VLDB 2025
WELCOME TO LONDON

# Path-Centric Cardinality Estimation for Subgraph Matching

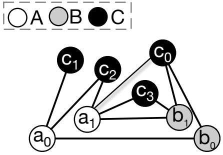
Zhengdong Wang (SJTU), Qiang Yin (SJTU), Longbin Lai (Alibaba)

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

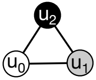
Alibaba

September 3, 2025

# Introduction




(a) Graph G




(b) Query Q

- $Q(G)$ has two matches
  - $u_0 \to a_1$, $u_1 \to b_1$ and $u_2 \to c_0$
  - $u_0 \to a_1$, $u_1 \to b_1$ and $u_2 \to c_3$
- $|Q(G)| = 2$.

- Subgraph matching: find all homomorphic matches of a query $Q$ in graph $G$; a fundamental building block in graph query languages (e.g., Cypher, GQL).
- Cardinality estimation: estimate $|Q(G)|$ without explicit computation; crucial for cost-based query optimization.
- Extensively studied in relational databases, but still underdeveloped for graph data.

# Existing Approaches

- Summary-based methods
  - Build statistics from small queries and combine them to estimate $|Q(G)|$.
  - Rely on graph data rather than specific queries.
  - Examples: CEG, SumRDF, Color, GLogS.
- Sampling-based methods
  - Estimate $|Q(G)|$ by executing $Q$ on random samples of $G$ and scaling the results.
  - Provide good accuracy under correlations and skewed data.
  - May suffer from high failure rates on cyclic queries.
- ML-based methods
  - Learn predictive models from data or queries.
  - Support both data-driven and query-driven approaches.
  - High training cost; often act as black boxes.

We focus on summary-based approaches in this work.

# Motivation

A summary-based estimator typically performs cardinality estimation iteratively.

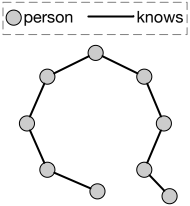



Figure: Query $\mathrm{Q}$

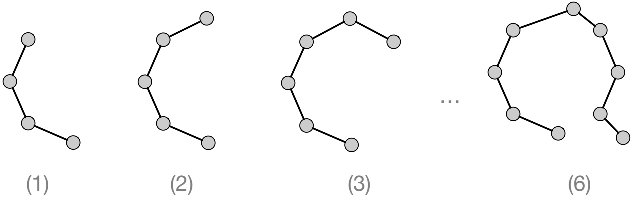



Figure: Iterative estimation using GLogS[1]

Each iteration estimates a subquery of $\mathrm{Q}$; we refer to each step as an estimation iteration.

[1] GLogS: Interactive graph pattern matching query at large scale. ATC 2023.

# Motivation (cont'd)

Example. Let's estimate $|Q(G)|$ using existing summary-based estimators.

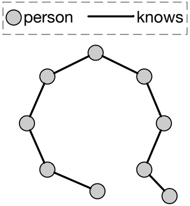



Figure: Query $Q$

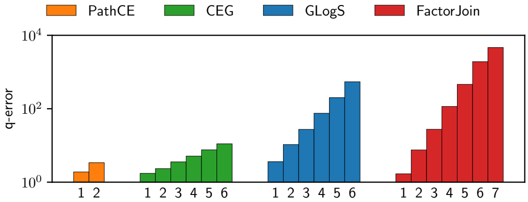



Figure: Estimation accuracy of subqueries across iterations

Observation. More iterations $\rightarrow$ higher Q-error (error accumulation).

Question. How to reduce estimation iterations and improve accuracy?

# Accuracy vs. Efficiency

| | Estimation Accuracy | Construction Efficiency |
|---|---|---|
| Edge & Vertex | ☹️ | 😃 |
| Triangle query | 😃 | ☹️ |
| Path query | 🙂 | 🙂 |

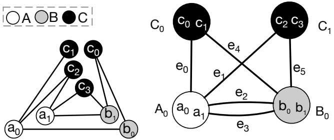
- Utilizing statistics of generic queries, e.g., triangle counts, reduces estimation iterations and improves accuracy[1].
- Constructing statistics like triangle counts on large graphs is prohibitive.
  - Systems like GLogS use techniques such as graph sparsification to mitigate the problem.
- Path query statistics strike a balance between accuracy and efficiency.

