Trade-offs in Static and Dynamic Query Evaluation

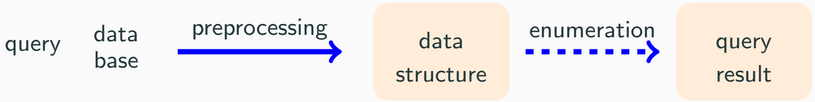
Ahmet Kara, Milos Nikolic
Dan Olteanu, and Haozhe Zhang

fdbresearch.github.io

Dagstuhl Seminar 20071, Feb. 2020
Foundations of Composite Event Recognition

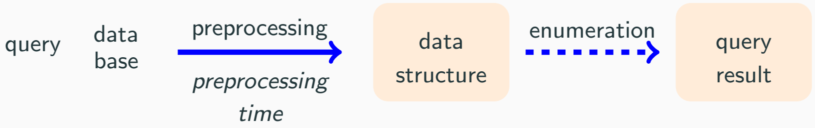
Static and Dynamic Query Evaluation

Static Query Evaluation



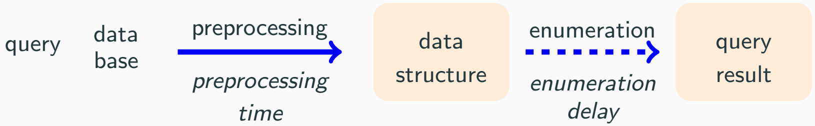
Static and Dynamic Query Evaluation

Static Query Evaluation



Static and Dynamic Query Evaluation

Static Query Evaluation



Static and Dynamic Query Evaluation

Static Query Evaluation

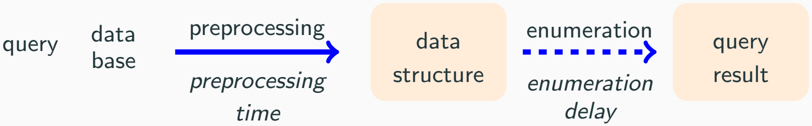
query, data base $\xrightarrow[\textit{preprocessing time}]{\text{preprocessing}}$ data structure $\xrightarrow[\textit{enumeration delay}]{\text{enumeration}}$ query result

Dynamic Query Evaluation

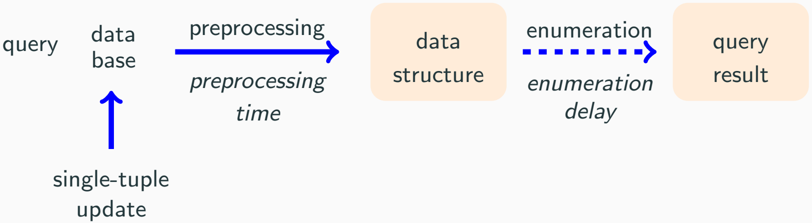
query, data base $\xrightarrow[\textit{preprocessing time}]{\text{preprocessing}}$ data structure $\xrightarrow[\textit{enumeration delay}]{\text{enumeration}}$ query result

Static and Dynamic Query Evaluation

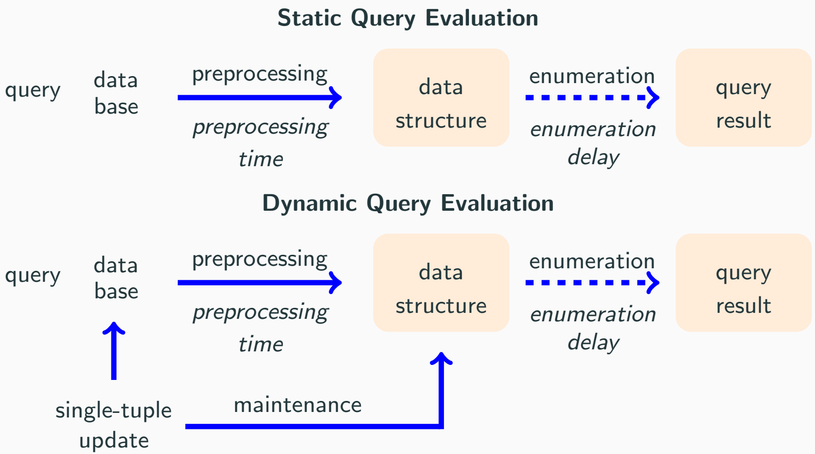
Static Query Evaluation



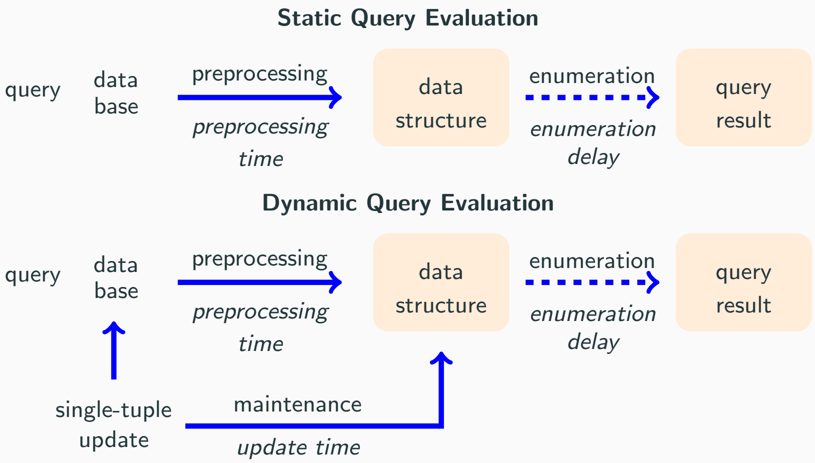
Dynamic Query Evaluation



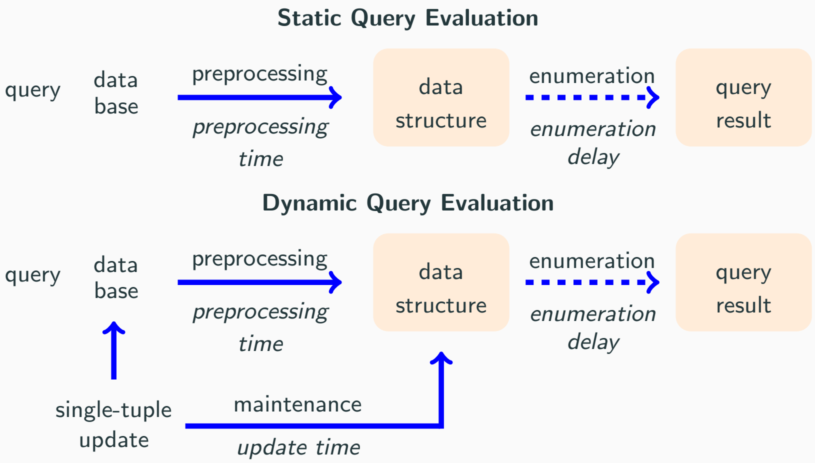
Static and Dynamic Query Evaluation



Static and Dynamic Query Evaluation



Static and Dynamic Query Evaluation



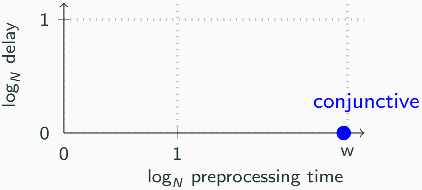
We are interested in the trade-off between:

preprocessing time - enumeration delay - (update time)

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay

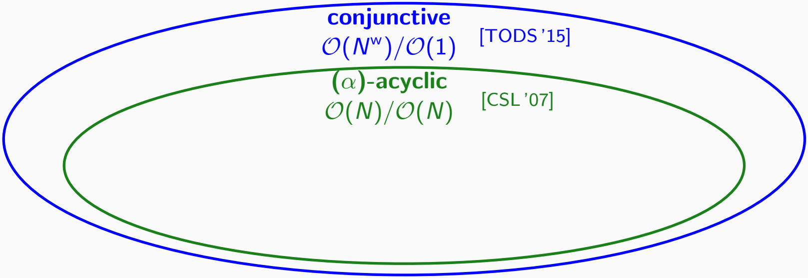
conjunctive
$\mathcal{O}(N^{\mathrm{w}})/\mathcal{O}(1)$ [TODS '15]

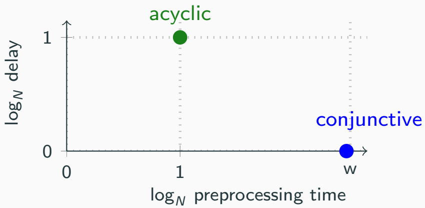


static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay

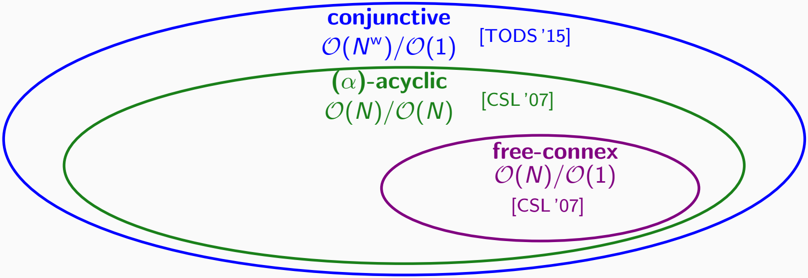


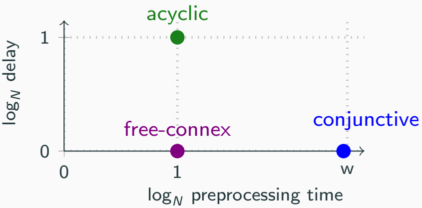


static width $\mathrm{w} = s^{\uparrow}$ [TODS'15] or faqw [PODS'16]

