NAVIGATIONAL QUERIES OVER GRAPHS

Domagoj Vrgoč

PUC Chile
Institute for Foundational Research on Data
THE MESSAGE

- Analysis vs synthesis
  - Push the envelope forward
  - Understand what we have thus far
  - The latter seems to be out of fashion

- What I would like to show:
  - Sometimes it is worth going back to the basics
We all know it, so what will we do?
  - Show that it can evaluate navigational queries efficiently
  - It can return paths as part of the query
  - Allows for "fully compositional" semantics of graph queries

How do I know these things?
  - Theoreicians have been saying something similar all along
  - Several papers on the topic
  - We built a graph database at IMFD that shows this
GRAPH DATABASES
WHY GRAPH DATABASES?

▪ More intuitive conceptualization:
  ▪ Naturally model relationships (e.g. social networks, bio pathways)

▪ Potential efficiency gains:
  ▪ Navigation vs. doing a bunch of joins
  ▪ This is where I'm trying to drive the discussion

▪ Fashion?
**Graphs in Practice I**

- Edge labeled graphs (RDF)

- Allows recording information in a compact way:
  - Node/edge *is* the data
  - The connections can be coded as triples
    - E.g. (Clint Eastwood, acts_in, Unforgiven)
GRAPHS IN PRACTICE II

- **Property graphs**

- **More realistic:**
  - Allows recording data using attribute values
  - Implicit typing through node/edge labels (good for indexing)
  - More robust than RDF
  - Node/Edge IDs are explicit
GRAPH DATABASE SYSTEMS

- Edge labelled graphs
  - RDF systems
  - SPARQL query language (W3C standard)
  - Multiple (reasonable) implementations (Virtuoso, Blaze Graph, Jena)

- Property graphs
  - Neo4j with the Cypher query language
  - Tinkerpop/Gremlin
  - TigerGraph

- MillenniumDB
QUERYING GRAPHS
GRAPH QUERY FEATURES

- Pattern matching
  - Matching a small graph (pattern/query) onto a bigger one (data)
  - Basically conjunctive queries
  - Different semantics available based on user needs

- Navigational queries
  - Designed to explore connections between points in graph
  - Most common query: path specification
  - Different semantics can have huge impact on evaluation efficiency

See [AABHRV17] for more details
NAVIGATION IN GRAPHS

- Exploring connections
  - Paths whose length is not known in advance
  - Patterns repeated in a "regular" manner

- Examples:
  - Friend-of-a-friend relation in a social network
  - Bacon/Erdos number
  - Shortest routes between two places
**Path Queries**

- The most basic navigational query class
  - Gives (pairs of) nodes connected by a path conforming to the query

- Usually specified using regular expressions
  - The labels on the edges along a path form a word in the language of the expression

- Present in many languages:
  - SPARQL property paths
  - Neo4J queries
  - Most theoretical literature takes paths as the base for navigation
SPECIFYING PATH QUERIES

- Expressions of the form

\[ x \xrightarrow{\text{regex}} y \]

- \( x \) and \( y \) are variables or constants (IDs)
- \( \text{regex} \) is a regular expression

- "Semantics": All \((x,y)\) connected by a path whose "label" is in the language of \( \text{regex} \)
EXAMPLE 1 — BACON NUMBER

- Actors that have a Bacon number in a movie database
EXAMPLE 1 — BACON NUMBER

\[ x \xrightarrow{(\text{acts\_in} \cdot \text{acts\_in}^-)^*} \text{Kevin Bacon} \]
EXAMPLE 1 — BACON NUMBER

```sql
SELECT ?x
MATCH (??x)?=([:(acts_in/^:acts_in)*]=>(Kevin Bacon))
```

Diagram showing relationships between movies and actors, with nodes and directed edges indicating acting or directing roles.
EXAMPLE 1 — BACON NUMBER

```
SELECT ?x
WHERE { ?x (:acts_in/^:acts_in)* Kevin_Bacon }
```
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/^:acts_in)* Kevin_Bacon }
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/^:acts_in)* Kevin_Bacon }
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/^acts_in)* Kevin_Bacon }

Yago: Documentary
---
N is a Number
---
Paul Erdős
---
Tomasz Łuczak
---
The Mill and The Cross
---
Charlotte Rampling

Yago: Movie
---
Mystic River
---
Kevin Bacon
---
Clint Eastwood
---
Sean Penn
---
Searching for Debra Winger
---
Rosanna Arquette

---
rdfs:type

---
:acts_in

---
:director

---
:acts_in
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/^:acts_in)* Kevin_Bacon }

Yago: Documentary
  rdfs:type
  :acts_in
  Tomasz_Luczak

N is a Number
  :acts_in
  Paul_Erdős

The Mill and The Cross
  :acts_in
  Charlotte_Rampling

Yago: Movie
  :director
  Searching_for_Debra_Winger

Mystic River
  :director
  Clint_Eastwood

Kevin Bacon
  :director
  Sean_Penn

Rosanna_Arquette
  :director
  Searching_for_Debra_Winger
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/^:acts_in)* Kevin_Bacon }
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/~:acts_in)* Kevin_Bacon }

The diagram shows a network of relationships between various entities, including movies, directors, and actors. The query selects entities that are connected to Kevin Bacon through the :acts_in relationship zero or more times.
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/~:acts_in)* Kevin_Bacon }

Diagram showing relationships between movies and actors, including "N is a Number," "Mystic River," "The Mill and The Cross," "Searching for Debra Winger," and actors such as Kevin Bacon, Tomasz Luczak, Paul Erdős, Clint Eastwood, Sean Penn, Charlotte Rampling, and Rosanna Arquette.
EXAMPLE 1 — BACON NUMBER

SELECT ?x
WHERE { ?x (:acts_in/¯:acts_in)* Kevin_Bacon }
EXAMPLE 2

- Tags of what my friends-of-friends like in a social network

\[ x \xrightarrow{\text{knows}^+ \cdot \text{likes} \cdot \text{hasTag}} y \]
FURTHER EXAMPLES

- Taken from the WikiData dataset user queries

- Bodies of water ending in the Black Sea:

  ```sql
  SELECT ?watercourse
  WHERE { ?watercourse :drains_to* Black_Sea }
  ```

- Hierarchies/Taxonomies:

  ```sql
  SELECT ?creature
  WHERE { ?creature :instance_of/:subclass_of* Human }
  ```
How should one implement path queries in a graph database?
**PATH QUERIES - SEMANTICS**

- **What is a path?**
  - A sequence of nodes/edges: n₁ e₁ n₂ e₂ n₃ ...

- **When does a path conform to a regular expressions?**
  - When the labels of the edges give a word in its language

- **What do we return?**
  - Just the endpoints (?)
  - Paths satisfying the constraint (together with ?)
PATH QUERIES - SEMANTICS

- Returning only endpoints
  - Bag semantics: how many times? Can be infinite!
  - Set semantics: just give me the nodes connected by a "good" path!

```sql
SELECT ?x
MATCH (?x)=[:knows*]=>(Q2)
```
PATH QUERIES - SEMANTICS

- Returning paths as well:
  - There can be infinitely many (if a cycle is present)!
  - How would you express this in SPARQL?
  - What does bag semantics mean here?

SELECT ?x, ?p
MATCH (?x)=[?p :knows*]=>(Q2)
SEMANTICS — AVOIDING INFINITY

- Using simple paths:
  - No repeated nodes – cycles not a problem anymore
  - Immediately leads to intractability [MW95]
  - A reason to exclude them from SPARQL standard [ML12]

- No-repeated-edges:
  - Similarly avoids infinity
  - The semantics of Cypher
  - Equivalent to simple paths
  - If implemented in full (all matches) runs into the same issues
REASONABLE SOLUTIONS

- Shortest paths:
  - No infinity issues
  - Allows bag semantics with endpoints (count all shortest paths)
  - Allows recurring paths

- Arbitrary paths:
  - Can be infinite
  - Works when returning endpoints with set semantics
  - SPARQL standard

- Both options have good theoretical properties
- Many algorithms known for implementing them
HOW DO WE IMPLEMENT PATH QUERIES?

