$

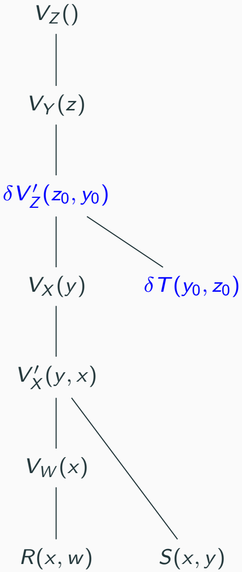
$V_W(x)$

$R(x, w)$

$S(x, y)$

Single-tuple update to T

Example: Single-Tuple Update to T



$V_Z()$

$V_Y(z)$

$\delta V'_Z(z_0, y_0)$

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

$V'_X(y, x)$

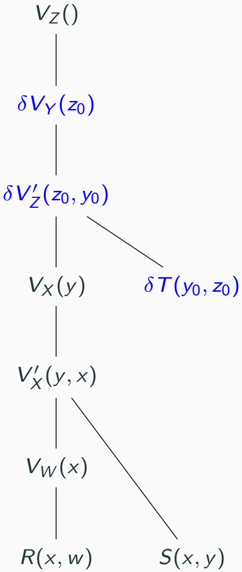
$V_W(x)$

$R(x, w)$ $S(x, y)$

Single-tuple update to T

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

Example: Single-Tuple Update to T



$V_Z()$

$\delta V_Y(z_0)$

$\delta V'_Z(z_0, y_0)$

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

$V'_X(y, x)$

$V_W(x)$

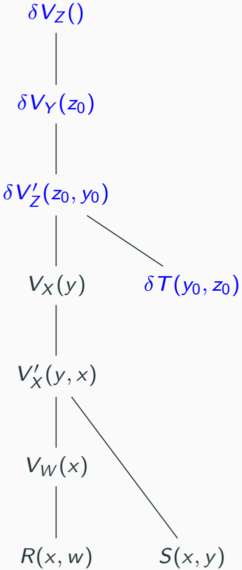
$R(x, w)$ $S(x, y)$

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



Single-tuple update to T

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

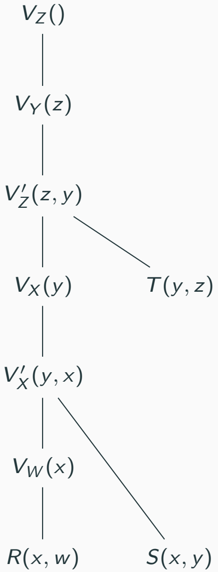
$$\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)$$

$$\delta V_Z() = \sum_{z_0} \delta V_Y(z_0) = \delta V_Y(z_0)$$

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

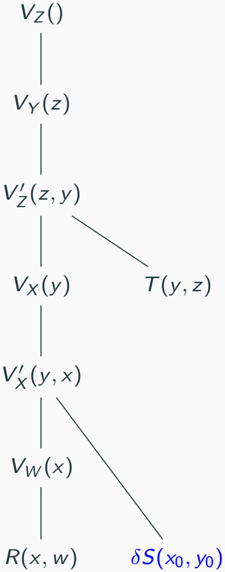
Each view update takes $O(1)$ time

Example: Single-Tuple Update to S



Single-tuple update to S

Example: Single-Tuple Update to S



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V_X(y)$

$T(y, z)$

$V'_X(y, x)$

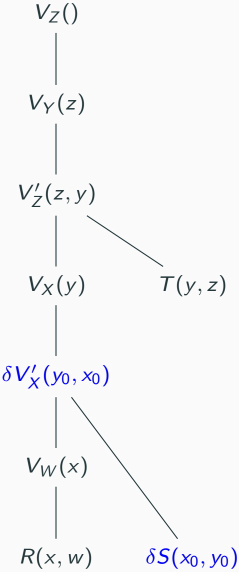
$V_W(x)$

$R(x, w)$

$\delta S(x_0, y_0)$

Single-tuple update to S

Example: Single-Tuple Update to S



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V_X(y)$ $T(y, z)$

$\delta V'_X(y_0, x_0)$

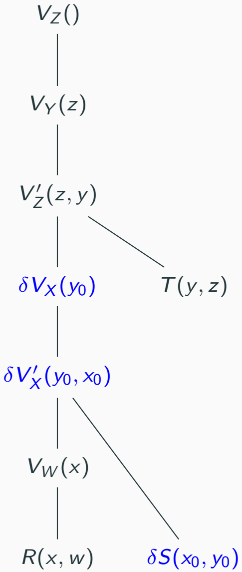
$V_W(x)$

$R(x, w)$ $\delta S(x_0, y_0)$

Single-tuple update to S

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

Example: Single-Tuple Update to S



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$\delta V_X(y_0)$

$T(y, z)$

$\delta V'_X(y_0, x_0)$

$V_W(x)$

$R(x, w)$

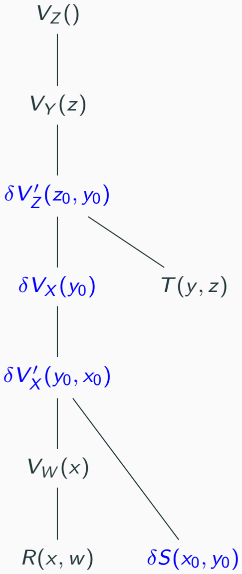
$\delta S(x_0, y_0)$

Single-tuple update to S

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

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

Example: Single-Tuple Update to S



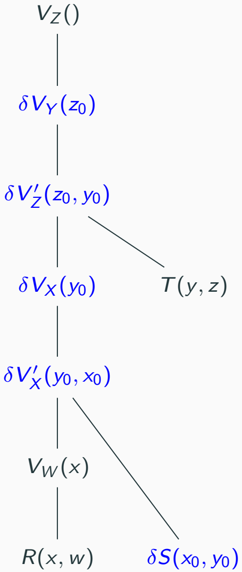
Single-tuple update to S

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

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

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

Example: Single-Tuple Update to S



$V_Z()$

$\delta V_Y(z_0)$

$\delta V'_Z(z_0, y_0)$

$\delta V_X(y_0)$ $T(y, z)$

$\delta V'_X(y_0, x_0)$

$V_W(x)$

$R(x, w)$ $\delta S(x_0, y_0)$

Single-tuple update to S

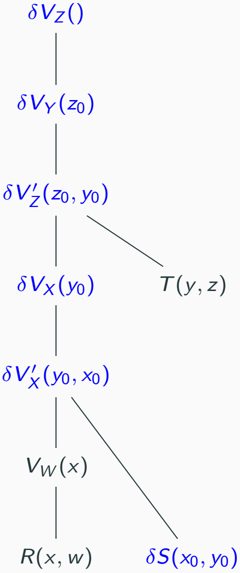
$$\delta V'_X(y_0, x_0) = \delta S(x_0, y_0) \cdot V_W(x_0)$$

