Factorization matters in *large graphs*

Nikolay Yakovets
not your grandma’s graph engine!
A conjunctive query (CQ) is a “query graph”:
- Its “nodes” are the query’s binding variables; and “edges” between nodes are constituent queries

The answer of a CQ:
- Are tuples of nodes (embeddings) that match the conjuncts, joining in the way the query asks
Conjunctive queries - challenges

**Cardinality:** The evaluation of graph queries is (often) dominated by the size of the **intermediate results** (IR)

- Queries are often very **selective**
- But, during the evaluation, the size of the intermediate results can **grow exponentially** (in the size of the graph), due to many-to-many joins inherent in graph queries
The answer-graph approach

**Factorization:** One way of reducing the size of intermediate results is to apply the concept of factorization

- the common unique node-pair patterns
- we call these factorized node-pairs, **Answer Graph** (AG)

**Defactorization:** To find all final result tuples (i.e., embeddings), all we need to do is to defactorize this answer graph.
The evaluation model

Answer graph generation

- Edge extension: to fetch data edges from the graph
- Node burn-back: to ensure the generated AG is minimal (in size)

\[ G \]

\[ ?w \rightarrow ?x \rightarrow ?y \]

\[ r_1 \rightarrow 1 \rightarrow 5 \rightarrow 9 \rightarrow r_2 \]

\[ r_3 \rightarrow 8 \rightarrow 12 \]

\[ \times \]

\[ G = \]

\[ ?w \rightarrow ?x \rightarrow ?y \rightarrow ?z \]

\[ r_1 \rightarrow 1 \rightarrow 5 \rightarrow 9 \rightarrow r_2 \]

\[ r_3 \rightarrow 6 \rightarrow 10 \rightarrow 13 \]

\[ = \]

\[ r_1 \rightarrow 6 \rightarrow 14 \]

\[ r_2 \rightarrow 4 \rightarrow 15 \]

\[ r_3 \rightarrow 3 \rightarrow 11 \]

edge extension
Node burn-back is a cascaded filter operation and has three processes:

- Nodes of AG that are not extendable with new edges are removed.
- Removing any node from AG will trigger the removal of edges that are attached to this node along with removal of nodes at the other side of removed edges.
Node burn-back is a cascaded filter operation and has three processes

- Nodes of AG that are not extendable with new edges are removed
- Removing any node from AG will trigger the removal of edges that are attached to this node along with removal of nodes at the other side of removed edges

**burning nodes - step 1**
Node burn-back is a cascaded filter operation and has three processes

- Nodes of AG that are not extendable with new edges are removed
- Removing any node from AG will trigger the removal of edges that are attached to this node along with removal of nodes at the other side of removed edges
Node burn-back is a cascaded filter operation and has three processes

- Nodes of AG that are not extendable with new edges are removed
- Removing any node from AG will trigger the removal of edges that are attached to this node along with removal of nodes at the other side of removed edges

\[ \text{done burning nodes} \]
The evaluation model

**Embedding generation**

- generates the final answer tuples of a CQ from an AG
- executing a join tree to further filter out tuples which do not belong to the final result

**Answer graph**

```
?d  ?c  ?y  ?x
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>owns</td>
<td>sameAs</td>
<td>isAffiliatedTo</td>
<td></td>
</tr>
</tbody>
</table>
```

**Embedding plan**

```
linksTo

| participatedIn | created | ...
|----------------|---------|-----|
```

**Embeddings**

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>13</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>11</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>60</td>
<td>13</td>
<td>2</td>
<td>10</td>
<td>6</td>
<td>11</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>60</td>
<td>13</td>
<td>2</td>
<td>10</td>
<td>15</td>
<td>11</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>
```
select distinct ?x, ?m, ?y, ?z, ?a
?b, ?c, ?d, ?e, ?f
where {
?x linksTo ?m .
?x isAffi liatedTo ?y .
?x wasBornIn ?z .
?m participatedIn ?b .
?m created ?a .
?y owl:sameAs ?c .
?y owns ?d .
?z isLocatedIn ?e .
?z isPreferredMeaningOf ?f .
}
The planners

A cost-based dynamic programming approach to find an optimal plan for evaluating the answer graph (AG) of a query

The plan space

- all execution orders of edges/sub-patterns in the CQ (left-deep, right-deep, zig-zag) + bushy
- finding the best plan is equivalent to enumerating over all possible orders

Plan enumeration and a cost model

- Dynamic programming to find the optimal order
- In each iteration, the planner uses its cardinality estimators to calculate the cost of a tree plan and updates the cardinality of nodes due to the node burn-back

The evaluation model

- execute based on the selected order of edges
- node burn-back filters the dead-end nodes accordingly
We call an answer graph ideal if it contains only those edges which participate in at least one final embedding.

Theorem: Node burn-back results in an ideal AG for acyclic graph CQs with number of cascaded semi-joins bounded in $O(|Q|)$. 

Pf.:

- node burn-back results in an ordered sequence of semi-joins which contain (in correct order) a semi-join ordering produced by the GYO algorithm
- this corresponds to a bounded full reducer semi-join program which guarantees no dangling tuples in the AG for acyclic CQs
What about cyclic CQs?

Node burn-back **does not** generate ideal AG for cyclic CQs in a fixed number of cascading semi-joins:

- some of the edges will not participate in the final embeddings
- we can still use this AG in the embedding generation, but it will be more expensive
- can we find an ideal AG for cyclic CQs? And at what cost?

