DuckPGQ:
Efficient Property Graph Queries in an analytical RDBMS

Daniël ten Wolde, Tavneet Singh, Gabor Szarnyas, Peter Boncz
CWI Database Architectures group

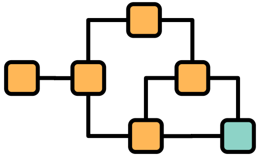
CIDR 2023
Amsterdam

CWI
Centrum Wiskunde & Informatica

Outline

1. the why and what of SQL/PGQ
2. competent graph database systems architecture
3. graph query processing in DuckDB

Graph data management



connected data

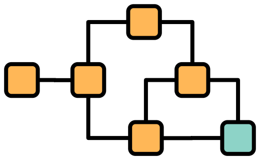
src	dst
1	2
2	3
2	5
…	…

src	dst
4	7
5	7

tables often represent graphs

Centrum Wiskunde & Informatica

Graph data management



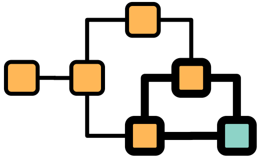
connected data

src	dst
1	2
2	3
2	5
…	…

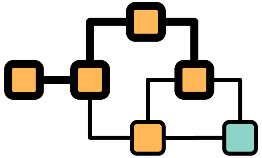
src	dst
4	7
5	7

tables often represent graphs

important functionalities:



pattern matching



path-finding

```
SELECT count(*)
FROM person
WHERE name LIKE 'E%'
```

relational operators

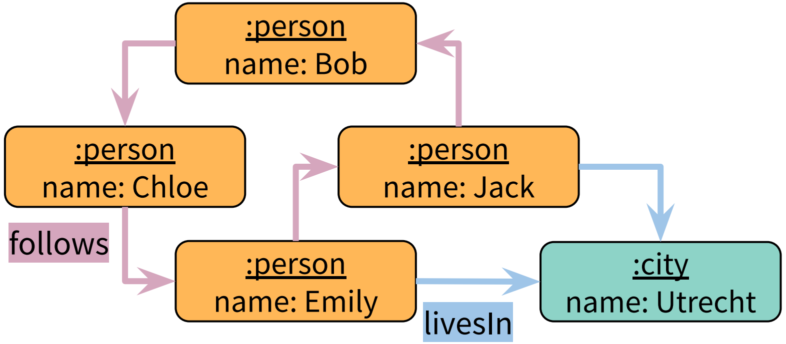
CWI
Centrum Wiskunde & Informatica

Storing graphs in SQL

```sql
CREATE TABLE city (
  id bigint PRIMARY KEY,
  name varchar
);

CREATE TABLE person (
  id bigint PRIMARY KEY,
  name varchar,
  livesIn bigint,
  CONSTRAINT c FOREIGN KEY (livesIn) REFERENCES city (id)
);

CREATE TABLE follows (
  p1id bigint,
  p2id bigint,
  CONSTRAINT p1 FOREIGN KEY (p1id) REFERENCES person (id),
  CONSTRAINT p2 FOREIGN KEY (p2id) REFERENCES person (id)
);
```



“count the number of people Bob (in)directly follows who live in the city Utrecht”

"count the number of people Bob (in)directly follows who live in the city Utrecht"

SQL:1999 query

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode, array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN city      ON city.id = p2.livesIn AND city.name = 'Utrecht'
```

"count the number of people Bob (in)directly follows who live in the city Utrecht"

SQL:1999 query

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode, array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN city      ON city.id = p2.livesIn AND city.name = 'Utrecht'
```

CWI
Centrum Wiskunde & Informatica

"count the number of people Bob (in)directly follows who live in the city Utrecht"