---

[1] Accurate Summary-based Cardinality Estimation Through the Lens of Cardinality Estimation Graphs. VLDB 2022.

# PathCE: A Path-Centric Framework

**(1)** PathCE precomputes short-path query statistics from the data graph and encodes them as a novel Path-Centric Summary Graph (PSG).




(a) Graph G (b) A PSG graph **G'**

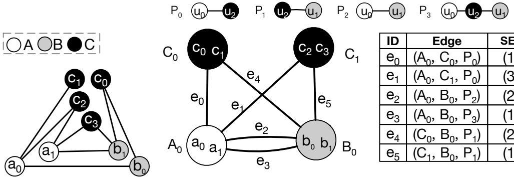
| ID | Edge | SE-triple |
|---|---|---|
| $e_0$ | $(A_0, C_0, P_0)$ | (1,1,1) |
| $e_1$ | $(A_0, C_1, P_0)$ | (3,2,2) |
| $e_2$ | $(A_0, B_0, P_2)$ | (2,1,1) |
| $e_3$ | $(A_0, B_0, P_3)$ | (1,1,1) |
| $e_4$ | $(C_0, B_0, P_1)$ | (2,2,1) |
| $e_5$ | $(C_1, B_0, P_1)$ | (1,1,1) |

(c) Edges and SE-triples

PSG stores both match counts and maximum-degree statistics for path queries.

**(2)** By using query decomposition and precomputed statistics encoded in PSG, PathCE achieves higher estimation accuracy with fewer iterations.

# Path-Centric Summary Graph




(a) Graph G

(b) A PSG graph **G'**

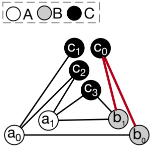
| ID | Edge | SE-triple |
|---|---|---|
| $e_0$ | $(A_0, C_0, P_0)$ | (1,1,1) |
| $e_1$ | $(A_0, C_1, P_0)$ | (3,2,2) |
| $e_2$ | $(A_0, B_0, P_2)$ | (2,1,1) |
| $e_3$ | $(A_0, B_0, P_3)$ | (1,1,1) |
| $e_4$ | $(C_0, B_0, P_1)$ | (2,2,1) |
| $e_5$ | $(C_1, B_0, P_1)$ | (1,1,1) |

(c) Edges and SE-triples

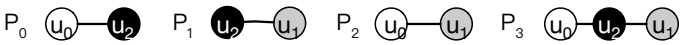
A path-centric summary graph (PSG) for a data graph G is itself a graph $G'$, where

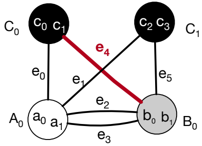
- each vertex in $G'$ represents a subset of vertices in G that share the same label;
- each edge in $G'$ represents a path query between the corresponding vertex subsets.

# Path-Centric Summary Graph (cont'd)




(a) Graph G




(b) A PSG graph **G'**

| ID | Edge | SE-triple |
|---|---|---|
| $e_0$ | $(A_0, C_0, P_0)$ | (1,1,1) |
| $e_1$ | $(A_0, C_1, P_0)$ | (3,2,2) |
| $e_2$ | $(A_0, B_0, P_2)$ | (2,1,1) |
| $e_3$ | $(A_0, B_0, P_3)$ | (1,1,1) |
| $e_4$ | $(C_0, B_0, P_1)$ | (2,2,1) |
| $e_5$ | $(C_1, B_0, P_1)$ | (1,1,1) |

(c) Edges and SE-triples

SE-triple. Each PSG edge carries an SE-triple $(c, d_1, d_2)$ that encodes path-query statistics.