```
\[ Q \]_G = \{3,4,2,1\} \\
\{7,8,6,5\}
```
We **triangulate** a query graph to reduce the AG further:

- during evaluation, additional end-points which correspond to triangles are materialized
- this materialization becomes an edge in a query graph called a **chord**
We **triangulate** a query graph to reduce the AG further:

- during evaluation, additional end-points which correspond to triangles are materialized
- this materialization becomes an edge in a query graph called a *chord*
We triangulate a query graph to reduce the AG further:

- during evaluation, additional end-points which correspond to triangles are materialized
- this materialization becomes an edge in a query graph called a chord
We **triangulate** a query graph to reduce the AG further:

- whenever a chord intersects a query edge (*-deep, zig-zag plan) or another chord (bushy plan), a **seal** happens
- a seal triggers the **edge burn-back** which removes the chord edges which don’t participate in the final embeddings, eventually removing "spurious" edges in the AG, on cascade
We **triangulate** a query graph to reduce the AG further:

- whenever a chord intersects a query edge (*-deep, zig-zag plan) or another chord (bushy plan), a seal happens
- a seal triggers the **edge burn-back** which removes the chord edges which don’t participate in the final embeddings, eventually removing “spurious” edges in the AG, on cascade

![Query graph](image1)

![Answer graph](image2)

**edge burn-back!**
We **triangulate** a query graph to reduce the AG further:

- whenever a chord intersects a query edge (*-deep, zig-zag plan) or another chord (bushy plan), a **seal** happens
- a seal triggers the **edge burn-back** which removes the chord edges which don’t participate in the final embeddings, eventually removing “spurious” edges in the AG, on cascade
We **triangulate** a query graph to reduce the AG further:

- whenever a chord intersects a query edge (*-deep, zig-zag plan) or another chord (bushy plan), a **seal** happens
- a seal triggers the **edge burn-back** which removes the chord edges which don’t participate in the final embeddings, eventually removing “spurious” edges in the AG, on cascade

**Query graph**

**Answer graph**

**edge burn-back!**
We **triangulate** a query graph to reduce the AG further:

- there are many different ways to triangulate the query graph
- similar to node burn-back, a DP cost-based enumeration is used to decide the best way to triangulate
Ideal answer graph

We call an answer graph ideal if it contains only those edges which participate in at least one final embedding.

Theorem: Edge burn-back results in an ideal AG for cyclic graph CQs with treewidth of 2 in cascade of at most $O(|Q|)$.

Pf.:
- triangulation corresponds to a tree decomposition of the query graph (with a max. treewidth = 2)
- similar to node burn-back, the seal on cascade generates a semi-join program which contains (in correct order) a valid bounded full reducer program produced by the GYO algorithm ran on the tree decomposition
- this guarantees no dangling tuples in the materialized triangles
- with easy book-keeping, we can remove the corresponding edges from the binary edge relations to guarantee no dangling tuples there
Ideal answer graph

We call an answer graph **ideal** if it contains only those edges which participate in at least one final embedding.

We can handle queries with higher treewidth graphs, but this requires more **materialization** to produce the ideal AG or using **fix-point cascade**

- Ultimately, this is a cost-based decision whether the extra materialization is worth the effort
- 99% of queries in practice are near acyclic
Implementation. WireFrame is implemented on top of PostgreSQL

- Edgifier in the first phase outputs an optimal left-deep tree plan
- For defactorizer, we use a greedy approach to generate a tree plan
- Node burn-back procedure is implemented via procedural SQL

Fuller-featured implementation is available in ...

AvantGraph
Experiments

- The size of the AG is exceedingly smaller that the number of embeddings
  - For Q2, 3000X smaller
  - this indicates lots of M-M joins and multiplicative effect (also in the IR)

- AG achieves excellent performance on such queries
  - as it avoids redundant edge walks in the IR
Experiments

Employing only node burn-back does not guarantee the ideal AG

- The resulting AGs can be significantly larger than the ideal AG
- For this reason, WF employing only node burn-back was slower on some of the cyclic CQs
- Even so, the overall performance was quite good
Thank you!

For more information, see our EDBT 21 paper: