MillenniumDB Path Query Challenge

AMW 2023
Benjamín Farías, Carlos Rojas, Domagoj Vrgoč
Regular path queries – more generally

A generic RPQ

Idea:
- find pairs of nodes
- connected by a path
- whose edge labels are a word matching regex
Regular path queries – more generally

Idea:
- find pairs of nodes
- connected by a path
- whose edge labels are a word matching regex

\(?start\) regex \(?end\) can be constants
What does an RPQ return?

**Result:**

- $?reachable$
- $end$

**All nodes:**
- Reachable from **start** in our graph
- Via a **path**
- Whose edge label matches **a*b**

**Diagram:**

- **start** connected to $n_1$: $a$
- $n_1$ connected to $n_2$: $a$
- $n_2$ connected to $n_3$: $a$
- $n_3$ connected to **end**: $b$
- **start** connected to $n_3$: $a^*b$
- $n_3$ connected to $?$reachable: $b$

**RPQ:**

- **start** connected to $?$reachable: $a^*b$
- $?$reachable connected to **end**: $?$
What does an RPQ return?

All nodes:
- Reachable from \textit{start} in our graph
- Via a \textit{path}
- Whose edge label matches \textit{a*b}

\begin{itemize}
\item \texttt{start}
\item \texttt{n_1}
\item \texttt{n_2}
\item \texttt{n_3}
\item \texttt{end}
\end{itemize}

What if I also want the path?
What does an RPQ return?

I also want the path:

- **Path #1**: start → n1 → n3 → end
- **Path #2**: start → n1 → n2 → n3 → end
- **Path #3**: start → n2 → n3 → end

Result:

| ?reachable | end |
What does an RPQ return?

I also want the path:
- **Path #1**: start→n1→n3→end
- **Path #2**: start→n1→n2→n3→end
- **Path #3**: start→n2→n3→end

Result:
- ?reachable
- end

Which one?
What GQL proposes – you tell me

I also want the path:

- **Path #1**: start\(\rightarrow\)n1\(\rightarrow\)n3\(\rightarrow\)end
- **Path #2**: start\(\rightarrow\)n1\(\rightarrow\)n2\(\rightarrow\)n3\(\rightarrow\)end
- **Path #3**: start\(\rightarrow\)n2\(\rightarrow\)n3\(\rightarrow\)end
What GQL proposes – you tell me

\[ ?p = \text{ANY WALK} \ (\text{start})=[a*b]=>(?\text{reachable}) \]

\[ ?p = \text{ANY SHORTEST WALK} \ (\text{start})=[a*b]=>(?\text{reachable}) \]

\[ ?p = \text{ALL SHORTEST WALK} \ (\text{start})=[a*b]=>(?\text{reachable}) \]

Why WALK?
Mathematicians call a path a walk
(AND THEY ARE STRANGE)

- Path #2: start\(\rightarrow\)n1\(\rightarrow\)n2\(\rightarrow\)n3\(\rightarrow\)end
- Path #3: start\(\rightarrow\)n2\(\rightarrow\)n3\(\rightarrow\)end
What GQL proposes – you tell me

For each ?reachable one path (nondeterministic)

\[ ?p = \text{ANY WALK } (\text{start})=[a^*b] \Rightarrow (?\text{reachable}) \]

\[ ?p = \text{ANY SHORTEST WALK } (\text{start})=[a^*b] \Rightarrow (?\text{reachable}) \]

\[ ?p = \text{ALL SHORTEST WALK } (\text{start})=[a^*b] \Rightarrow (?\text{reachable}) \]

Result:

<table>
<thead>
<tr>
<th>?reachable</th>
<th>end</th>
</tr>
</thead>
</table>

I also want the path:

- **Path #1**: start → n1 → n3 → end
- **Path #2**: start → n1 → n2 → n3 → end
- **Path #3**: start → n2 → n3 → end
What GQL proposes – you tell me

For each ?reachable one shortest path (nondeterministic)

\[ ?p = \text{ANY WALK} \ (\text{start}) \ [a \cdot b] = \rightarrow (?\text{reachable}) \]

\[ ?p = \text{ANY SHORTEST WALK} \ (\text{start}) = [a \cdot b] = \rightarrow (?\text{reachable}) \]

\[ ?p = \text{ALL SHORTEST WALK} \ (\text{start}) = [a \cdot b] = \rightarrow (?\text{reachable}) \]

Result:

<table>
<thead>
<tr>
<th>?reachable</th>
<th>end</th>
</tr>
</thead>
</table>

I also want the path:

