Incremental Graph Queries for Cypher

Gábor Szárnyas, József Marton

Budapest University of Technology and Economics
McGill University, Montréal
Live railway model
Live railway model
Live railway model
Live railway model
Live railway model

Proximity detection
Live railway model

Proximity detection
Live railway model

Proximity detection
Live railway model

Proximity detection

Trailing the switch
Live railway model
Live railway model
Live railway model
Live railway model
Proximity detection

Proximity detection

≤ 2 segments
Proximity detection

Proximity detection

≤ 2 segments

seg1

seg2

NEXT: 1..2
Proximity detection

MATCH

(t1:Train)-[:ON]->(seg1:Segment)
-[:NEXT*1..2]->(seg2:Segment)
<-[:ON]-(t2:Train)

RETURN t1, t2, seg1, seg2
MATCH
(t1:Train)-[:ON]->(seg1:Segment)
-[:NEXT*1..2]->(seg2:Segment)
<-[:ON]-(t2:Train)
RETURN t1, t2, seg1, seg2
Trailing the switch
Trailing the switch
MATCH (t:Train)-[:ON]-(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment)<-[[:STRAIGHT]]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
Trailing the switch

MATCH (t:Train)-[:ON]-(seg:Segment)-[:STRAIGHT]->(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw

Evaluate continuously
Incremental queries
Incremental queries

- Register a set of **standing queries**
- **Continuously evaluate** queries on changes
Incremental queries

- Register a set of **standing queries**
- **Continuously evaluate** queries on changes

- The **Rete algorithm** (1974)
  - Originally for rule-based expert systems
  - Indexes the graph and caches interim query results
Trailing the switch

\[ \pi_{\text{t.number, sw}} \]

\[ \sigma_{\text{sw.position = 'diverging'}} \]

\[ \bowtie \]

ON

STRAIGHT
Trailing the switch

\( \pi_{t.number, sw} \)

\( \sigma_{sw.position = 'diverging'} \)

1. a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow div \rightarrow TOP \rightarrow f
2. a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow div \rightarrow TOP \rightarrow f
Trailing the switch

$\pi_{t.\text{number}, \text{sw}}$

$\sigma_{\text{sw.\text{position} = 'diverging'}}$

ON

STRAIGHT

1

STRAIGHT
div

NEXT

NEXT

NEXT

NEXT

STRAIGHT

TOP
div

g

f
Trailing the switch

$\pi_{t.number, sw}$

$\sigma_{sw.position = 'diverging'}$

ON

STRAIGHT

1

NEXT

a

b

c

NEXT
d

e

NEXT

f

2

NEXT

div

g

NEXT

STRAIGHT

TOP
Trailing the switch

\[ \pi t.\text{number}, \ sw \]

\[ \sigma_{\text{sw.position} = \text{'diverging'}} \]

ON

STRAIGHT

\[ \text{div} \]

\[ \text{div} \]

TOP

NEXT

DIVERGING

STRAIGHT
Trailing the switch

\[
\pi t.\text{number}, \text{sw}
\]

\[
\sigma_{\text{sw.position} = \text{'diverging'}}
\]

1. STRAIGHT
2. STRAIGHT
Trailing the switch

\[
\pi_{t.\text{number}, \text{sw}}
\]

\[
\sigma_{\text{sw.position} = \text{'diverging'}}
\]

\[
\otimes
\]

ON

1 ON a 2 ON e

STRAIGHT

1

NEXT a

NEXT b

NEXT c

NEXT d

NEXT e

STRAIGHT div TOP f

DIVERGING g
Trailing the switch

\[ \pi_{t.number, sw} \]

\[ \sigma_{sw.position = 'diverging'} \]

\[ \bowtie \]

ON

1 → a → 2 → e

STRAIGHT

e → div

1

ON

a → b → c → d → e

STRAIGHT

div

2

ON

e → f

ON

g → NEXT

STRAIGHT

TOP

DIVERGING
Trailing the switch

\[ \pi_{t.number, sw} \]

\[ \sigma_{sw.position = 'diverging'} \]

\[ \bowtie \]

Next

\[ g \]

\[ f \]
Trailing the switch

\[ \pi_{t.number, sw} \]

\[ \sigma_{sw.position = 'diverging'} \]
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'} } \]
Trailing the switch

\( \pi_{t.number, \text{sw}} \)

\( \sigma_{\text{sw.position} = \text{'diverging'}} \)
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'}{}} \]