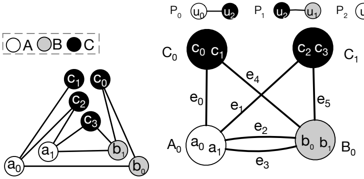
$$(c, d_1, d_2)$$

Path match count — Maximal degree information

Example. For edge $e_4 = (C_0, B_0, P_1)$, the SE-triple is $(c, d_1, d_2) = (2, 2, 1)$.

- There are 2 matches of $P_1$ in $G$, where $u_2$ (resp. $u_1$) matches a vertex in $C_0$ (resp. $B_0$).
- $d_1 = 2$ since $c_0 \in C_0$ has the maximum number of occurrences in these matches, i.e., 2.
- $d_2 = 1$ since every vertex in $B_0$ is associated with at most one $P_1$ match.

# Parallel PSG Construction




(a) Graph G (b) A PSG graph **G'**

| ID | Edge | SE-triple |
|---|---|---|
| $e_0$ | $(A_0, C_0, P_0)$ | (1,1,1) |
| $e_1$ | $(A_0, C_1, P_0)$ | (3,2,2) |
| $e_2$ | $(A_0, B_0, P_2)$ | (2,1,1) |
| $e_3$ | $(A_0, B_0, P_3)$ | (1,1,1) |
| $e_4$ | $(C_0, B_0, P_1)$ | (2,2,1) |
| $e_5$ | $(C_1, B_0, P_1)$ | (1,1,1) |

(c) Edges and SE-triples

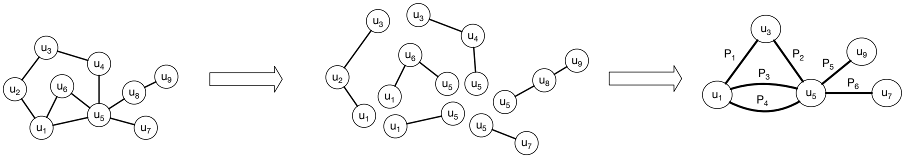
We develop PSGBuilder, a PSG construction algorithm that

- PSGBuilder constructs a PSG for any given graph in linear time, and
- guarantees reduced running time when using more processors.

Key ideas behind PSGBuilder. (1) Vertex-level parallelism; (2) Efficient neighborhood access.

# Cardinality Estimation

(1) Decompose $Q$ into a new query $Q'$ with a simpler structure, such that each edge in $Q'$ represents a path query in $Q$.



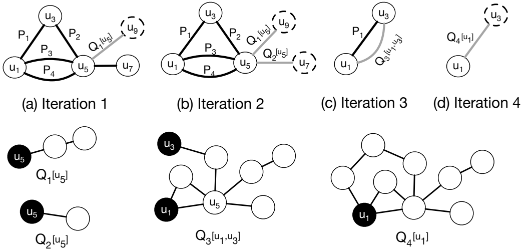

(a) Query Q (b) Short-path queries (c) Query Q'

To leverage the precomputed PSG statistics and improve accuracy, PathCE ensures that

- the statistics of each path query in $Q'$, e.g., $P_1$, are precomputed and stored in the PSG;
- the number of vertices in $Q'$ is minimized to reduce the number of estimation iterations.

# Cardinality Estimation (cont'd)

**(2)** Estimate $|Q(G)|$ using $Q'$ and the precomputed PSG – fewer iterations, higher accuracy.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3 (d) Iteration 4

(e) Subqueries processed in iterations; designated endpoints are marked in black.

- Using maximum-degree statistics[1], PathCE ensures that the estimation for each subquery of $Q$ is pessimistic.
- **Proposition.** Let $c$ be the estimate produced by PathCE. Then $|Q(G)| \leqslant c$.

[1] Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. SIGMOD 2019.

# Evaluation

## Datasets and Queries

| | \|V\| | \|E\| | Queries |
|---|---|---|---|
| LDBC | 3.73M | 21.4M | LSQB + GLogs |
| IMDB | 52.6M | 119M | JOB |
| AIDS | 254K | 548K | G-CARE Queries |

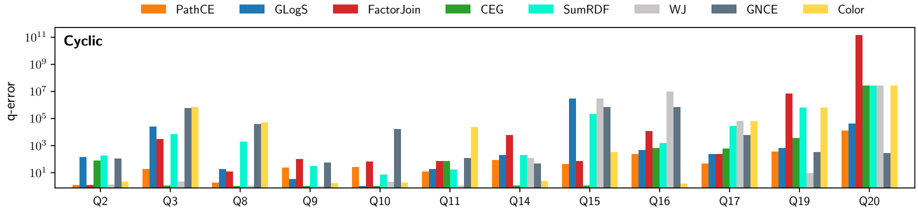
## Baselines

- Summary-based: GLogS, CEG, FactorJoin, SumRDF, Color
- Sampling-based: WanderJoin (WJ)
- ML-based: GNCE

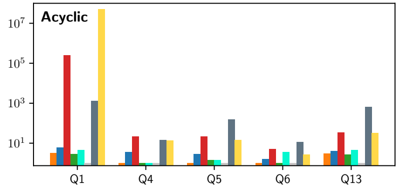
- Metrics: estimation accuracy, estimation latency, and summary-construction efficiency.
- PSG construction efficiency also evaluated on LDBC with SF = 0.1, 0.3, 1, 3, 10.

# EXP-1: Estimation Accuracy

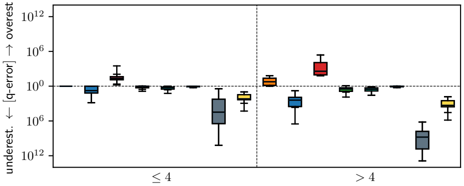
- For cyclic queries, PathCE yields the most accurate estimates on both real-world and synthetic datasets.
- For acyclic queries, PathCE delivers accuracy comparable to CEG and WJ.




(a). Accuracy (varying query, LDBC cyclic)



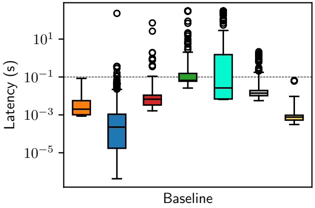

(b). Accuracy (varying query, LDBC acyclic)



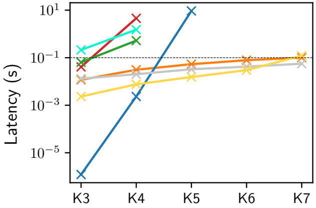

(c). Accuracy (varying query size, IMDB)

# EXP-2: Estimation Latency

- PathCE delivers fast estimation with consistently low latency variance.
- Rationale: PathCE has a smaller search space with fewer iterations.



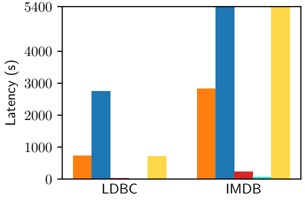

(a). Latency on all datasets



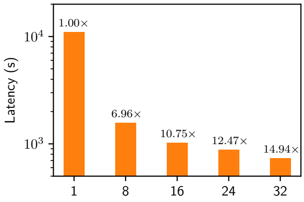

(b). Latency using K3–K7 (LDBC)

# EXP-3: Summary Construction Efficiency

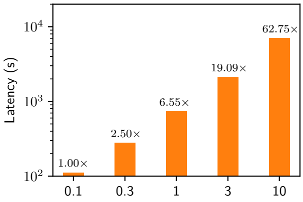
- PathCE builds PSG efficiently, and is the fastest estimator among those that consider path-query statistics.
- PSG construction scales with both thread count and data graph size.



(a). Summary construction time



(b). Scalability (varying thread count)



(c). Scalability (varying scale factor)

# PathCE Recap

- PathCE is a path-centric framework for cardinality estimation in subgraph matching.
- It introduces PSG, a novel data structure that encodes short-path query statistics.
- With path-query statistics, PathCE delivers higher accuracy with fewer iterations.
- PathCE also includes a parallel, scalable PSG builder for large data graphs.

# Future Work

- Q1. How can we effectively handle predicates?
- Q2. How can we efficiently maintain a PSG under data-graph updates?
- Q3. Can a PathCE variant (or similar technique) be applied in relational DBMSs?

Thanks!

Q & A.