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

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

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

Example: Single-Tuple Update to S



Single-tuple update to S

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

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

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

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

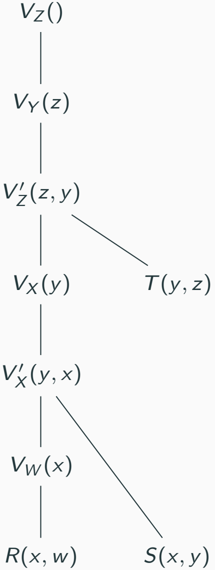
$$\delta V_Z() = \sum_{z_0} \delta V_Y(z_0) = \delta V_Y(z_0)$$

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

Each view update takes $O(1)$ time

Example: Single-Tuple Update to R

Single-tuple update to R



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V_X(y)$

$T(y, z)$

$V'_X(y, x)$

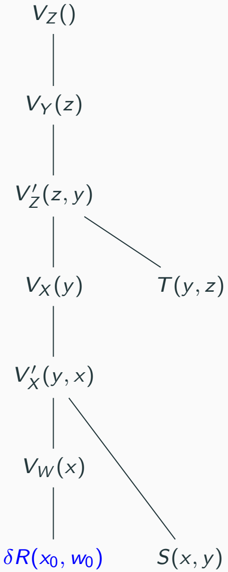
$V_W(x)$

$R(x, w)$

$S(x, y)$

Example: Single-Tuple Update to R

Single-tuple update to R



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V_X(y)$

$T(y, z)$

$V'_X(y, x)$

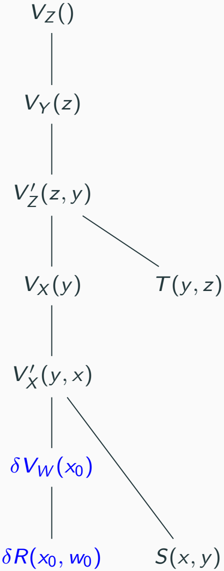
$V_W(x)$

$\delta R(x_0, w_0)$

$S(x, y)$

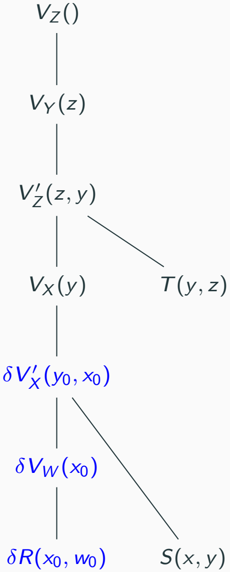
Example: Single-Tuple Update to R

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



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$V_X(y)$

$T(y, z)$

$\delta V'_X(y_0, x_0)$

$\delta V_W(x_0)$

$\delta R(x_0, w_0)$

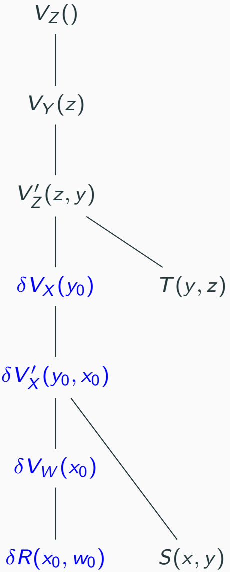
$S(x, y)$

Single-tuple update to R

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

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

Example: Single-Tuple Update to R



$V_Z()$

$V_Y(z)$

$V'_Z(z, y)$

$\delta V_X(y_0)$ $T(y, z)$

$\delta V'_X(y_0, x_0)$

$\delta V_W(x_0)$

$\delta R(x_0, w_0)$ $S(x, y)$

Single-tuple update to R

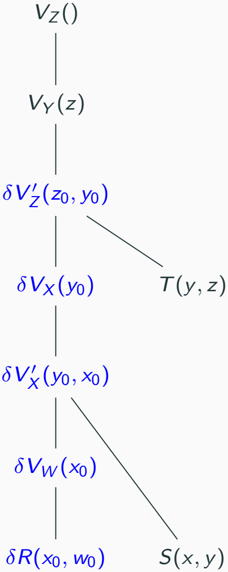
$$\delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)$$

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

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

Example: Single-Tuple Update to R

Single-tuple update to R



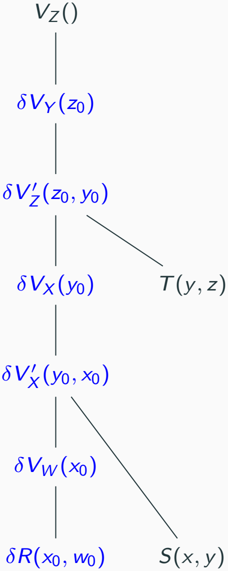
$$\delta V_W(x_0) = \sum_{w_0} \delta R(x_0, w_0) = \delta R(x_0, w_0)$$

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

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

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

Example: Single-Tuple Update to R



$V_Z()$

$\delta V_Y(z_0)$

$\delta V'_Z(z_0, y_0)$

$\delta V_X(y_0)$

$T(y, z)$

$\delta V'_X(y_0, x_0)$

$\delta V_W(x_0)$

$\delta R(x_0, w_0)$

$S(x, y)$

Single-tuple update to R

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

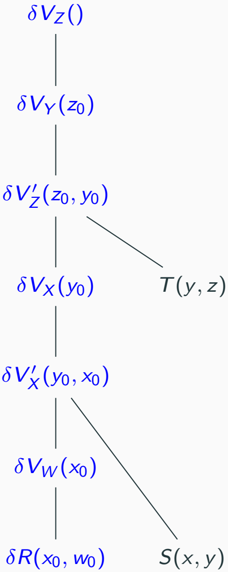
$$\delta V'_X(y_0, x_0) : \delta V_W(x_0) \cdot S(x_0, y) \overset{x \to y}{=} \delta V_W(x_0) \cdot S(x_0, y_0)$$