- **Path #1**: start \(\rightarrow\) n1 \(\rightarrow\) n3 \(\rightarrow\) end
- **Path #2**: start \(\rightarrow\) n1 \(\rightarrow\) n2 \(\rightarrow\) n3 \(\rightarrow\) end
- **Path #3**: start \(\rightarrow\) n2 \(\rightarrow\) n3 \(\rightarrow\) end
What GQL proposes – you tell me

For each \(\text{reachable}\) all shortest paths

\[
?p = \text{ANY WALK} \ (\text{start})=[a^*b]=>(?\text{reachable})
\]

\[
?p = \text{ANY SHORTEST WALK} \ (\text{start})=[a^*b]=>(?\text{reachable})
\]

\[
?p = \text{ALL SHORTEST WALK} \ (\text{start})=[a^*b]=>(?\text{reachable})
\]

I also want the path:

- **Path #1**: start→n1→n3→end
- **Path #2**: start→n1→n2→n3→end
- **Path #3**: start→n2→n3→end
This would be too much

\[ p = \text{ALL WALK (start)=}[a^*]=>(?\text{reachable}) \]
This would be too much

$$p = \text{ALL WALK} \ (\text{start})=\text{[a*]}\Rightarrow(\text{?reachable})$$

For each ?reachable all paths
This would be too much

\[ p = \text{ALL WALK (start)} = [a^*] \Rightarrow (?\text{reachable}) \]

\textbf{A} is reachable from \textbf{start} by:

- start \rightarrow A
- start \rightarrow A \rightarrow B \rightarrow \text{start} \rightarrow A
- start \rightarrow A \rightarrow B \rightarrow \text{start} \rightarrow A \rightarrow B \rightarrow \text{start} \rightarrow A
- ...
This would be too much

\[ ?p = \text{ALL WALK (start)} = [a^*] \rightarrow (?\text{reachable}) \]

**A** is reachable from **start** by:

- start → A
- start → A → B → start → A
- start → A → B → start → A → B → start → A
- ...

Infinite 😞 (NOT GOOD FOR YOUR PC)
But this is OK – ALL SIMPLE

\[ ?p = \text{SIMPLE} \ (\text{start})=\{a^*\}=>(\text{?reachable}) \]

No node is repeated in the path
SIMPLE Path semantics

A is reachable from start by:
- start → A
- start → A → B → start → A

(No infinite looping)
What else?

\[ ?p = \text{TRAIL} \ (\text{start})=\left[a^*\right]=>(\text{?reachable}) \]

No edge is repeated in the path; (We need property graphs)
What else?

\[ ?p = \text{TRAIL} (\text{start})=[a^*]=>(?\text{reachable}) \]

Good trails:
- start → n1
- start → n1 → start
- start → n1 → start → n2

(No infinite looping – limited by the number of edges)
What else?

\[ \exists p = \text{TRAIL} \ (\text{start})=\{a^*\} \Rightarrow (?\text{reachable}) \]

Good trails:
- start\(\rightarrow\)n1
- start\(\rightarrow\)n1\(\rightarrow\)start
- start\(\rightarrow\)n1\(\rightarrow\)start\(\rightarrow\)n2

(No infinite looping – limited by the number of edges)
ALL OPTIONS

\[ ?p = \text{ANY WALK} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]

\[ ?p = \text{ANY SHORTEST WALK} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]

\[ ?p = \text{ALL SHORTEST WALK} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]

\[ ?p = \text{ANY SIMPLE} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]

\[ ?p = \text{ANY SHORTEST SIMPLE} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]

\[ ?p = \text{ALL SHORTEST SIMPLE} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]

\[ ?p = \text{SIMPLE} \ (\text{start})=\text{[regex]}=\rightarrow(\text{end}) \]
\[ ?p = \text{ANY WALK} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]

\[ ?p = \text{ANY SHORTEST WALK} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]

\[ ?p = \text{ALL SHORTEST WALK} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]

\[ ?p = \text{ANY SIMPLE} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]

\[ ?p = \text{ANY SHORTEST SIMPLE} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]

\[ ?p = \text{ALL SHORTEST SIMPLE} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]

\[ ?p = \text{SIMPLE} \ (\text{start})=[\text{regex}]=\rightarrow(\text{end}) \]
\( ?p = \text{ANY TRAIL} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{ANY SHORTEST TRAIL} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{ALL SHORTEST TRAIL} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{TRAIL} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{ANY ACYCLIC} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{ANY SHORTEST ACYCLIC} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{ALL SHORTEST ACYCLIC} \ (\text{start})=[\text{regex}]=>(\text{end}) \)

\( ?p = \text{ACYCLIC} \ (\text{start})=[\text{regex}]=>(\text{end}) \)
Any benchmark for this?
MillenniumDB Path Query Challenge

There is now:

- Benchmark for **returning paths**
- **Challenge #1**: real-world
  - Large data (WikiData derivate)
  - Real queries (WikiData query log)
  - Maintains real query distribution (what the users asked)
- **Challenge #2**: synthetic
  - Small graph (parameterizable)
  - Large number of paths (exponential)
  - Stress test for the choking points

And a reference implementation (**MillenniumDB**)
Challenge #1: real-world

The **data**:

- WikiData truthy dump
- Only direct properties (Q-nodes, P-edges)
  - Basically, the structure of WikiData
- As RDF: 1.25 billion triples
- As a graph: 1.25 billion edges, 364 million nodes

Also see: WDBench
Challenge #1: real-world

The queries:
- WikiData public query logs
- Code 500 queries (timeout)
- Total of 659 SPARQL queries:
  - 6 queries:  
  - 586 queries:  
  - 67 queries:
Challenge #1: real-world

What to do with the queries:

\[ ?p = \text{ANY WALK} \ (\text{StartConstant})=[\text{regex}]=>(?\text{end}) \]

\[ ?p = \text{ANY SHORTEST WALK} \ (\text{StartConstant})=[\text{regex}]=>(?\text{end}) \]

\[ ?p = \text{ALL SHORTEST WALK} \ (\text{StartConstant})=[\text{regex}]=>(?\text{end}) \]

\[ ?p = \text{ANY SIMPLE} \ (\text{StartConstant})=[\text{regex}]=>(?\text{end}) \]

...
Challenge #1: real-world

What to do with the queries:

\[ ?p = \text{ANY WALK} \ (\text{StartConstant})=[\text{regex}]=>(\text{?end}) \]

\[ ?p = \text{ANY SHORTEST WALK} \ (\text{StartConstant})=[\text{regex}]=>(\text{?end}) \]

\[ ?p = \text{ALL SHORTEST WALK} \ (\text{StartConstant})=[\text{regex}]=>(\text{?end}) \]

\[ ?p = \text{ANY SIMPLE} \ (\text{StartConstant})=[\text{regex}]=>(\text{?end}) \]

...
Challenge #1: real-world

What Cypher supports (natively):

\[
?p = \text{ANY SHORTEST WALK } (\text{StartConstant})=[\text{regex}]=>(\text{?end})
\]

\[
?p = \text{ALL SHORTEST WALK } (\text{StartConstant})=[\text{regex}]=>(\text{?end})
\]

\[
?p = \text{TRAIL } (\text{StartConstant})=[\text{regex}]=>(\text{?end})
\]
Challenge #1: real-world

What MillenniumDB supports:

- About 90% of all queries
- But with every possible semantics
Challenge #1: real-world

Benchmarking parameters:

- Some queries have >80 million results (paths)
- Limit to 100,000 paths
  - Does any user want more?
  - For a fixed (start,end) 1000 paths are 1000 results
- Timeout at 1 minute
  - What WikiData uses
  - Allows testing if query interruption works
Challenge #1: some results
Challenge #1: some results

- **B++tree**
- **CSR**
- **My fancy algorithm**

**ANY**

**ALL SHORTEST**

- Shortest
Challenge #1: some results

B++tree
CSR
My fancy algorithm

Total construction time: 52sec
Size: 4GB
Edges: >>100 million

Shortest
Challenge #1: some results

SIMPLE: B+Tree vs BCSR (Full Cache)

ANY

ALL

ALL SHORTEST

Time (ms)

1 2 3 4

1 2 3 4

1 3

(1) B+Tree (BFS)  
(2) B+Tree (DFS)  
(3) BCSR Full Cache (BFS)  
(4) BCSR Full Cache (DFS)
Challenge #1: some results
Challenge #2: synthetic

- 3n+1 nodes, 4n edges
- $2^n$ paths
- All are: shortest, trail, simple!
Challenge #2: some results

![Diagram showing performance comparisons between Neo4J and MDB for ANY SHORTEST and ALL SHORTEST cases. The graphs plot the number of nodes against the number of iterations for different values of \( k \).]
In conclusion

Go and benchmark your system!

Thank you!