SQL:1999 query

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode, array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN city      ON city.id = p2.livesIn AND city.name = 'Utrecht'
```

Graph query languages

NebulaGraph	Amazon Neptune	JanusGraph
nGQL	SPARQL	Gremlin
Oracle Labs PGX	TigerGraph	neo4j
PGQL	GSQL	Cypher

Centrum Wiskunde & Informatica

The (sorry) State of Graph Database Systems

Peter Boncz
CWI

EDBT 2022 Keynote

comparing graph with relational database systems..

\+ *provide pointers to related literature*

CWI
Centrum Wiskunde & Informatica

The Sorry State of Graph Database Systems

“The six blunders of graph database systems” (see keynote)

- time may be running out for native property graph database systems
  - Some success in certain use cases: Data Integration, Data cleaning & Enrichment, Fraud Detection, Recommendation, Historical Analysis, Root-Cause Analysis,...
  - still a niche solution and maturity+usability problems remain
- especially if SQL/PGQ becomes a (moderate) success
  - Relational systems will be able to handle their use cases
  - Only Data Integration, Data cleaning & Enrichment would be left (RDF/SPARQL territory)

SQL/PGQ (Property Graph Queries)

Centrum Wiskunde & Informatica

SQL/PGQ

- Extension in the upcoming SQL:2023 standard, 2b released in June
- Property Graphs as views over existing tables
  - edge,vertex=table, property (value) =column (value), label=table-name
- Read-only operations for property graph queries
  - Path-finding + Pattern matching in Cypher-like syntax, producing a “Graph-Table” in FROM

CWI
Centrum Wiskunde & Informatica

SQL/PGQ

- Extension in the upcoming SQL:2023 standard, 2b released in June
- Property Graphs as views over existing tables
  - edge,vertex=table, property (value) =column (value), label=table-name
- Read-only operations for property graph queries
  - Path-finding + Pattern matching in Cypher-like syntax, producing a “Graph-Table” in FROM

TigerGraph
GSQL

Oracle Labs PGX
PGQL

neo4j
Cypher

LDBC
G-CORE

ISO
SQL:2023

Tabular schema

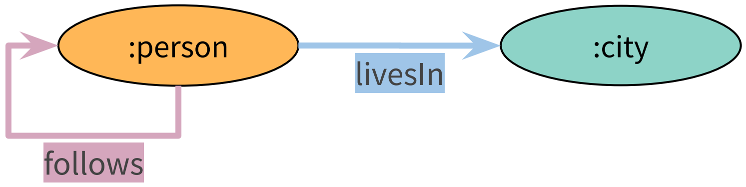
```
CREATE TABLE city (
  id bigint PRIMARY KEY,
  name varchar
);

CREATE TABLE person (
  id bigint PRIMARY KEY,
  name varchar,
  livesIn bigint,
  CONSTRAINT c FOREIGN KEY ...
);

CREATE TABLE follows (
  p1id bigint,
  p2id bigint,
  CONSTRAINT p1 FOREIGN KEY ...
  CONSTRAINT p2 FOREIGN KEY ...
);
```

SQL/PGQ graph tables

```
CREATE PROPERTY GRAPH socialNetwork
  VERTEX TABLES (
    city,
    person
  )
  EDGE TABLES (
    livesIn SOURCE person DESTINATION city,
    follows SOURCE person DESTINATION person
  );
```



SQL/PGQ query

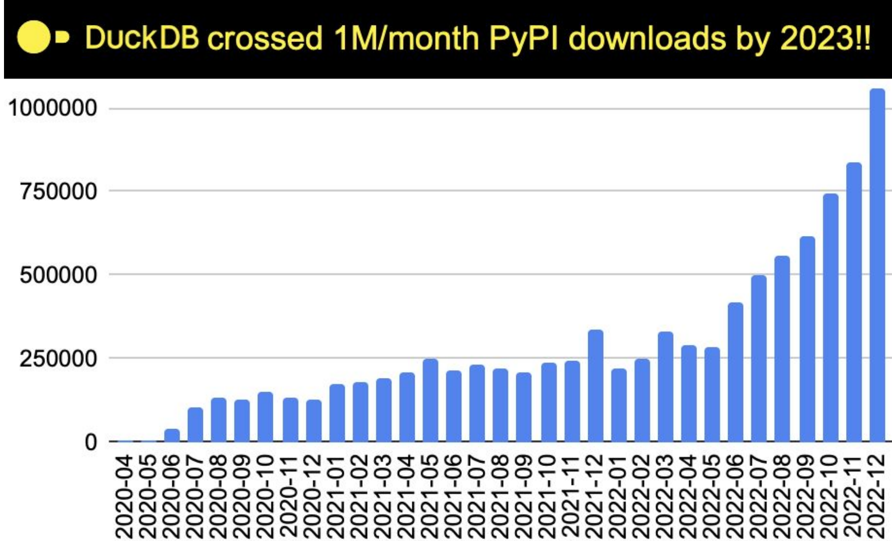
“count the number of people Bob (in)directly follows who live in the city Utrecht”