1. ON → a → 2 → ON → e

ON → 1 → STRAIGHT → e

2. ON → e → STRAIGHT → div

ON → 2 → STRAIGHT → TOP → f

NEXT → a → NEXT → b → NEXT → c → NEXT → d → NEXT → e → STRAIGHT → TOP → f

NEXT → g → DIVERTING → e

NEXT → b → NEXT → c → NEXT → d → NEXT → e → STRAIGHT → TOP → f
Trailing the switch

\[ \pi_t.number, \text{sw} \]

\[ \sigma_{\text{sw.position} = \text{'diverging'}} \]

\[ \text{ON} \quad 1 \quad \text{ON} \quad a \quad 2 \quad \text{ON} \quad e \]

\[ \text{STRAIGHT} \quad e \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{ON} \quad 2 \quad \text{ON} \quad e \quad \text{ON} \quad \text{div} \]

\[ \text{STRAIGHT} \quad e \quad \text{STRAIGHT} \quad \text{div} \]

\[ a \quad \text{NEXT} \quad b \quad \text{NEXT} \quad c \quad \text{NEXT} \quad d \quad \text{NEXT} \quad e \quad \text{NEXT} \quad f \]

\[ g \quad \text{DIVERGING} \quad \text{TOP} \quad \text{div} \]
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = \text{\textquoteleft}diverging\textquoteright} \]

\[ \bowtie \]

ON

\[ 1 \rightarrow a \rightarrow 2 \rightarrow e \]

STRAIGHT

\[ e \rightarrow \text{div} \]

NEXT

\[ a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \]

STRAIGHT

\[ e \rightarrow \text{div} \rightarrow f \]

TOP

DIVERGING

\[ g \]

NEXT
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'} \}

ON

STRAIGHT

NEXT

DIVERGING

TOP

DIVING

ON

STRAIGHT
Trailing the switch

$\pi_{t.number, \text{sw}}$

$\sigma_{\text{sw.position} = \text{'diverging'}}$

ON

STRAIGHT

NEXT

ON

STRAIGHT

ON
Trailing the switch

\[ \pi_{t.number, \, sw} \]

\[ \sigma_{sw.position = 'diverging'} \]

\[ a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow div \]

\[ 1 \rightarrow ON \rightarrow a \rightarrow 2 \rightarrow ON \rightarrow e \rightarrow STRAIGHT \rightarrow div \]

\[ g \rightarrow DIVERGING \rightarrow \]
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'} \]
Trailing the switch

\[ \pi_{t \text{.number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = '\text{diverging'} } \]

---

1. ON
2. ON
3. ON
4. STRAIGHT
5. STRAIGHT
6. STRAIGHT

a → b → c → d → e → div

---

1. NO
2. STRAIGHT
3. div
4. div
5. div

a → b → c → d → e → div → f → g

---

\[ \bowtie \]
Trailing the switch

\[ \pi_{t.\text{number, sw}} \]

\[ \sigma_{\text{sw.position = 'diverging'}} \]

NEXT

ON

STRAIGHT

ON

STRAIGHT

ON

STRAIGHT

ON

STRAIGHT

TARGET

ON

STRAIGHT

ON
Trailing the switch

\[ \pi_{t.number, \text{sw}} \]

\[ \sigma_{\text{sw.position} = \text{'diverging'}} \]

\[ \bowtie \]

ON

1 → a → 2 → d

STRAIGHT

e → div

ON

1

a → b → c → d → e → f

div → TOP → STRAIGHT

g → DIVERGING
Batch vs. incremental queries

- **Batch queries** (pull / request-driven):
  1. Client selects a query
  2. Results are calculated

- Query results obtained on demand
Batch vs. incremental queries

**Batch queries** (pull / request-driven):

1. Client selects a query
2. Results are calculated

**Query results obtained on demand**

**Incremental queries** (push / event-driven):

1. Client registers queries
2. Graph is changed
3. Results are maintained
4. Goto 2

**Query results are always available**
Incremental query engines

- **CLIPS**  
  C structures  
  NASA

- **Drools**  
  POJO  
  Red Hat

- **VIATRA**  
  EMF  
  BME / IncQuery Labs.
Incremental query engines

- **CLIPS**
  - C structures
  - NASA

- **Drools**
  - POJO
  - Red Hat

- **VIATRA**
  - EMF
  - BME / IncQuery Labs.

- **INSTANS**
  - RDF
  - Aalto University

- **i3QL**
  - POJO
  - TU Darmstadt

- **IncQuery-D**
  - RDF
  - BME
Incremental query engines

- **CLIPS**
  - C structures
  - NASA

- **Drools**
  - POJO
  - Red Hat

- **VIATRA**
  - EMF
  - BME / IncQuery Labs.