- Theoretician's answer ("This is trivial"): [MW95]
  - Graph is an automaton
  - Regular expression is an automaton
  - Do the cross product (on-the-fly to be "efficient")
  - Do reachability check from start states to end states

- Which algorithms can do this?
  - BFS
  - DFS
  - A*
  - IDDFS
  - ...

HOW DOES THIS ACTUALLY WORK?

Kevin Bacon

Footloose

Lori Singer

John Lithgow

Crazy, Stupid, Love

Dianne Wiest

Steve Carell

J. Moore

Julianne Moore
HOW DOES THIS ACTUALLY WORK?

SELECT ?x
MATCH (Kevin Bacon)=[([^:actor/:actor)*/:name] => (?x)
HOW DOES THIS ACTUALLY WORK - DFS

$:actor$  $^\text{actor}$  $\text{Kevin Bacon}$

$q_{\text{mov}}$  $q_{\text{act}}$  $q_{\text{end}}$  $q_{\text{name}}$
HOW DOES THIS ACTUALLY WORK - DFS

Kevin Bacon
Footloose

$q_{mov} \rightarrow q_{act}$

$q_{act} \rightarrow q_{name}$

$q_{name} \rightarrow q_{end}$

$q_{end}$

$:^{actor}$

$:^{actor}$
HOW DOES THIS ACTUALLY WORK - DFS
HOW DOES THIS ACTUALLY WORK - DFS
HOW DOES THIS ACTUALLY WORK - DFS

Kevin Bacon
Footlose
Lori Singer
Shortcuts
J. Moore
HOW DOES THIS ACTUALLY WORK - DFS

Kevin Bacon

Footlose

Lori Singer

Shortcuts

J. Moore

Julianne Moore
**How Does This Actually Work - DFS**

- Can produce long/unintuitive paths
HOW DOES THIS ACTUALLY WORK - BFS

\(q_{\text{mov}}\)

\(q_{\text{act}}\)

\(q_{\text{name}}\)

\(q_{\text{end}}\)

\(q_{\text{act}}\) Kevin Bacon
HOW DOES THIS ACTUALLY WORK - BFS

Kevin Bacon

Footloose

Crazy, Stupid, Love

:^:actor

:^:actor

:^:actor

:actor

:name

:end

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor

:actor
HOW DOES THIS ACTUALLY WORK - BFS

Kevin Bacon
Footlose
Lori Singer
John Lithgow
J. Moore
Crazy, Stupid, Love
Steve Carell

:^:actor
:actor
:name
:actor
:actor
:actor
:actor
:actor
:actor
:actor
:actor
:actor
:actor
HOW DOES THIS ACTUALLY WORK - BFS

Kevin Bacon
Footlose
Lori Singer
John Lithgow
J. Moore
Steve Carell
Crazy, Stupid, Love

$q_{act}$
$q_{mov}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
$q_{act}$
HOW DOES THIS ACTUALLY WORK - BFS