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay

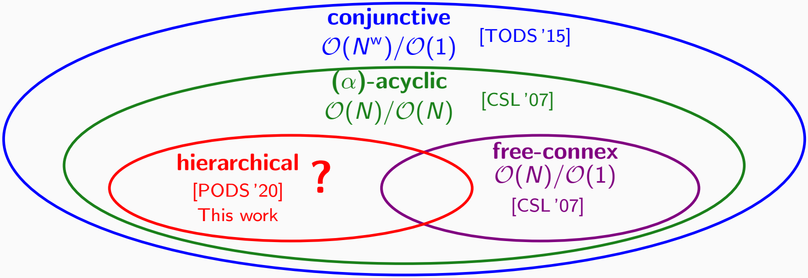


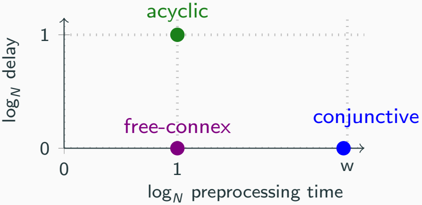


static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay

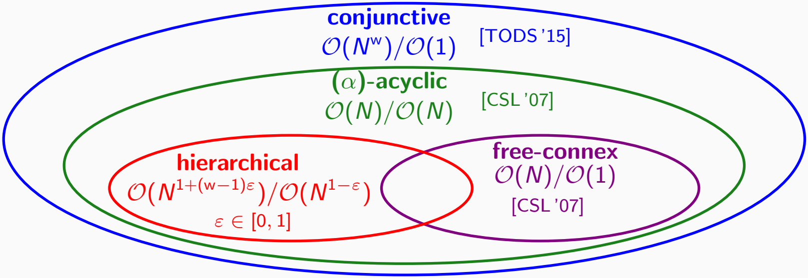


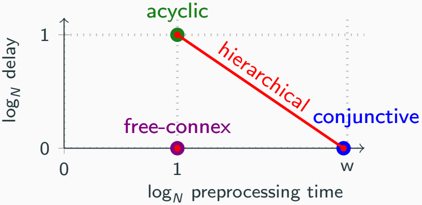


static width w $= s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay

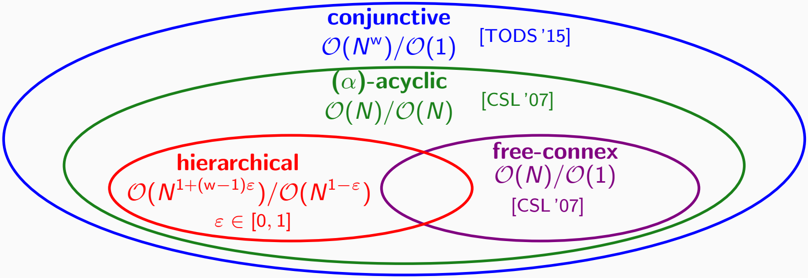


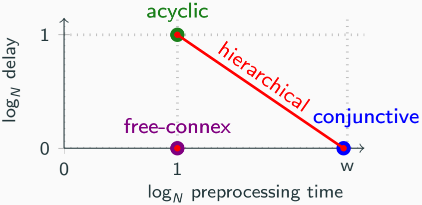


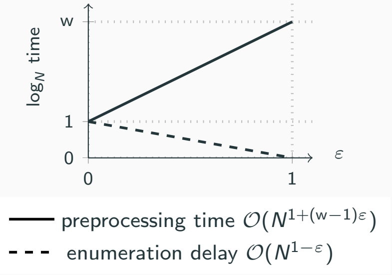
static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay



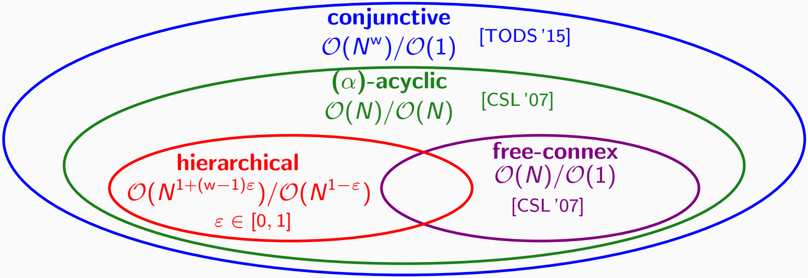


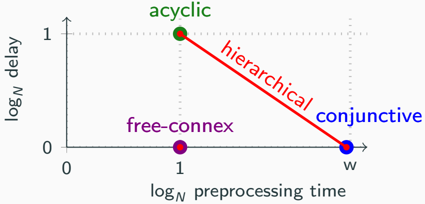


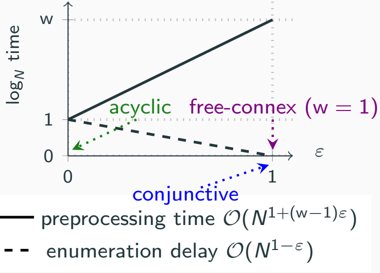
static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

Landscape of Static Query Evaluation

Preprocessing time/Enumeration delay







static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay

conjunctive

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

static width $\mathsf{w} = s^{\uparrow}$ [TODS'15] or faqw [PODS'16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS'20]

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay

conjunctive

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

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

static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS '20]

$(*)$: amortized update time

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay

conjunctive

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

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

(α-)acyclic

joins $\mathcal{O}(N)/\mathcal{O}(N)/\mathcal{O}(1)$ [SIGMOD '17]

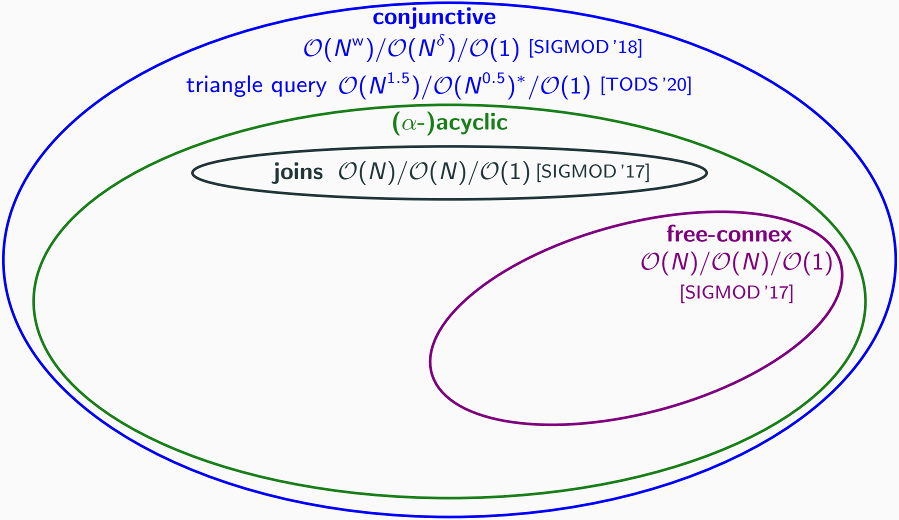
static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS '20]

$(*)$: amortized update time

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay



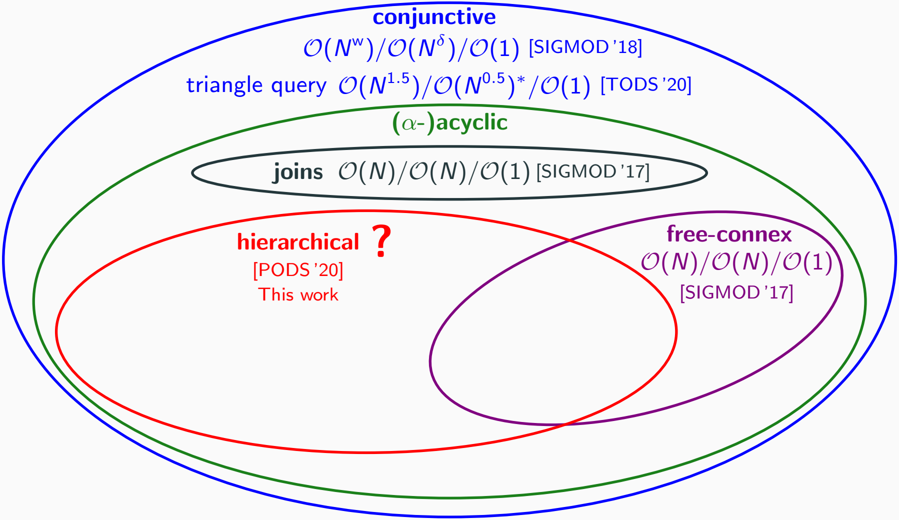
static width $\mathsf{w} = s^{\uparrow}$[TODS '15] or faqw [PODS '16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS '20]

($*$): amortized update time

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay



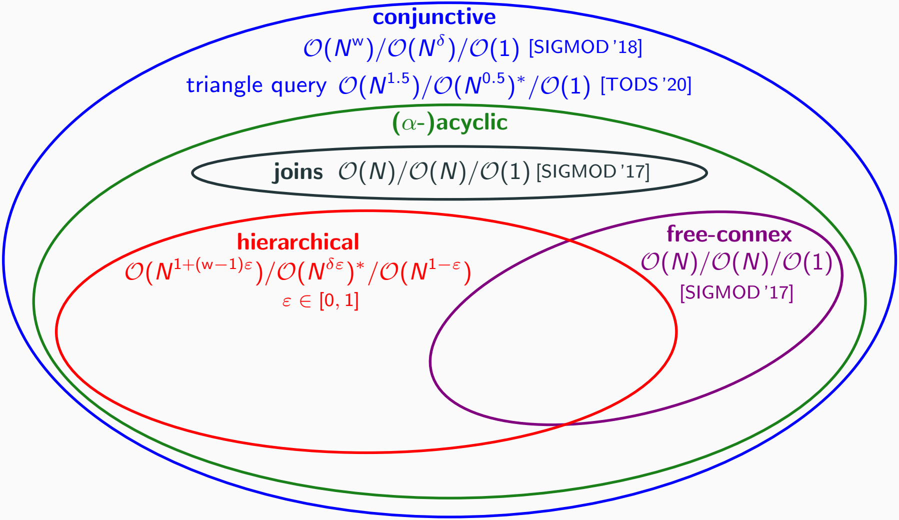
static width $\mathrm{w} = s^{\uparrow}$ [TODS '15] or faqw [PODS '16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS '20]

$(*)$: amortized update time

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay



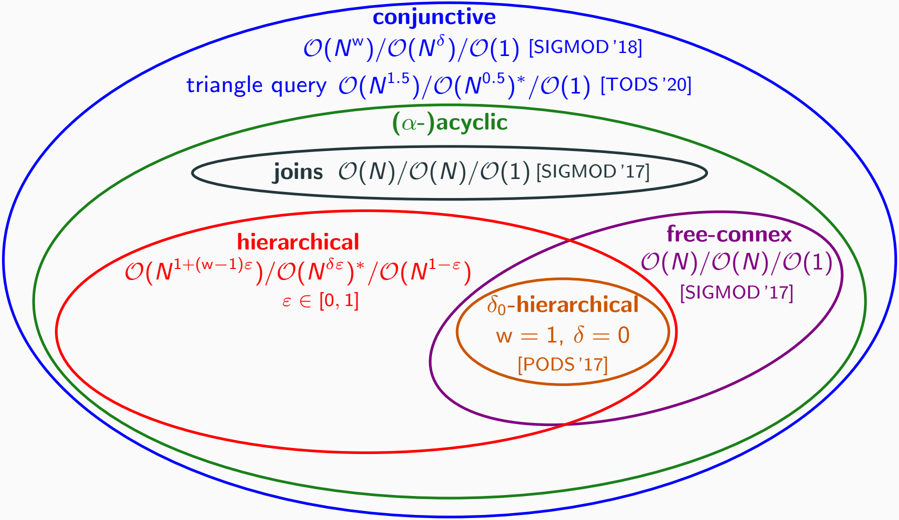
static width $\mathrm{w} = s^{\uparrow}$[TODS '15] or faqw [PODS '16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS '20]

$(*)$: amortized update time

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay



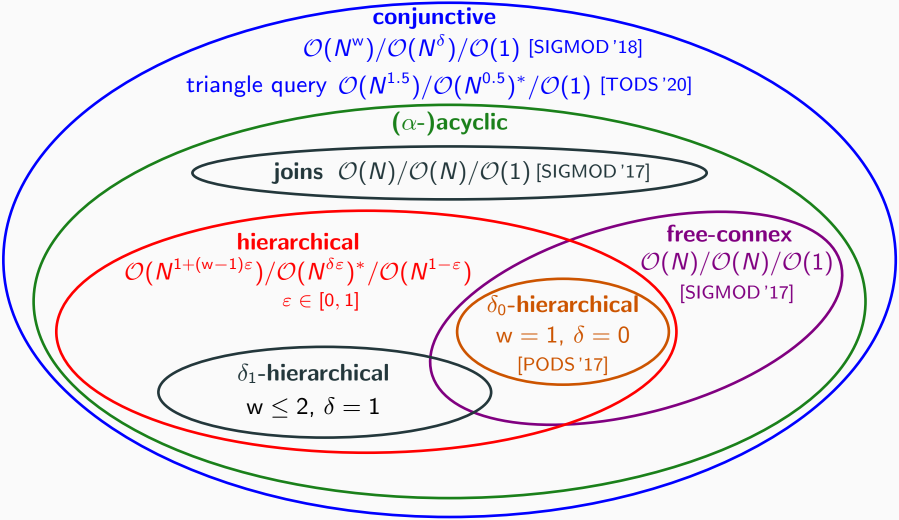
static width $\mathrm{w} = s^{\uparrow}$[TODS'15] or faqw [PODS'16]

dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS'20]

$(*)$: amortized update time

Landscape of Dynamic Query Evaluation

Preprocessing time/Update time/Enumeration delay

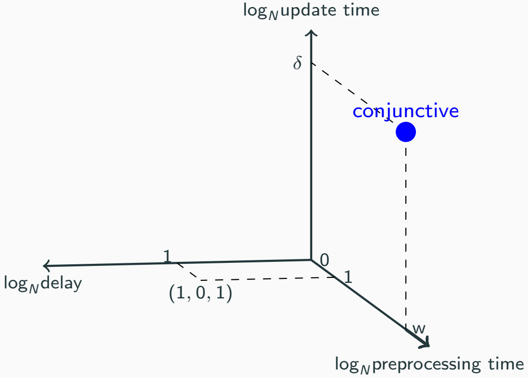


static width $\mathrm{w} = s^{\uparrow}$[TODS '15] or faqw [PODS '16]

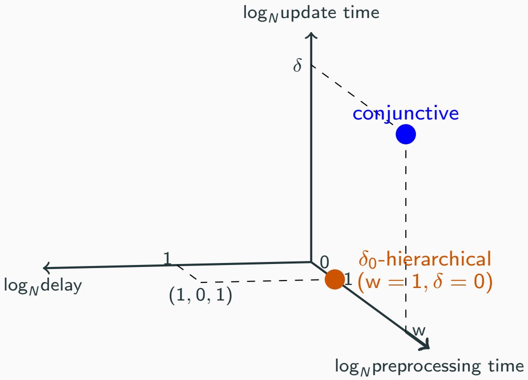
dynamic width $\delta = \max\limits_{\text{delta queries}}$ static width [PODS '20]

($*$): amortized update time

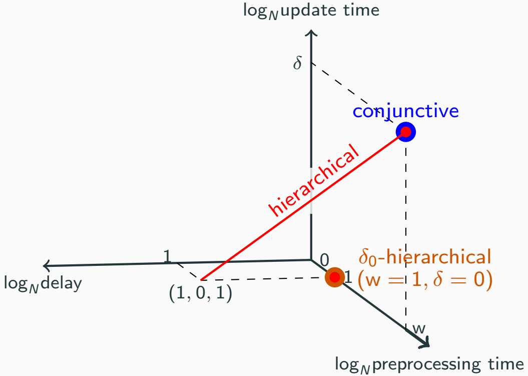
1. Recovery of Prior Approaches



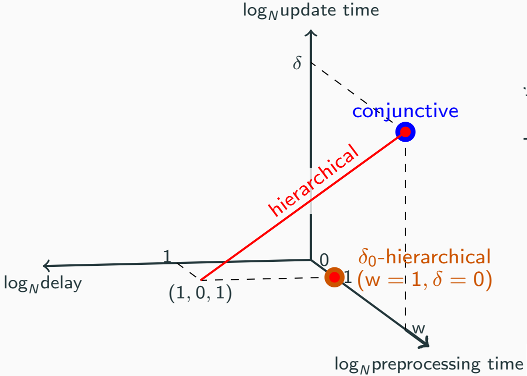
1. Recovery of Prior Approaches

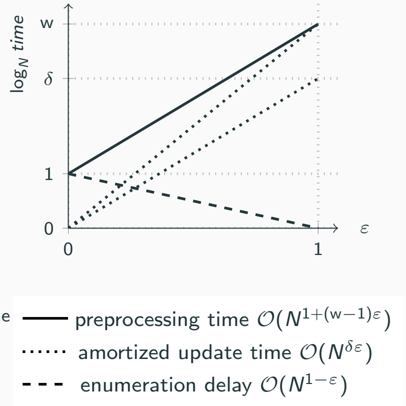


1. Recovery of Prior Approaches

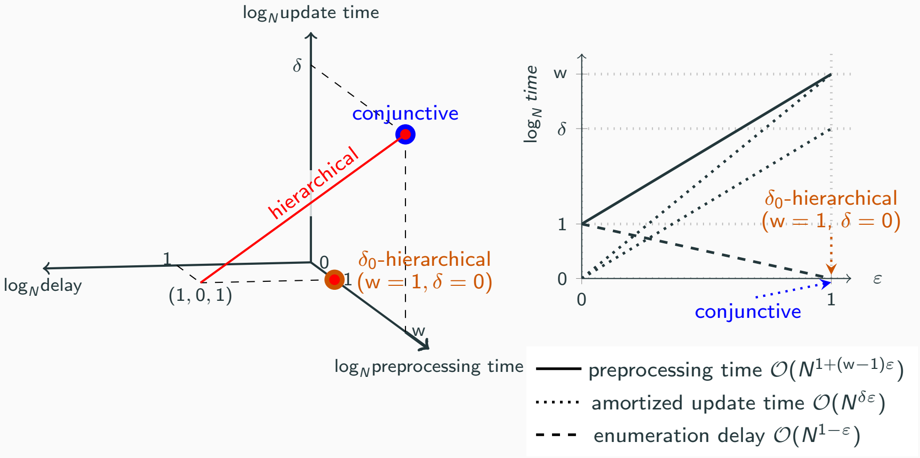


1. Recovery of Prior Approaches

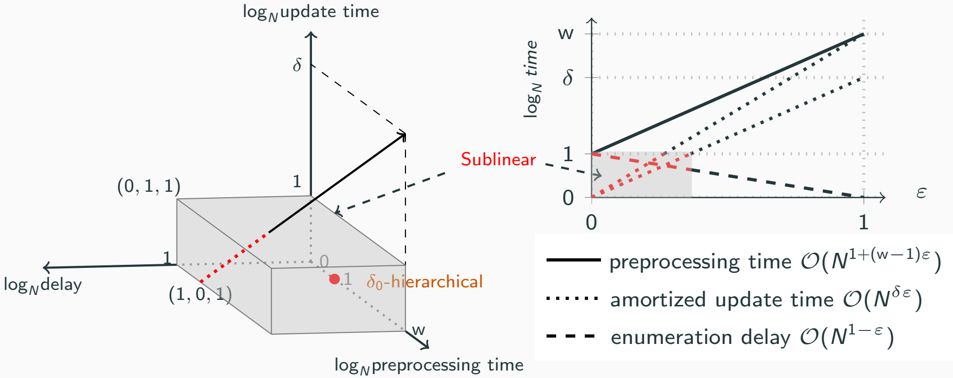




1. Recovery of Prior Approaches



2. Sublinear Update Time and Delay



- First approach with sublinear amortized update time and enumeration delay for hierarchical queries.

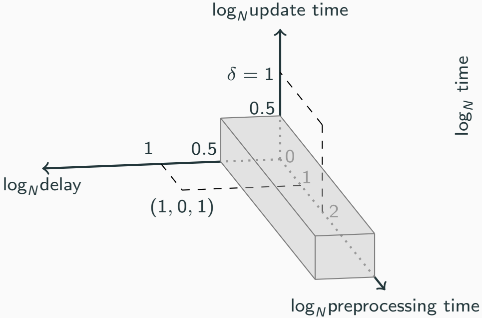
3. Optimality for δ_1-Hierarchical Queries

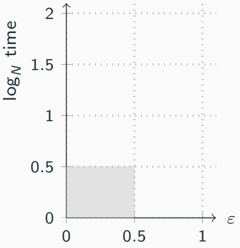
- For any δ_1-hierarchical query, there is no algorithm that admits

preprocessing time	amortized update time	enumeration delay
arbitrary	$\mathcal{O}(N^{0.5-\gamma})$	$\mathcal{O}(N^{0.5-\gamma})$

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

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





3. Optimality for δ_1-Hierarchical Queries

- For any δ_1-hierarchical query, there is no algorithm that admits

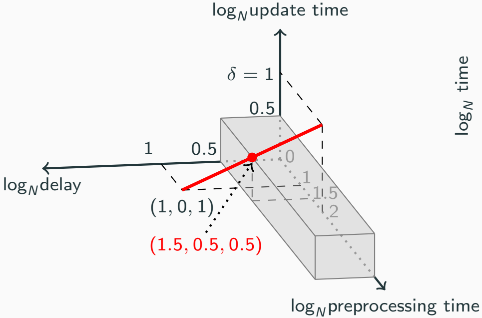
preprocessing time	amortized update time	enumeration delay
arbitrary	$\mathcal{O}(N^{0.5-\gamma})$	$\mathcal{O}(N^{0.5-\gamma})$

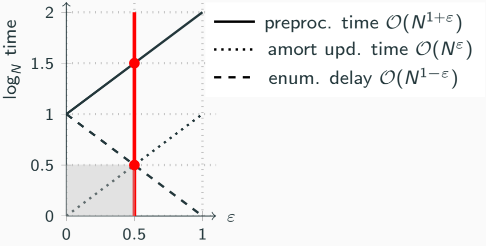
  for any $\gamma > 0$, unless the OMv Conjecture (*) fails.

- Our approach maintains any δ_1-hierarchical query with

preprocessing time	amortized 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.





3. Optimality for δ_1-Hierarchical Queries

- For any δ_1-hierarchical query, there is no algorithm that admits

preprocessing time	amortized update time	enumeration delay
arbitrary	$\mathcal{O}(N^{0.5-\gamma})$	$\mathcal{O}(N^{0.5-\gamma})$

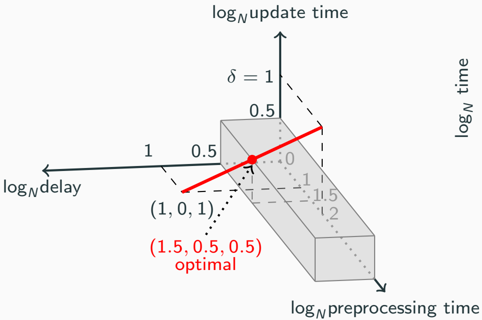
  for any $\gamma > 0$, unless the OMv Conjecture (*) fails.

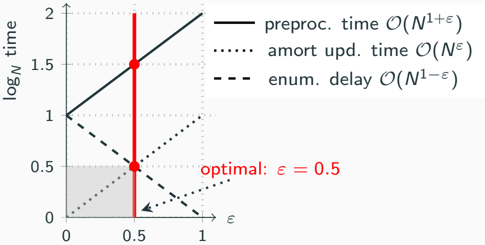
- Our approach maintains any δ_1-hierarchical query with

preprocessing time	amortized update time	enumeration delay
$\mathcal{O}(N^{1+\varepsilon})$	$\mathcal{O}(N^{\varepsilon})$	$\mathcal{O}(N^{1-\varepsilon})$.

$\Longrightarrow$ For $\varepsilon = 0.5$, this is weakly Pareto optimal, unless OMv Conjecture fails.

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





4. Single-Tuple vs Bulk Updates

$\delta = \mathsf{w} - 1$ or $\delta = \mathsf{w}$ for hierarchical queries.

Case $\delta = \mathsf{w} - 1$

Time to insert N tuples: $\mathcal{O}(N \cdot N^{(\mathsf{w}-1)\varepsilon}) = \mathcal{O}(N^{1+(\mathsf{w}-1)\varepsilon})$.

$\Longrightarrow$ Preprocessing can be simulated by executing N single-tuple updates.

4. Single-Tuple vs Bulk Updates

$\delta = \mathsf{w} - 1$ or $\delta = \mathsf{w}$ for hierarchical queries.

Case $\delta = \mathsf{w} - 1$

Time to insert N tuples: $\mathcal{O}(N \cdot N^{(\mathsf{w}-1)\varepsilon}) = \mathcal{O}(N^{1+(\mathsf{w}-1)\varepsilon})$.

