Relational Databases can Handle Graphs Too!

Altan Birler

Technische Universität München
• UMBRA: Very fast Relational DBMS

• LDBC-BI: OLAP Graph Workload. 2 Graph, 1 Relational System

• Graph queries $\rightarrow$ SQL
• UMBRA: Very fast relational DBMS

• LDBC-BI: OLAP Graph Workload. 2 Graph, 1 Relational System

• Graph queries → SQL

• Umbra is fast at executing every single query
  • Including the shortest path queries!
umbra-db.com/interface
• A relational DB is great at executing *every single graph query*

• *What is going on here?*

```cpp
// Umbra.cpp: L192
if (query == IDbSql)
  executeFastCppImplementation();
```
The Graph Perspective

• “Navigate deep hierarchies”  
  neo4j.com/developer/graph-database/

• Connections
The Graph Perspective
The Graph Perspective

Scalable?

Altan → Cousin → Wife → Fish → Color → Red
Scalable?  Now you are thinking with relations.
The Relational Perspective

• A scalable model of the world: “SQL is embarrassingly parallel”

• **Big** (multi-)sets of unordered data

• Highly scalable, deeply researched, simple, standard operators
  • *Join, Group By*

• **Breadth is scalable**
  • *Depth is not*
Is My Query Scalable?

• Can you express it with set oriented relational algebra?
  
  • Yes: Most likely scalable
  • No: You might have some trouble
How To Scale LDBC BI Queries

• Express them in relational algebra (SQL)

• Eliminate depth, increase breadth
Eliminate depth

- Works since there are no link/cut operations
- Unforeseen (positive) consequences
  - Removing recursions improves cardinality estimates which improve accuracy of the optimizer
- Query: Average number of messages per forum
  - 800ms vs 80ms
# How to Beat Umbra

## Execution
- As fast as (faster than) highly optimized C++ code you would specifically write for a query.
- Highly scalable algorithms, WCOJ [1]
  - Death to $O(n^2)$
- JIT compilation of queries
- Morsel based parallelism
- Missing graph specific algorithms

*Not likely to improve by large margins*

## Optimization
- Unnesting arbitrary queries [2]
- Join ordering with optimal DP [3,4]
  - Adaptive optimization for huge joins
    - (high quality plans for high depth)
- Rule based optimization is not always consistent.
  - Order of application matters.
- Equivalent queries:
  - Some more equal than others

*Lots of potential for improvement*

## Statistics
- Statistics on base relations:
  - Great
- Recently saw great improvements [5]
- If isKey(attribute):
  - amazingEstimates();
  - Else:
    - startCrying();
- Exceptionally hard problem

*Just getting started!***
LDBC BI SQL Queries

• The queries changed over time
  • Over 10x improvement gained by rewriting queries
  • *The optimizer should have been doing what we had to do by hand!*
    • Remove redundant joins with redundant relations
    • Common subquery elimination

• Are you interested in execution?
  • Check out the latest query versions

• Are you interested in optimization?
  • Go through the git history and check out earlier query versions
SQL Shortest Path (PostgreSQL dialect)

shorts(dir, gsrc, dst, w, dead, iter) as (
  
  select false, f, f, 0::double precision, false, 0 from srcs
  union all
  select true, t, t, 0::double precision, false, 0 from dsts
)
union all
with

  ss as (select * from shorts),

toExplore as (select * from ss where dead = false order by w limit 1000),
  -- assumes graph is undirected
newPoints(dir, gsrc, dst, w, dead) as (          
    select e.dir, e.gsrc as gsrc, p.dst as dst, e.w + p.w as w, false as dead
    from path p join toExplore e on (e.dst = p.src)
    union all
    select dir, gsrc, dst, w, dead or exists (select * from toExplore e where e.dir = o.dir and e.gsrc = o.gsrc and e.dst = o.dst) from ss o
  ),
fullTable as (                                     
    select distinct on(dir, gsrc, dst) dir, gsrc, dst, w, dead
    from newPoints
    order by dir, gsrc, dst, w, dead desc
  ),
found as (                                          
    select min(l.w + r.w) as w
    from fullTable l, fullTable r
    where l.dir = false and r.dir = true and l.dst = r.dst
  )
select dir,
gsrc,
dst,
w,
dead or (coalesce(t.w > (select f.w/2 from found f), false)),
e.iter + 1 as iter
from fullTable t, (select iter from toExplore limit 1) e
Dijkstra’s Algorithm

Visit nodes one by one by increasing distance
Invariant: Every path within the circle has been seen
Dijkstra’s Algorithm Modified

Visit nodes **1000s at a time** by increasing distance
**Invariant:** Every path within the circle has been seen
We have to make sure no shorter path is available

Additional improvement: Bidirectional search
Hacking SQL Recursion

• Can’t access results of arbitrary recursion steps
  • So just propagate everything you ever compute at every step!
  • Absolutely horrible, destroys memory and efficiency
    • But we still beat the other graph systems!
    • This emphasizes the importance of breadth of depth
SQL Shortest Path

shorts(dir, gsrc, dst, w, dead, iter) as (
    select false, f, f, 0::double precision, false, 0 from srcs
    union all
    select true, t, t, 0::double precision, false, 0 from dsts
)
union all

with ss as (select * from shorts),
toExplore as (select * from ss where dead = false order by w limit 1000),
-- assumes graph is undirected
newPoints(dir, gsrc, dst, w, dead) as (  
    select e.dir, e.gsrc as gsrc, p.dst as dst, e.w + p.w as w, false as dead  
    from path p join toExplore e on (e.dst = p.src)  
    union all  
    select dir, gsrc, dst, w, dead or exists (select * from toExplore e where e.dir = o.dir and e.gsrc = o.gsrc and e.dst = o.dst) from ss o
),
fullTable as (  
    select distinct on(dir, gsrc, dst) dir, gsrc, dst, w, dead  
    from newPoints  
    order by dir, gsrc, dst, w, dead desc
),
found as (  
    select min(l.w + r.w) as w  
    from fullTable l, fullTable r  
    where l.dir = false and r.dir = true and l.dst = r.dst
)
select dir,
gsrc,
dst,
w,
dead or (coalesce(t.w > (select f.w/2 from found f), false)),
e.iter + 1 as iter  
from fullTable t, (select iter from toExplore limit 1) e
References