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

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

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

Example: Single-Tuple Update to R



Single-tuple update to R

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

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

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

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

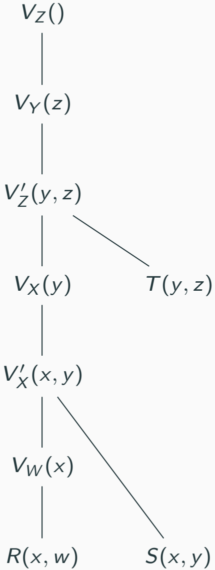
$$\delta V_Y(z_0) = \sum_{y_0} \delta V'_Z(z_0, y_0) = \delta V'_Z(z_0, y_0)$$

$$\delta V_Z() = \sum_{z_0} \delta V_Y(z_0) = \delta V_Y(z_0)$$

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

Each view update takes $O(1)$ time

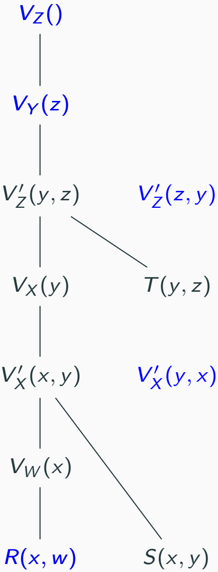
Example: Enumeration of Query Answers



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

Example: Enumeration of Query Answers



$V_Z()$

$V_Y(z)$

$V'_Z(y,z)$ $V'_Z(z,y)$

$V_X(y)$ $T(y,z)$

$V'_X(x,y)$ $V'_X(y,x)$

$V_W(x)$

$R(x,w)$ $S(x,y)$

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

The enumeration procedure:

- 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 $T(z,y)$, $S(x,y)$, $R(x,w)$

Part 3.

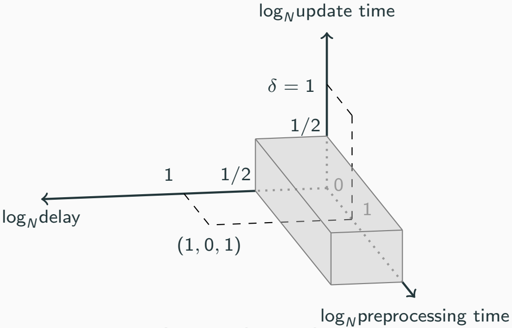
IVM Optimality: $\mathcal{O}(\sqrt{N})$ Update Time & Delay

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

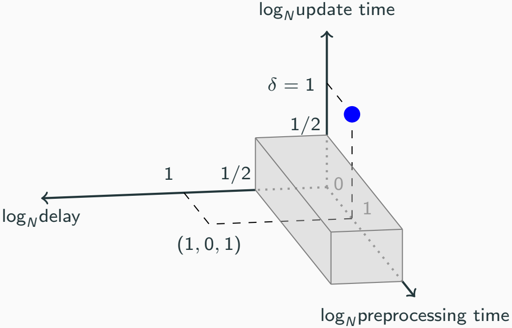
For this query, there is no algorithm that admits

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

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

Simplest Hierarchical Query without “Q” Property

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

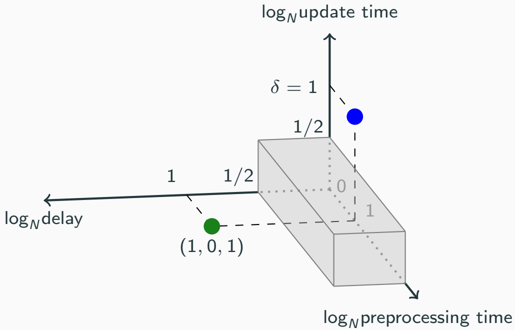


Known approach: Eager update, quick enumeration

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

Simplest Hierarchical Query without “Q” Property

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

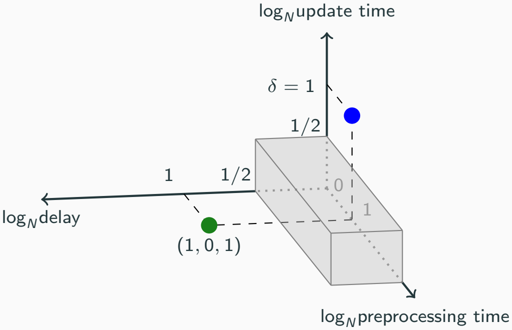


Known approach: Lazy update, heavy enumeration

- Preprocessing: Eliminate dangling tuples
- Upon update: Update only input factors
- Enumeration: Eliminate dangling tuples and enumerate from R

Simplest Hierarchical Query without “Q” Property

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

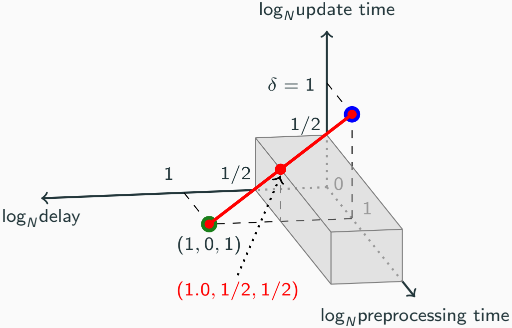


Yet, there is an algorithm that admits

sub-linear update time and sub-linear enumeration delay

Simplest Hierarchical Query without “Q” Property

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



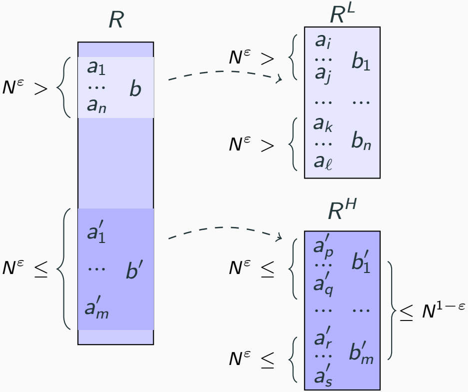
Weak Pareto optimality

Factor 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 $R^H = R - R^L$

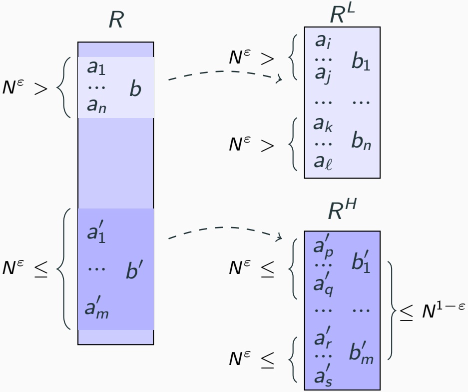


Factor Partitioning

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

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

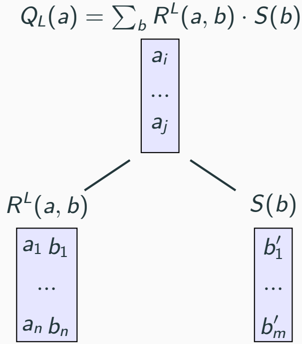
$$Q_L(a) = \sum_b R^L(a, b) \cdot S(b)$$

Materialize the result

Light Case

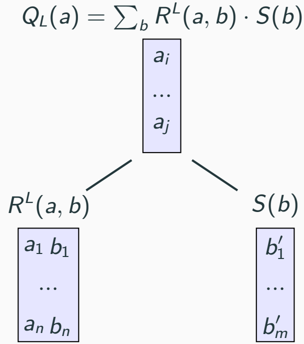
$$Q_L(a) = \sum_b R^L(a,b) \cdot S(b)$$

Materialize the result



Preprocessing in the Light Case

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



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

a_i
...
a_j

$R^L(a,b)$

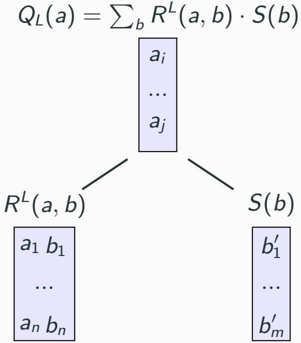
a_1 b_1
...
a_n b_n

$S(b)$

b'_1
...
b'_m

- Q_L can be computed in time $\mathcal{O}(N)$

Enumeration in the Light Case



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

a_i
$\ldots$
a_j

$R^L(a,b)$

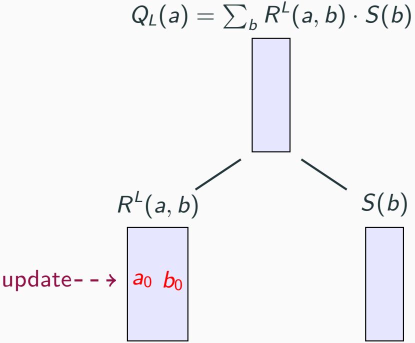
$a_1\ b_1$
$\ldots$
$a_n\ b_n$

$S(b)$

b'_1
$\ldots$
b'_m

- Q_L allows constant-time lookups and constant-delay enumeration

Updates in the Light Case

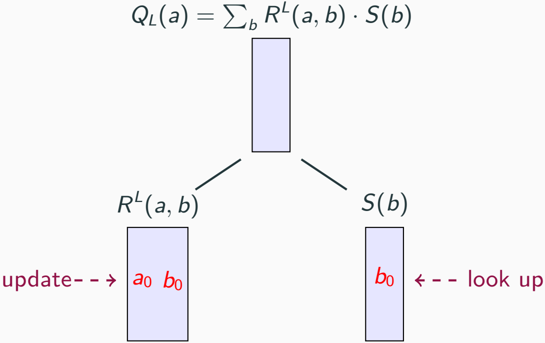


$Q_L(a) = \sum_b R^L(a, b) \cdot S(b)$

$R^L(a, b)$

$S(b)$

Updates in the Light Case



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

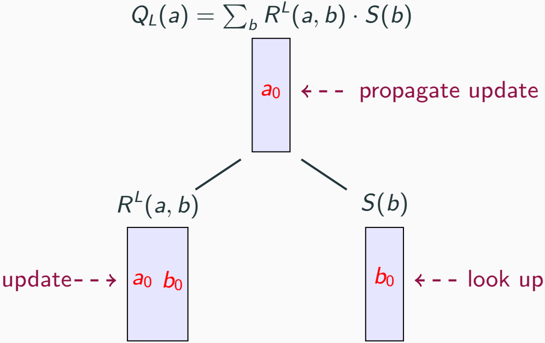
$R^L(a, b)$

$S(b)$

update- - → a_0 b_0

b_0 ← - - look up

Updates in the Light Case



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

a_0

propagate update

$R^L(a,b)$

$S(b)$

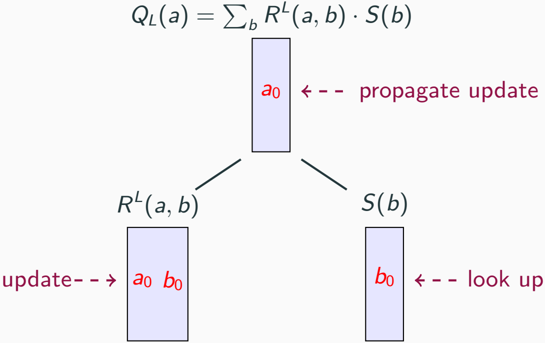
update

a_0 b_0

b_0

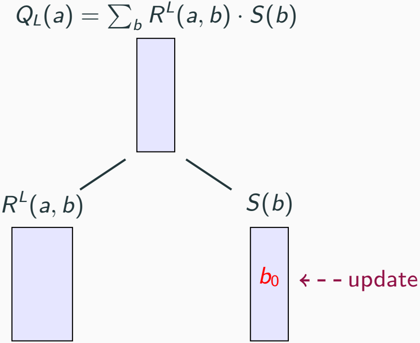
look up

Updates in the Light Case



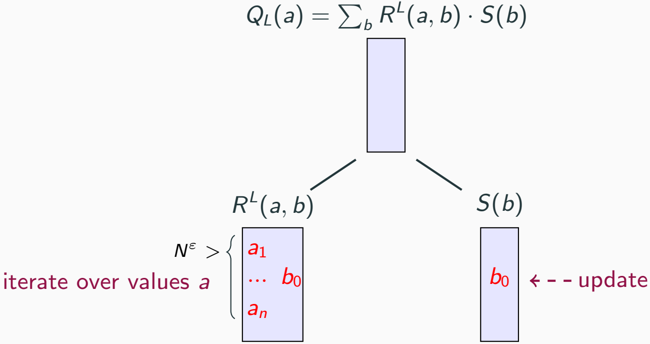
- Updates to R^L: $\mathcal{O}(1)$

Updates in the Light Case



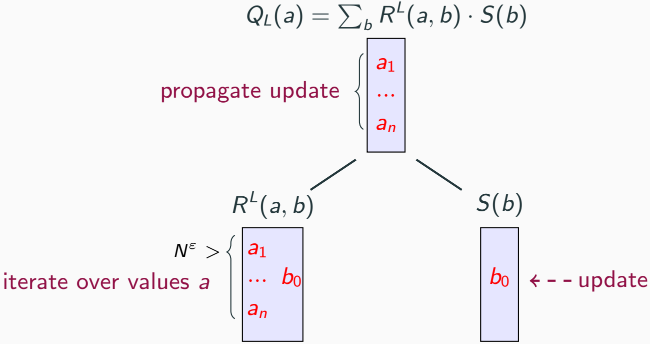
- Updates to R^L: $\mathcal{O}(1)$

Updates in the Light Case



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

Updates in the Light Case



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

propagate update

a_1

$\dots$

a_n

$R^L(a, b)$

$N^\varepsilon >$

iterate over values a

a_1

$\dots$ b_0

a_n

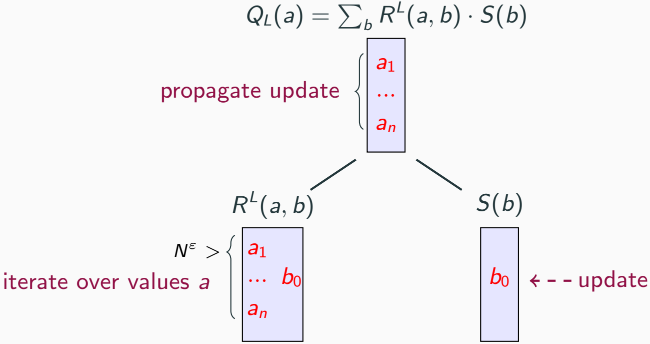
$S(b)$

b_0

$\leftarrow$ - - update

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

Updates in the Light Case



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

- Updates to R^L: $\mathcal{O}(1)$
- Updates to S: $\mathcal{O}(N^\varepsilon)$

Heavy Case

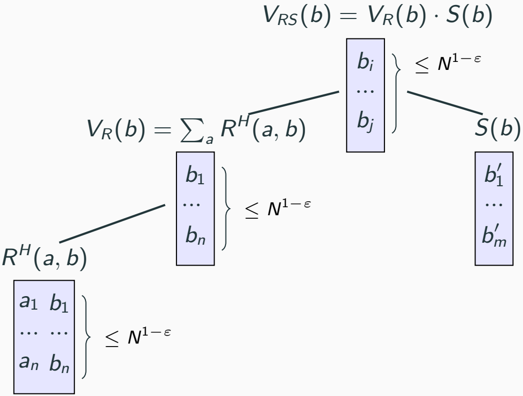
$$Q_H(a) = \sum_b R^H(a, b) \cdot S(b)$$

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



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

$$\begin{array}{|c|} \hline b_i \\ \cdots \\ b_j \\ \hline \end{array} \Big\} \leq N^{1-\varepsilon}$$

$$V_R(b) = \textstyle\sum_a R^H(a,b)$$

$$\begin{array}{|c|} \hline b_1 \\ \cdots \\ b_n \\ \hline \end{array} \Big\} \leq N^{1-\varepsilon}$$

$$S(b)$$

$$\begin{array}{|c|} \hline b'_1 \\ \cdots \\ b'_m \\ \hline \end{array}$$

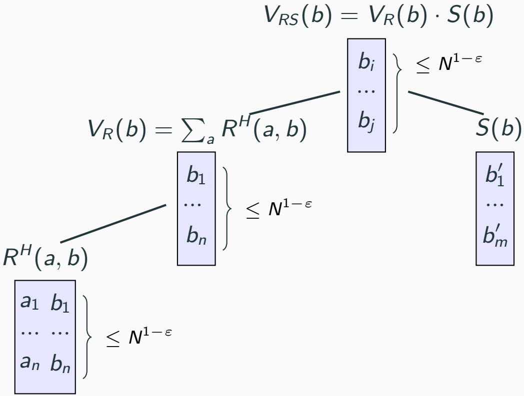
$$R^H(a,b)$$

$$\begin{array}{|cc|} \hline a_1 & b_1 \\ \cdots & \cdots \\ a_n & b_n \\ \hline \end{array} \Big\} \leq N^{1-\varepsilon}$$

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



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

b_i
$\cdots$
b_j
$\leq N^{1-\varepsilon}$

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

b_1
$\cdots$
b_n
$\leq N^{1-\varepsilon}$

$S(b)$

b'_1
$\cdots$
b'_m

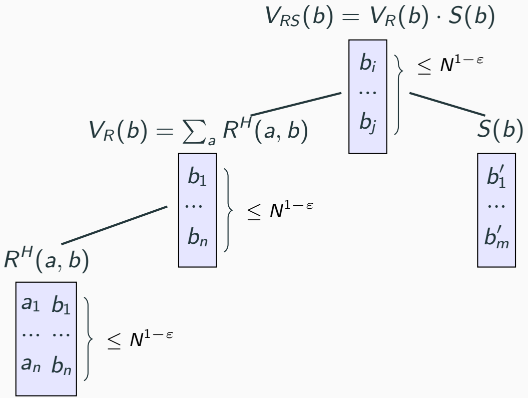
$R^H(a,b)$

$a_1 \; b_1$
$\cdots \; \cdots$
$a_n \; b_n$
$\leq N^{1-\varepsilon}$

- 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

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



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

b_i $\cdots$ b_j $\leq N^{1-\varepsilon}$

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

b_1 $\cdots$ b_n $\leq N^{1-\varepsilon}$

$S(b)$

b'_1 $\cdots$ b'_m

$R^H(a,b)$

a_1 b_1 $\cdots$ $\cdots$ a_n b_n $\leq N^{1-\varepsilon}$

- V_{RS} contains at most $N^{1-\varepsilon}$ 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

  $\Longrightarrow$ 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

  $\Longrightarrow$ 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 ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay

S_1	S_2	$S_1 \cup S_2$
a_3 a_4 a_1 a_2 **EOF**	a_5 a_6 a_2 a_4 **EOF**	

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay

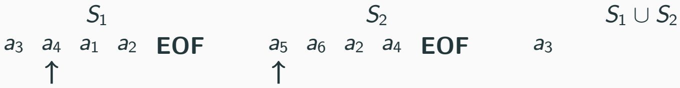
S_1	S_2	$S_1 \cup S_2$
a_3 a_4 a_1 a_2 **EOF**	a_5 a_6 a_2 a_4 **EOF**	
$\uparrow$	$\uparrow$	

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay



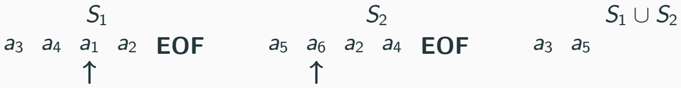
S_1					S_2					$S_1 \cup S_2$
a_3	a_4	a_1	a_2	**EOF**	a_5	a_6	a_2	a_4	**EOF**	a_3
	↑				↑					

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay



S_1

a_3 a_4 a_1 a_2 **EOF**

$\uparrow$ (under a_1)

S_2

a_5 a_6 a_2 a_4 **EOF**

$\uparrow$ (under a_6)

$S_1 \cup S_2$

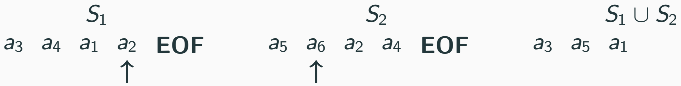
a_3 a_5

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay



S_1

a_3 a_4 a_1 a_2 **EOF**

↑

S_2

a_5 a_6 a_2 a_4 **EOF**

↑

$S_1 \cup S_2$

a_3 a_5 a_1

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay

S_1

a_3 a_4 a_1 a_2 **EOF**

$\uparrow$

S_2

a_5 a_6 a_2 a_4 **EOF**

$\uparrow$

$S_1 \cup S_2$

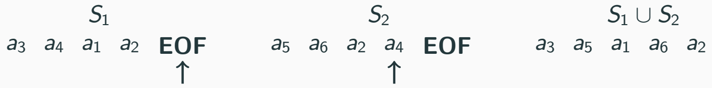
a_3 a_5 a_1 a_6

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay



S_1

a_3 a_4 a_1 a_2 **EOF**

S_2

a_5 a_6 a_2 a_4 **EOF**

$S_1 \cup S_2$

a_3 a_5 a_1 a_6 a_2

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay

S_1

$a_3 \quad a_4 \quad a_1 \quad a_2$ **EOF**

↑

S_2

$a_5 \quad a_6 \quad a_2 \quad a_4$ **EOF**

↑

$S_1 \cup S_2$

$a_3 \quad a_5 \quad a_1 \quad a_6 \quad a_2 \quad a_4$

The Union Algorithm: Example

Enumeration of the distinct tuples in the union of two sets

- Both sets allow lookup time ℓ and enumeration delay d

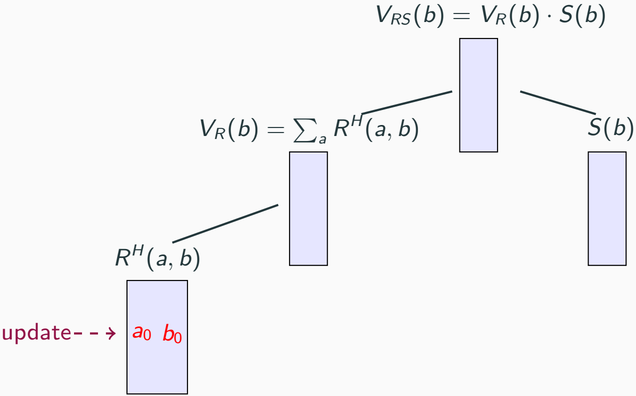
$\Longrightarrow$ The union of the sets can be enumerated with $\mathcal{O}(\ell + d)$ delay

S_1	S_2	$S_1 \cup S_2$
a_3 a_4 a_1 a_2 **EOF**	a_5 a_6 a_2 a_4 **EOF**	a_3 a_5 a_1 a_6 a_2 a_4
↑		

Generalization: Enumeration from the union of n sets

- Each set allows lookup time ℓ and enumeration delay d
- The union of the sets can be enumerated with $\mathcal{O}(n(\ell + d))$ delay

Updates in the Heavy Case



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

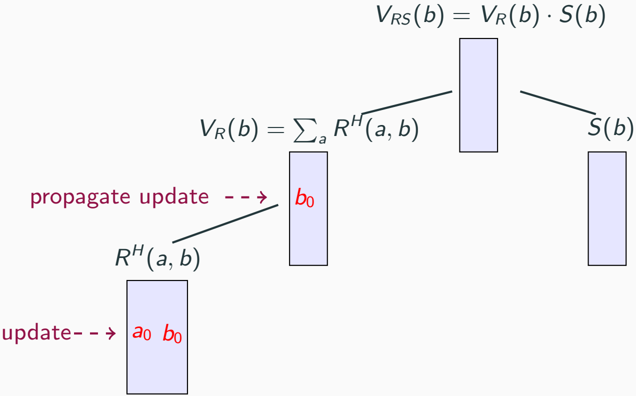
$V_R(b) = \sum_a R^H(a, b)$

$S(b)$

$R^H(a, b)$

update $\dashrightarrow$ a_0 b_0

Updates in the Heavy Case



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

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

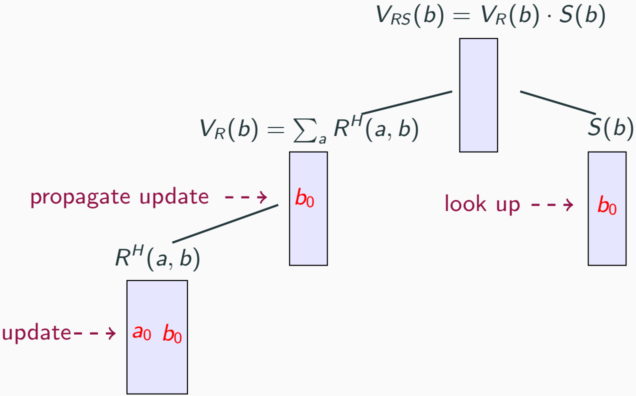
$S(b)$

propagate update - - → b_0

$R^H(a, b)$

update- - → a_0 b_0

Updates in the Heavy Case



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

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

$S(b)$

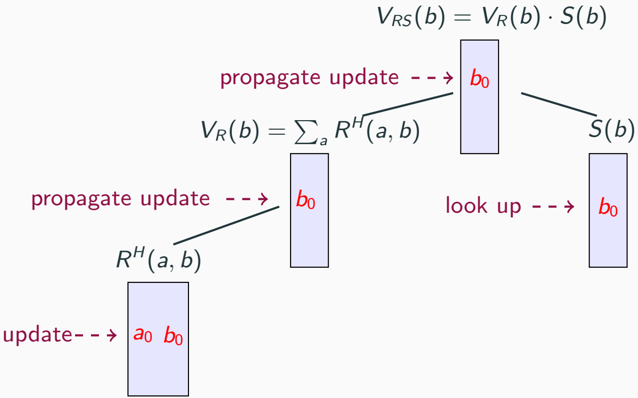
propagate update $\dashrightarrow$ b_0

look up $\dashrightarrow$ b_0

$R^H(a, b)$

update $\dashrightarrow$ a_0 b_0

Updates in the Heavy Case



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

propagate update $\dashrightarrow$ b_0

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

$S(b)$

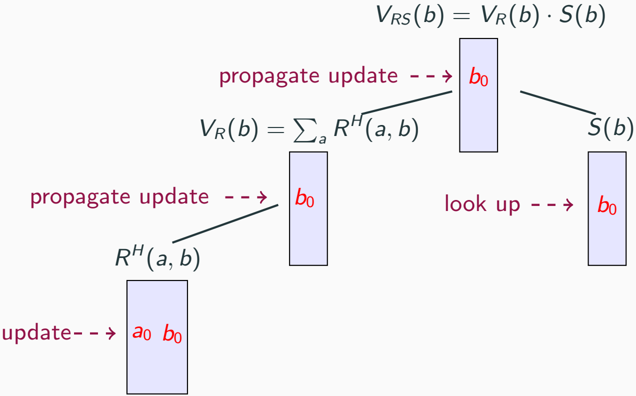
propagate update $\dashrightarrow$ b_0

look up $\dashrightarrow$ b_0

$R^H(a,b)$

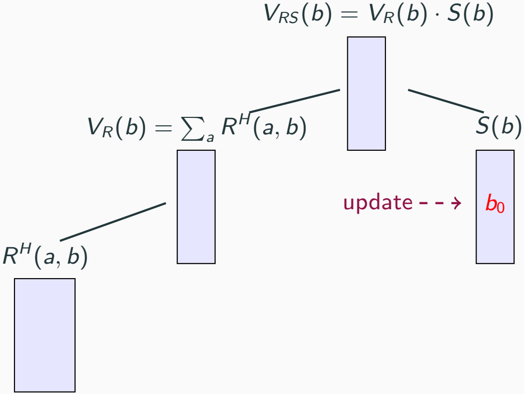
update $\dashrightarrow$ a_0 b_0

Updates in the Heavy Case



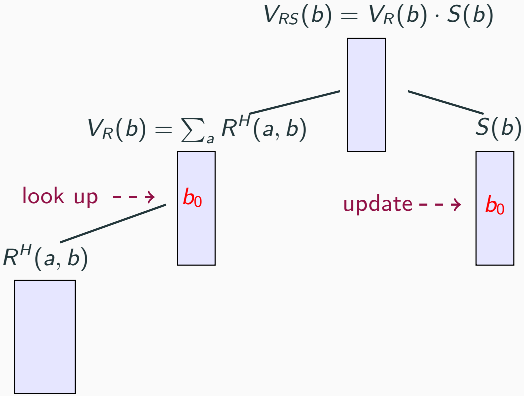
- Updates to R^H: $\mathcal{O}(1)$

Updates in the Heavy Case



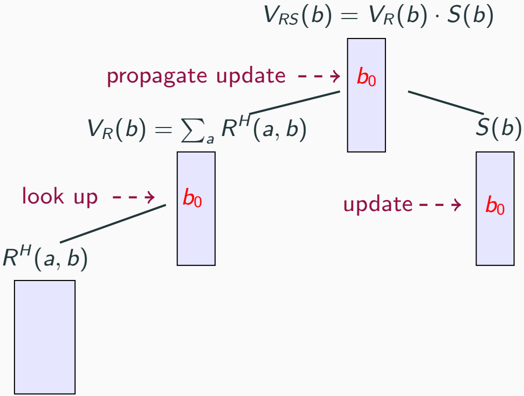
- Updates to R^H: $\mathcal{O}(1)$

Updates in the Heavy Case



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

Updates in the Heavy Case



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

propagate update $\dashrightarrow$ b_0

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

$S(b)$

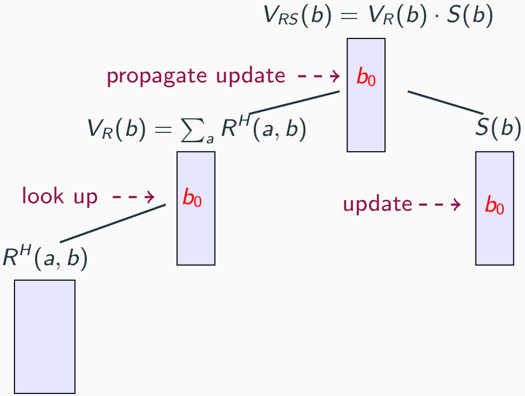
look up $\dashrightarrow$ b_0

update $\dashrightarrow$ b_0

$R^H(a, b)$

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

Updates in the Heavy Case



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

propagate update

b_0

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

$S(b)$

look up

b_0

update

b_0

$R^H(a, b)$

- 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

light case	heavy case	overall
$\mathcal{O}(N)$	$\mathcal{O}(N^{1-\varepsilon})$	$\mathcal{O}(N)$

Enumeration Delay

light case	heavy case	overall
$\mathcal{O}(1)$	$\mathcal{O}(N^{1-\varepsilon})$	$\mathcal{O}(N^{1-\varepsilon})$

Update Time

light case	heavy case	overall
$\mathcal{O}(N^{\varepsilon})$	$\mathcal{O}(1)$	$\mathcal{O}(N^{\varepsilon})$

Are there more queries

with the same

weak Pareto optimality

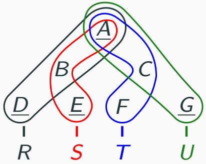
as our previous example?

δ_1-Hierarchical Queries

- For any bound variable X and any atom α of X, there is at most one other atom β so that all free variables dominated by X are covered by α and β together
- The query is hierarchical and not q-hierarchical

δ_1-hierarchical

$$Q(a,d,e,g) = R(a,b,d) \cdot {\color{red}S}(a,b,e) \cdot {\color{blue}T}(a,c,f) \cdot {\color{green}U}(a,c,g)$$

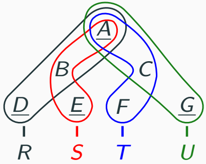


δ_1-Hierarchical Queries

- For any bound variable X and any atom α of X, there is at most one other atom β so that all free variables dominated by X are covered by α and β together
- The query is hierarchical and not q-hierarchical

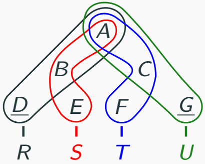
δ_1-hierarchical

$Q(a, d, e, g) = R(a, b, d) \cdot S(a, b, e) \cdot$
$T(a, c, f) \cdot U(a, c, g)$



hierarchical but not δ_1-hierarchical

$Q(d, g) = R(a, b, d) \cdot S(a, b, e) \cdot$
$T(A, C, F) \cdot U(a, c, g)$



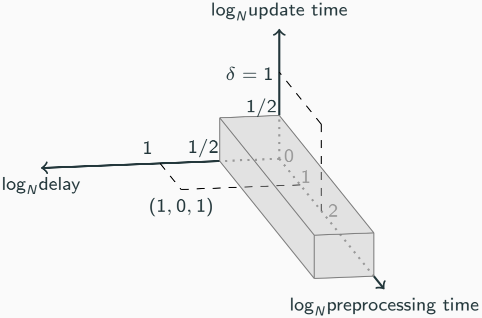
Optimality for δ_1-Hierarchical Queries

- For any δ_1-hierarchical query, there is no algorithm that admits

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

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

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



Optimality for δ_1-Hierarchical Queries

- For any δ_1-hierarchical query, there is no algorithm that admits

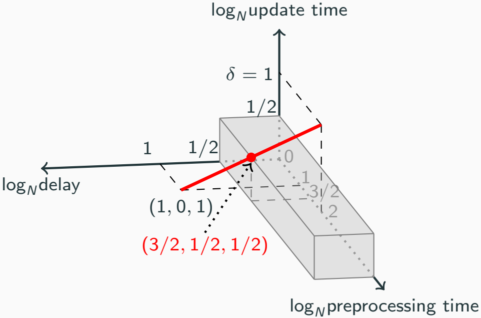
preprocessing time	update time	enumeration delay
arbitrary	$\mathcal{O}(N^{1/2-\gamma})$	$\mathcal{O}(N^{1/2-\gamma})$

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

- Any δ_1-hierarchical query can be maintained with

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

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



Optimality for δ_1-Hierarchical Queries

- For any δ_1-hierarchical query, there is no algorithm that admits

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

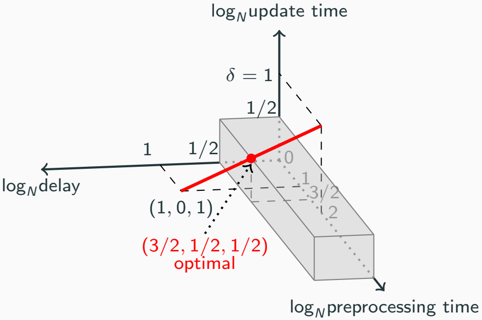
  for any $\gamma > 0$, unless the OMv Conjecture (*) fails

- Any δ_1-hierarchical query can be maintained with

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

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

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



References i

[VLDB 2004] Nilesh N. Dalvi, Dan Suciu. *Efficient Query Evaluation on Probabilistic Databases.*

[ICDE 2009] Dan Olteanu, Jiewen Huang, Christoph Koch. *SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases.*

[CSL 2011] Arnaud Durand, Yann Strozecki. *Enumeration Complexity of Logical Query Problems with Second-order Variables.* CSL 2011

[ICDT 2012] Dan Olteanu, Jakub Zavodny. *Factorised representations of query results: size bounds and readability.*

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

References ii

[SIGMOD 2018] Milos Nikolic, Dan Olteanu. *Incremental View Maintenance with Triple Lock Factorization Benefits.*

[ICDT 2023] Ahmet Kara, Milos Nikolic, Dan Olteanu, Haozhe Zhang. *Conjunctive Queries with Free Access Patterns Under Updates.*

[VLDBJ 2023] Ahmet Kara, Milos Nikolic, Dan Olteanu, Haozhe Zhang. *F-IVM: Analytics over Relational Databases under Updates.* (To appear)

[LMCS 2023] Ahmet Kara, Milos Nikolic, Dan Olteanu, Haozhe Zhang. *Trade-offs in Static and Dynamic Evaluation of Hierarchical Queries.*

Thank You!