$\Longrightarrow$ Preprocessing can be simulated by executing N single-tuple updates.

Case $\delta = \mathsf{w}$

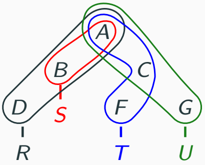
Time to insert N tuples: $\mathcal{O}(N \cdot N^{\mathsf{w}\varepsilon}) = \mathcal{O}(N^{1+(\mathsf{w}-1)\varepsilon+\varepsilon})$.

$\Longrightarrow$ Complexity gap of $\mathcal{O}(N^{\varepsilon})$ between single-tuple updates and bulk updates.

Hierarchical Queries

A query is hierarchical if for any two variables X, Y:
$atoms(X) \subseteq atoms(Y)$ or $atoms(X) \supseteq atoms(Y)$ or $atoms(X) \cap atoms(Y) = \emptyset$

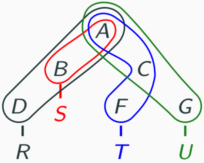
hierarchical
$Q(\mathcal{F}) = R(A, B, D), S(A, B),$
$T(A, C, F), U(A, C, G)$
$\mathcal{F}$ any set of variables



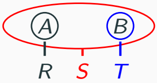
Hierarchical Queries

A query is hierarchical if for any two variables X, Y:
$atoms(X) \subseteq atoms(Y)$ or $atoms(X) \supseteq atoms(Y)$ or $atoms(X) \cap atoms(Y) = \emptyset$

hierarchical
$Q(\mathcal{F}) = R(A, B, D), S(A, B),$
$T(A, C, F), U(A, C, G)$
$\mathcal{F}$ any set of variables



not hierarchical
$Q(\mathcal{F}) = R(A), S(A, B), T(B)$
$\mathcal{F}$ any set of variables

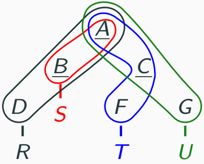


δ_0-Hierarchical Queries

A hierarchical query is δ_0-hierarchical if
all free variables dominate the bound ones.

δ_0-hierarchical

$$Q(A,B,C) = R(A,B,D), S(A,B), T(A,C,F), U(A,C,G)$$

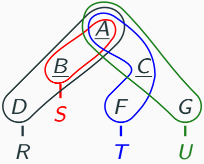


δ_0-Hierarchical Queries

A hierarchical query is δ_0-hierarchical if
all free variables dominate the bound ones.

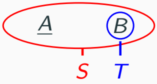
δ_0-hierarchical

$Q(A, B, C) = R(A, B, D), S(A, B),$
$T(A, C, F), U(A, C, G)$



hierarchical but not
δ_0-hierarchical

$Q(A) = S(A, B), T(B)$

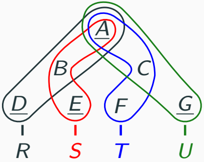


δ_1-Hierarchical Queries

- For any bound variable X and any atom using X, we need at most one further atom to cover all free variables dominated by X.
- The query is not δ_0-hierarchical.

δ_1-hierarchical

$$Q(A, D, E, G) = R(A, B, D), \textcolor{red}{S}(A, B, E),$$
$$\textcolor{blue}{T}(A, C, F), \textcolor{green}{U}(A, C, G)$$

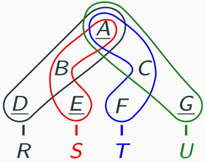


δ_1-Hierarchical Queries

- For any bound variable X and any atom using X, we need at most one further atom to cover all free variables dominated by X.
- The query is not δ_0-hierarchical.

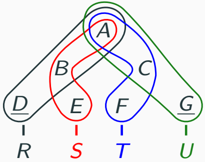
δ_1-hierarchical

$Q(A, D, E, G) = R(A, B, D), S(A, B, E), T(A, C, F), U(A, C, G)$



hierarchical but not δ_1-hierarchical

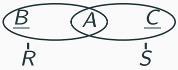
$Q(D, G) = R(A, B, D), S(A, B, E), T(A, C, F), U(A, C, G)$



Trade-Off in Static Query Evaluation: Example

Simple δ_1-hierarchical query

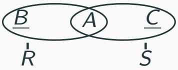
$$Q(B, C) = R(A, B), S(A, C)$$

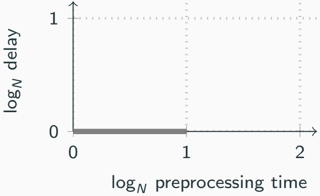


Trade-Off in Static Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(B, C) = R(A, B), S(A, C)$$





Lower bound [CSL '07]

There is no algorithm that admits

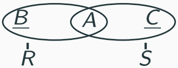
preprocessing time	enumeration delay
$\mathcal{O}(N)$	$\mathcal{O}(1)$

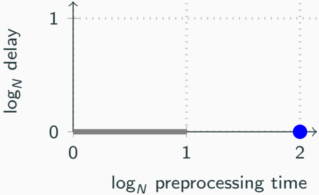
unless Boolean Matrix Multiplication can be solved in quadratic time.

Trade-Off in Static Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(B, C) = R(A, B), S(A, C)$$





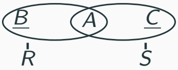
Known approach: Eager preprocessing, quick enumeration

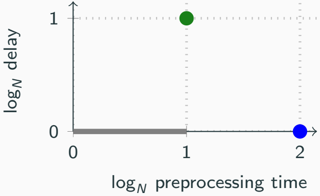
- Preprocessing: Materialize the result.
- Enumeration: Enumerate from materialized result.

Trade-Off in Static Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(B, C) = R(A, B), S(A, C)$$





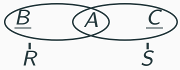
Known approach: Lazy preprocessing, heavy enumeration

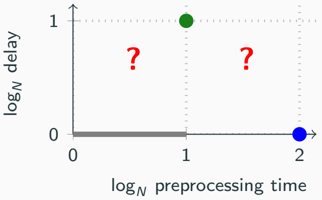
- Preprocessing: Eliminate dangling tuples.
- Enumeration: For each B-value, enumerate distinct C-values.

Trade-Off in Static Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(B, C) = R(A, B), S(A, C)$$





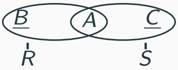
Open question

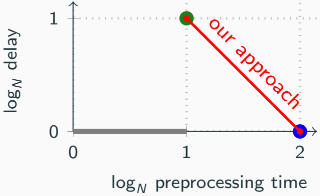
Is there an algorithm that admits sub-quadratic preprocessing time and sub-linear enumeration delay?

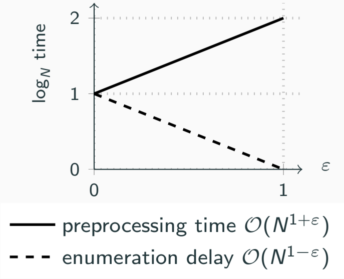
Trade-Off in Static Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(B,C) = R(A,B), S(A,C)$$



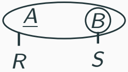




Trade-Off in Dynamic Query Evaluation: Example

Simple δ_1-hierarchical query

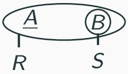
$$Q(A) = R(A, B), S(B)$$

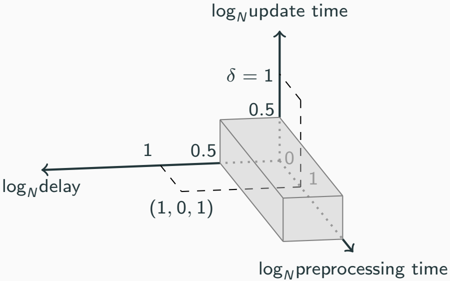


Trade-Off in Dynamic Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(A) = R(A, B), S(B)$$





Lower bound

For this query, there is no algorithm that admits

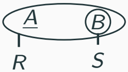
preprocessing time	amortized update time	enumeration delay
arbitrary	$\mathcal{O}(N^{0.5-\gamma})$	$\mathcal{O}(N^{0.5-\gamma})$

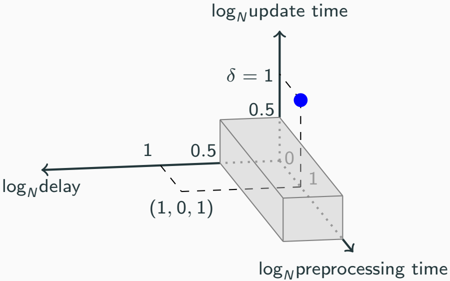
for any $\gamma > 0$, unless the OMv Conjecture fails.

Trade-Off in Dynamic Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(A) = R(A, B), S(B)$$





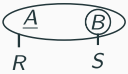
Known approach: Eager update, quick enumeration

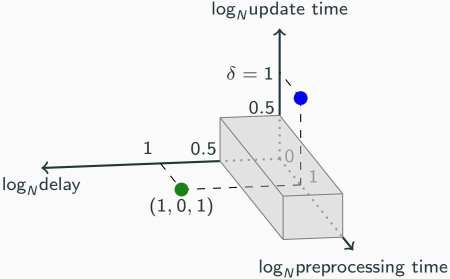
- Preprocessing: Materialize the result.
- Upon update: Maintain the materialized result.
- Enumeration: Enumerate from materialized result.

Trade-Off in Dynamic Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(A) = R(A, B), S(B)$$





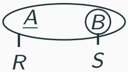
Known approach: Lazy update, heavy enumeration

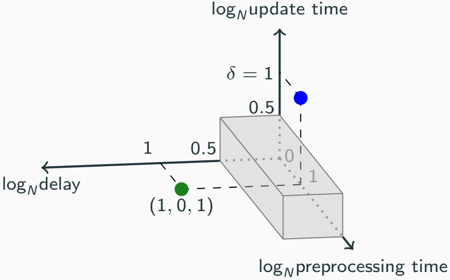
- Preprocessing: Eliminate dangling tuples.
- Upon update: Update only base relations.
- Enumeration: Eliminate dangling tuples and enumerate from R.

Trade-Off in Dynamic Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(A) = R(A, B), S(B)$$





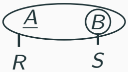
Open question

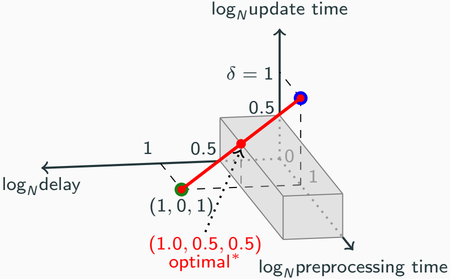
Is there an algorithm that admits
sub-linear (amortized) update time and sub-linear enumeration delay?

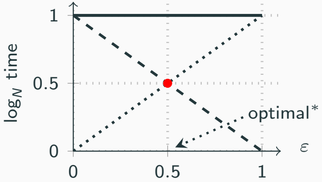
Trade-Off in Dynamic Query Evaluation: Example

Simple δ_1-hierarchical query

$$Q(A) = R(A, B), S(B)$$







(∗): Weak Pareto optimality by OMv Conjecture

- preprocessing time $\mathcal{O}(N^1)$
- amortized update time $\mathcal{O}(N^\varepsilon)$
- enumeration delay $\mathcal{O}(N^{1-\varepsilon})$

Conclusion

Benefits of Our Approach

- Allows to tune the trade-off between preprocessing time, update time, and enumeration delay.
- Recovers existing results as specific points.
- Maintains hierarchical queries with sub-linear amortized update time and enumeration delay.
- Maintains δ_1-queries with weakly Pareto optimal update time and delay.

Ongoing Work

- Extension of our approach to
  - conjunctive queries,
  - aggregate queries, and
  - enumeration in desired order.
- System prototype.