```
SELECT count(gt.id)
FROM
  GRAPH_TABLE (socialNetwork,
    MATCH (p1:person WHERE p1.name='Bob')-[:follows]->*(p2:person)
            -[:livesIn]->(c:city WHERE c.name='Utrecht')
    COLUMNS (p2.id)
  ) gt
```

DuckPGQ module for DuckDB

Centrum Wiskunde & Informatica

DuckDB



- open-source in-process SQL OLAP DBMS
- Created by Mark Raasveldt

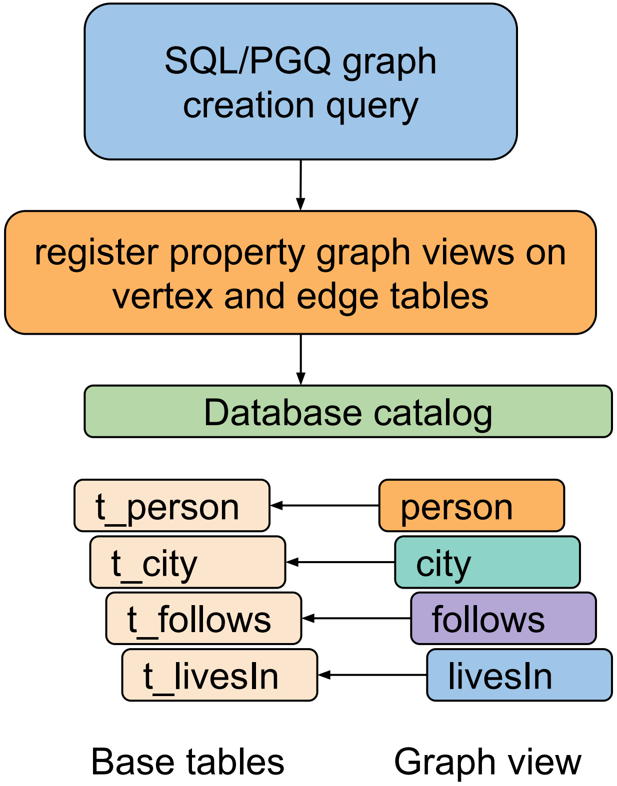
  & Hannes Mühleisen (keynote Wednesday)
- very popular in data science notebooks, but suitable for many analytics applications
- "Modern": Vectorized execution engine, Morsel-driven parallelism, ..
- Allows extension modules:
  - scalar user-defined functions (UDF), parser extensions
  - data sources (scans), table-returning functions

DuckDB Labs

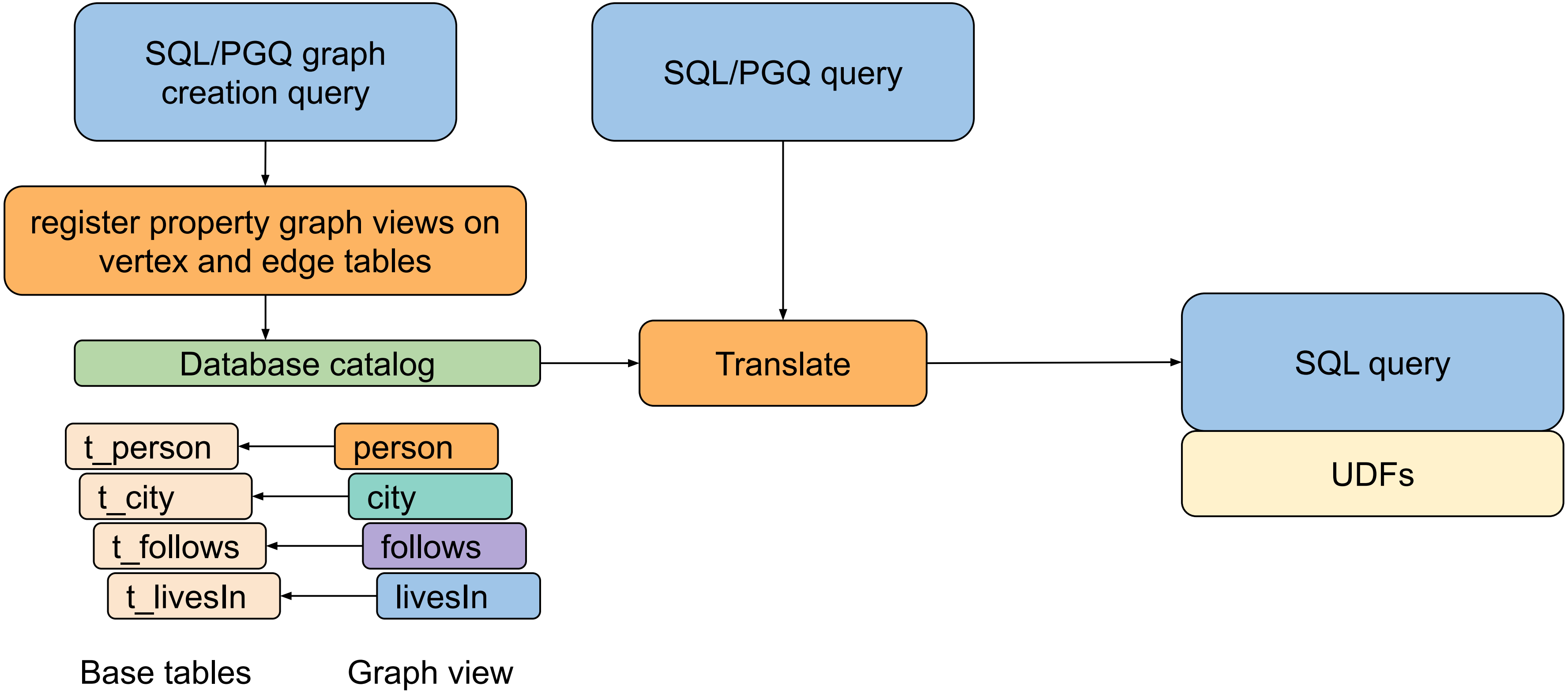
MotherDuck

Centrum Wiskunde & Informatica

Current DuckPGQ pipeline

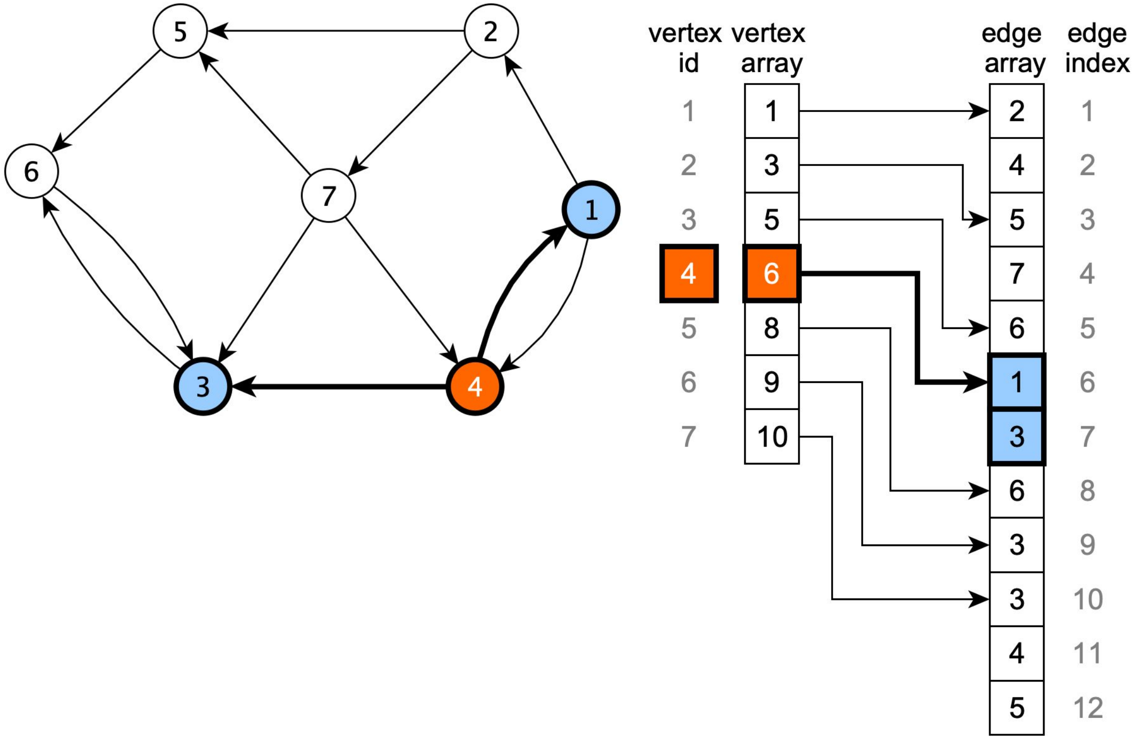


Current DuckPGQ pipeline



Path finding: Compressed Sparse Row (CSR)

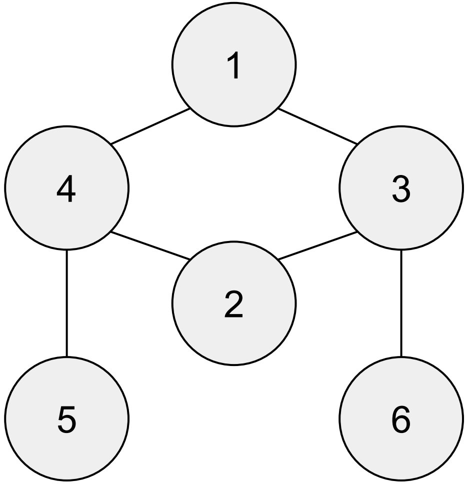
- **On-the-fly** creation (no update handling needed)
- Using **scalar UDFs** (parallel, very fast)
- Index in the **vertex array** corresponds to the ROWID of the vertex
- Vertex array contains offsets for the **edge arrays**



CWI
Centrum Wiskunde & Informatica

Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)



VLDB’14

The More the Merrier: Efficient Multi-Source Graph Traversal

Manuel Then* then@in.tum.de
Moritz Kaufmann* kaufmanm@in.tum.de
Fernando Chirigati† fchirigati@nyu.edu
Tuan-Anh Hoang-Vu† tuananh@nyu.edu
Kien Pham† kien.pham@nyu.edu
Alfons Kemper* kemper@in.tum.de
Thomas Neumann* neumann@in.tum.de
Huy T. Vo† huy.vo@nyu.edu
* Technische Universität München
† New York University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Many graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose *Multi-Source BFS* (MS-BFS), an algorithm that is designed to

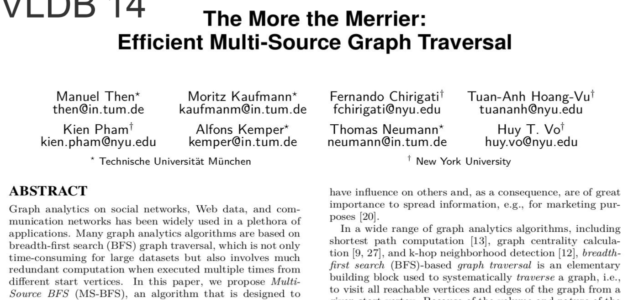
have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [20].

In a wide range of graph analytics algorithms, including shortest path computation [13], graph centrality calculation [9, 27], and k-hop neighborhood detection [12], *breadth-first search* (BFS)-based *graph traversal* is an elementary building block used to systematically *traverse* a graph, i.e., to visit all reachable vertices and edges of the graph from a

Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

VLDB'14



The More the Merrier:
Efficient Multi-Source Graph Traversal

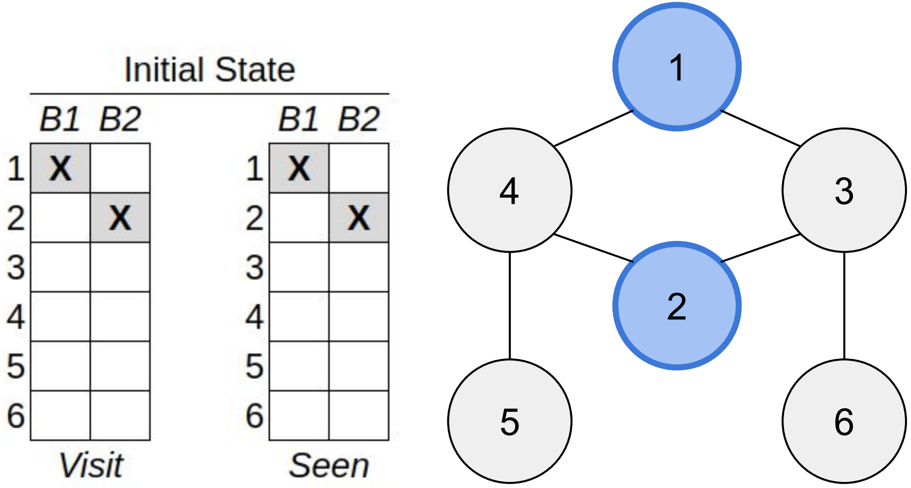
Manuel Then* then@in.tum.de
Moritz Kaufmann* kaufmanm@in.tum.de
Fernando Chirigati† fchirigati@nyu.edu
Tuan-Anh Hoang-Vu† tuananh@nyu.edu
Kien Pham† kien.pham@nyu.edu
Alfons Kemper* kemper@in.tum.de
Thomas Neumann* neumann@in.tum.de
Huy T. Vo† huy.vo@nyu.edu
* Technische Universität München
† New York University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Many graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose *Multi-Source BFS* (MS-BFS), an algorithm that is designed to

have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [20].

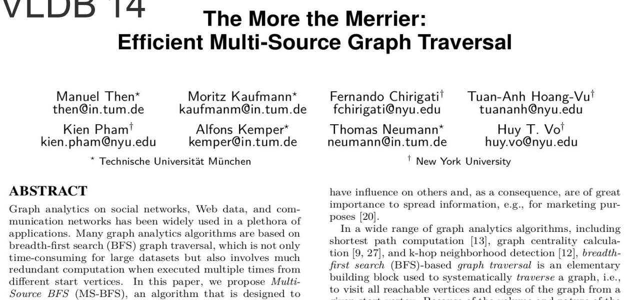
In a wide range of graph analytics algorithms, including shortest path computation [13], graph centrality calculation [9, 27], and k-hop neighborhood detection [12], *breadth-first search* (BFS)-based *graph traversal* is an elementary building block used to systematically *traverse* a graph, i.e., to visit all reachable vertices and edges of the graph from a



Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

VLDB'14



The More the Merrier: Efficient Multi-Source Graph Traversal

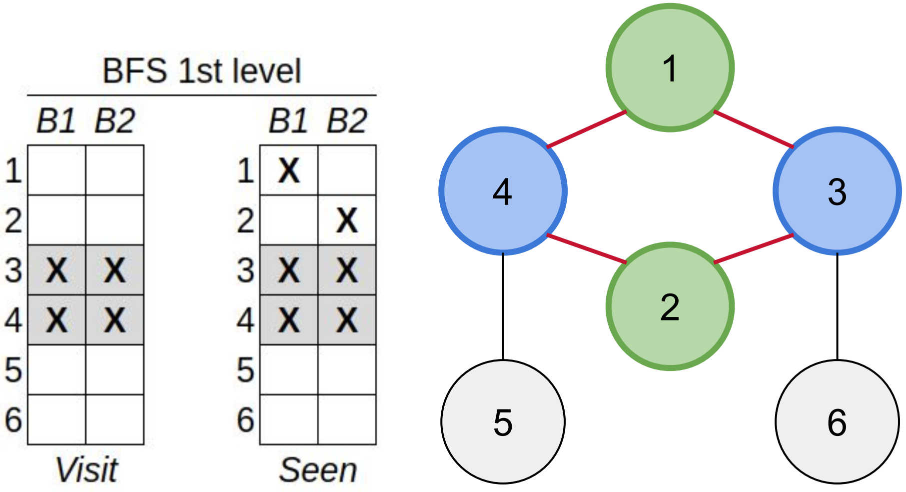
Manuel Then* then@in.tum.de
Moritz Kaufmann* kaufmanm@in.tum.de
Fernando Chirigati† fchirigati@nyu.edu
Tuan-Anh Hoang-Vu† tuananh@nyu.edu
Kien Pham† kien.pham@nyu.edu
Alfons Kemper* kemper@in.tum.de
Thomas Neumann* neumann@in.tum.de
Huy T. Vo† huy.vo@nyu.edu
