Graph databases are a promising branch of storage systems. We have proposed a new algorithm for querying such databases with the aim to ensure the best possible worst-case time complexity. Our approach is based on the recent results from the field of graph homomorphism and uses them in the context of graph databases. By a series of experiments we have shown that our algorithm is faster than the leading graph database Neo4j in some of the scenarios.

**ASK THE GRAPH**

- **QUERY (Graph G)**
  - Give me mutual friends of Bob and his brother!

- **DATABASE (Graph H)**
  - Names of the vertices include:
    - Bob
    - Alice
    - Thomas
    - Adam

**SOLUTION**

1. **Read the query graph**
   - [Diagram of the query graph]

2. **Compute a tree decomposition**
   - **Treewidth 2** ("almost tree")
   - [Diagram of the tree decomposition]

3. **Compute the solution for one of the nodes**
   - Conventional methods may be used.

4. **Use resolved nodes to solve neighbours.**
   - **Trick:** Dismiss solutions that are known to be irrelevant for the original problem.
   - This is a local property within the tree. Repeat until an unresolved node exists.

5. **Enumerate solutions**
   - Combine matching solutions from individual nodes.

**TREEWIDTH**

- Measures similarity of a graph to a tree.
- The size of the largest portion of a graph, that has to be treated together

<table>
<thead>
<tr>
<th>Treewidth 0</th>
<th>Treewidth 1</th>
<th>Treewidth 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(trees)</td>
<td>(without edges)</td>
<td>(series parallel graphs)</td>
</tr>
</tbody>
</table>

**KNOWN RESULTS**

For simple undirected graphs the decision problem (i.e. decide whether a homomorphism exists) is in:

- **NP-C** for unrestricted queries (unless database is a bipartite graph) [1990, Hell and Nešetřil]
- **PTIME** for queries of bounded treewidth
- **PTIME** iff queries have bounded treewidth modulo homomorphic equivalence (decision variant only) [2002, Dalmau et al.; 2007, Grohe]

**BEST RESULT**

Finding directed cycles of an increasing length in a DAG

- **Our algorithm**
- **Neo4j**

**Time (ms)** vs **Length of cycle in G**

**$O(V(T) \cdot tw(G) \cdot V(H)^{tw(G)+1} + OUT)$**

$V(T)$ - number of nodes, $tw(G)$ – treewidth of the query, $V(H)$ - number of vertices in the database, $OUT$ - number of results

Exponential only in the treewidth