- **INSTANS**
  - RDF
  - Aalto University

- **i3QL**
  - POJO
  - TU Darmstadt

- **IncQuery-D**
  - RDF
  - BME

- **No implementations for property graphs yet**
An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
- An incremental, in-memory graph query engine

```
client
```

```
ingraph
```

register queries
An incremental, in-memory graph query engine
- An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]- (sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment)
  <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
Relational algebra model

Rete network model

Rete network

Transformer and optimizer

VIATRA
FORMALIZATION OF OPENCYPHER
Relational algebra

- Standard relational algebra
  - $\pi, \sigma$
  - $\cup, \cap, \setminus$
  - $\times, \bowtie$

- Common extensions
  - $\delta$ – duplicate elimination
  - $\gamma$ – grouping
  - $\tau$ – sorting
Graph-specific operators

**GetVertices:** returns a graph relation containing all vertices of the underlying graph G

\[ \bigcirc (v : \text{types}) \]

**Expand:** return the neighbors of a given node

\[ \uparrow (w : \text{types}) \quad [E : \text{labels} \ast \min \ldots \max] (p) \]

Jürgen Hölsch, Michael Grossniklaus:
*An Algebra and Equivalences to Transform Graph Patterns in Neo4j*, GraphQ 2016, EDBT,
Additional extensions

GetEdges: returns a graph relation containing all edges of the underlying graph G

\[(\text{trg} : \text{trgTypes}) \uparrow (\text{src} : \text{srcTypes}) [\text{e} : \text{labels}]\]

Gábor Szárnyas, József Marton:
openCypher specification, Technical report
While pattern matching, Neo4j makes sure to not include matches where the same graph relationship is found multiple times in a single pattern.

MATCH
  (user:User)-[r1:FRIEND]-()
  -[r2:FRIEND]-(foaf)
WHERE user.name = 'Adam'
RETURN foaf.name AS foafName

All-different operator

\[ \forall E_1, E_2, E_3, \ldots (r) = \sigma \bigwedge_{e_1, e_2 \in \bigcup_i E_i \land e_1 \neq e_2} r.e_1 \neq r.e_2 (r) \]
EXAMPLE QUERIES
MATCH  
  (user:User)-[r1:FRIEND]-()  
  -[r2:FRIEND]-(foaf)  
WHERE user.name = 'Adam'  
RETURN foaf.name AS foafName
MATCH (n1)-[rel:KNOWS]->(n2)
RETURN n1, n2
Filter out based on node prop name

MATCH ()-[rel:X]-(a)
WHERE a.name = 'Andres'
RETURN a
Use multiple MATCH clauses to do a Cartesian product

MATCH (n), (m)
RETURN
  n.value AS n, m.value AS m
MATCH (a)-[r]-(b)
WHERE r.name = 'r1'
OPTIONAL MATCH (b)-[r2]-(c)
WHERE r <> r2
RETURN a, b, c
MATCH (sw:Switch)
WHERE NOT
((sw)-[:monitoredBy]->(:Sensor))
RETURN sw
MATCH (cypher:QueryLanguage)-[:QUERIES]->(graphs)
MATCH (cypher)<-[:USES]-(u:User)
WHERE u.name IN
    ['Oracle', 'Apache Spark', 'Tableau', 'Structr']
MATCH (openCypher)-[:MAKESAVAILABLE]->(cypher)
RETURN cypher.attributes
INCREMENTAL GRAPH QUERIES WITH OPENCYPHER
openCypher constructs

- Standard constructs
  - pattern matching
  - filtering
  - lists, maps
  - data manipulation
  - variable length paths
openCypher constructs

- **Standard constructs**
  - pattern matching
  - filtering
  - lists, maps
  - data manipulation
  - variable length paths

- **Legacy constructs**
  - indexing, constraints
  - regular expressions
  - some list functions, including reduce
  - most predicate functions
  - shortest path functions
  - **CASE** expressions
  - **id()**
openCypher constructs

- **Standard constructs**
  - pattern matching
  - filtering
  - lists, maps
  - data manipulation
  - variable length paths

- **Legacy constructs**
  - indexing, constraints
  - regular expressions
  - some list functions, including reduce
  - most predicate functions
  - shortest path functions
  - CASE expressions
  - id()

Difficult to handle incrementally
Challenges for incremental openCypher

- Lists
  - ['a', 1, 2, true]
  - ['a', [1, [2]], true]
  - UNWIND

- Efficient aggregation
  - min(), max()
  - collect()

- Bag semantics, ORDER BY, SKIP and LIMIT
  - Idea: collect(x ORDER BY x.name)
Incremental queries – use cases

Standing queries on large & quickly changing graph

- Runtime monitoring (train example)
- Model validation: The Train Benchmark
- Static analysis of JavaScript source code
- Fraud detection
- IT infrastructure monitoring
Future work

Path operator


Unwind operator: $\omega$

Open-Source Projects

Incremental Graph Engine:
https://github.com/ftsrg/ingraph

Train Benchmark:
https://github.com/ftsrg/trainbenchmark

BME-MODES3:
https://github.com/ftsrg/bme-modes3

Available under EPL v1.0.