* Technische Universität München
† New York University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Many graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose *Multi-Source BFS* (MS-BFS), an algorithm that is designed to

have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [20].

In a wide range of graph analytics algorithms, including shortest path computation [13], graph centrality calculation [9, 27], and k-hop neighborhood detection [12], *breadth-first search* (BFS)-based *graph traversal* is an elementary building block used to systematically *traverse* a graph, i.e., to visit all reachable vertices and edges of the graph from a



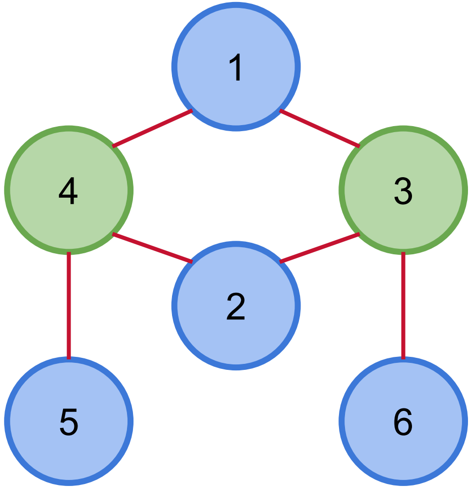
Centrum Wiskunde & Informatica

Multi-Source Breadth-First Search (MS-BFS)

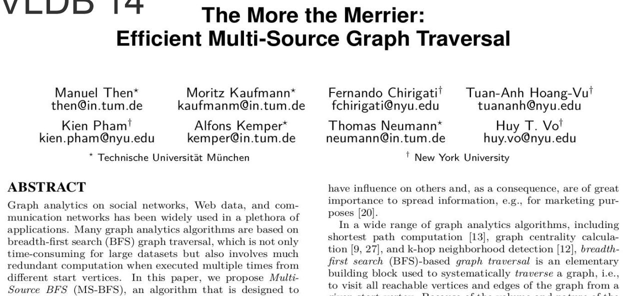
- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

BFS 2nd level

	Visit B1	Visit B2	Seen B1	Seen B2
1		X	X	X
2	X		X	X
3			X	X
4			X	X
5	X	X	X	X
6	X	X	X	X



VLDB’14



The More the Merrier:
Efficient Multi-Source Graph Traversal

Manuel Then* then@in.tum.de
Moritz Kaufmann* kaufmanm@in.tum.de
Fernando Chirigati† fchirigati@nyu.edu
Tuan-Anh Hoang-Vu† tuananh@nyu.edu
Kien Pham† kien.pham@nyu.edu
Alfons Kemper* kemper@in.tum.de
Thomas Neumann* neumann@in.tum.de
Huy T. Vo† huy.vo@nyu.edu
* Technische Universität München
† New York University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Many graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose *Multi-Source BFS* (MS-BFS), an algorithm that is designed to

have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [20].

In a wide range of graph analytics algorithms, including shortest path computation [13], graph centrality calculation [9, 27], and k-hop neighborhood detection [12], *breadth-first search* (BFS)-based *graph traversal* is an elementary building block used to systematically *traverse* a graph, i.e., to visit all reachable vertices and edges of the graph from a

Centrum Wiskunde & Informatica

Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

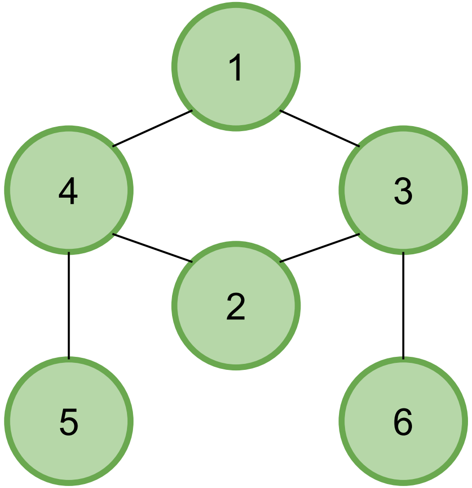
BFS 2nd level

	B1	B2
1		X
2	X	
3		
4		
5	X	X
6	X	X

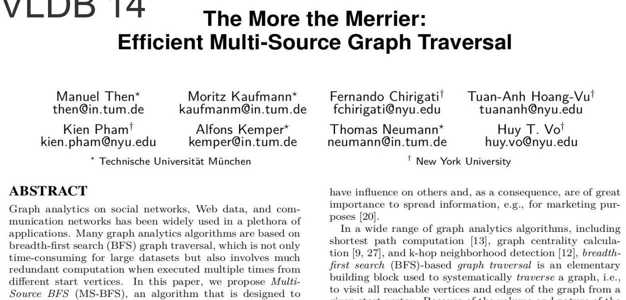
Visit

	B1	B2
1	X	X
2	X	X
3	X	X
4	X	X
5	X	X
6	X	X

Seen



VLDB’14



The More the Merrier:
Efficient Multi-Source Graph Traversal

Manuel Then* then@in.tum.de
Moritz Kaufmann* kaufmanm@in.tum.de
Fernando Chirigati† fchirigati@nyu.edu
Tuan-Anh Hoang-Vu† tuananh@nyu.edu
Kien Pham† kien.pham@nyu.edu
Alfons Kemper* kemper@in.tum.de
Thomas Neumann* neumann@in.tum.de
Huy T. Vo† huy.vo@nyu.edu
* Technische Universität München
† New York University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Many graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose *Multi-Source BFS* (MS-BFS), an algorithm that is designed to

have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [20].

In a wide range of graph analytics algorithms, including shortest path computation [13], graph centrality calculation [9, 27], and k-hop neighborhood detection [12], *breadth-first search* (BFS)-based *graph traversal* is an elementary building block used to systematically *traverse* a graph, i.e., to visit all reachable vertices and edges of the graph from a

Last Slide

Conclusion

- Why should you read our DuckPGQ paper?
  - **Learn SQL/PGQ** in less than 1 page (or become ldbcouncil.org member & read 200+ pages of spec)
  - Read our **12 golden rules** of competent graph systems design (just 1 page of reading)
  - See how DuckDB extensibility can be leveraged for a modular **implementation of SQL/PGQ**

    (..and we also present some benchmark results..)
- DuckPGQ availability? Not yet.. WIP & ETA in 2023
- Many avenues for future data systems research :
  - Factorized query execution, Vectorized WCOJs & their query optimization
  - Path-finding and query optimization, better path-finding parallelism