- Gets the first result slower than necessary

![Graph](image.png)
HOW DOES THIS ACTUALLY WORK — A* 

Heuristic: get close to $q_{end}$ 

$q_{mov}$ 

$q_{act}$ 

:^:actor 

:name 

$q_{end}$ 

$q_{act}$ 

Kevin Bacon
HOW DOES THIS ACTUALLY WORK — A*

Heuristic: get close to $q_{end}$

Kevin Bacon

Footloose

Crazy, Stupid, Love

$q_{act}$

$q_{mov}$

$q_{act}$

$q_{mov}$

$q_{mov}$
HOW DOES THIS ACTUALLY WORK – A*

**Heuristic:** get close to \( q_{\text{end}} \)

\[
\begin{align*}
q_{\text{mov}} & \quad :\text{name} \quad q_{\text{end}} \\
\quad :\text{actor} & \quad q_{\text{act}} & \quad :\text{actor} & \quad q_{\text{mov}} \\
\quad :\text{actor} & \quad q_{\text{act}} & \quad :\text{actor} & \quad q_{\text{act}} \\
\quad :\text{actor} & \quad q_{\text{act}} & \quad :\text{actor} & \quad q_{\text{act}}
\end{align*}
\]
**How Does This Actually Work — A**

**Heuristic:** get close to $q_{end}$

- $q_{mov}$
  - $:^\text{actor}$
  - $:^\text{name}$
- $q_{end}$
- $q_{act}$
- Footlose
  - $q_{mov}$
  - $:^\text{actor}$
  - Lori Singer
    - $:^\text{name}$
  - John Lithgow
    - $:^\text{name}$
- Kevin Bacon
  - $q_{act}$
  - $:^\text{actor}$
- Crazy, Stupid, Love
  - $q_{mov}$
  - $:^\text{actor}$
HOW DOES THIS ACTUALLY WORK — A*

Heuristic: get close to $q_{\text{end}}$

- Seems to bring the best of both worlds
So you would think this works
- Endpoints/set semantics
- No counting paths (standard)
SPARQL'S ODDITIES

```sparql
SELECT *
WHERE {
}
```
SOME APPROACHES

- On top of RDF3X [GBS13]
  - In-memory index used to compress graphs [SABW13]
  - Implements BFS to evaluate property paths
  - Good performance compared to other solutions
  - Endpoint/set semantics

- SQL based approach [RSV15]
  - Seminaive SQL-style recursion in SPARQL
  - Can do decently when evaluating property paths
  - Endpoint/set semantics
  - Controlling recursion depth shown better than manual joins
Some Approaches

- Query planners for path queries [YGG16]
  - Uses automaton of the regex to plan evaluation
  - Based on Postgres
  - Implemented using stored procedures

- Path index [FPP16]
  - Assumes bounded repetitions (not Kleene star)
  - Index "popular" path prefixes
  - Rewrite the query and run joins over this
SOME APPROACHES

- On Linked Data [BDRV17]
  - No data available locally (everything is accessed via an IRI)
  - Tests how BFS/DFS/A* perform
  - Allows incremental solutions and returning paths
  - For a small amount of solutions runs better than existing engines
WHAT ARE WE DOING NOW

- MillenniumDB
  - A persistend graph database engine
  - Based on standard relational techniques tweaked for graphs
  - Pipelined query evaluation
  - WCO query planner (in conjunction with Selinger) [HRRS19]
  - Full support for navigational queries [BDRV17]:
    - Endpoints/set semantics
    - Returning paths
    - Bag semantics with shortest paths
We use good old B+ trees

Relations required:
- NodeLabel ... e.g. (n1, Person)
- NodeKeyValue ... e.g. (n1, gender, male), (e1, role, Bill)
- FromTypeToEdge ... e.g. (n1, :acts_in, n2, e1)

and several of their permutations
**MILLENNIUM DB — PATH QUERIES**

- **FromTypeToEdge**  ... e.g. \((n_1, :\text{acts\_in}, n_2, e_1)\)
  - Support for BFS/DFS/A*
  - All implemented in a pipelined fashion (via a linear iterator)
  - How to return a result as soon as it is encountered: B+tree iter stays live
Some results on full WikiData (cca 1TB on disk)
- Q1 ... Bacon Number ... 160K results
- Q2 ... Places located in a Nato state ... 4.8M results
- Q3 ... Organizations dealing with EU capitals ... 17K results
- Q4 ... Where can I get to from the Netherlands ... 4K results
- Q5 ... Spouses of people born in a place in the US ... 35K results

What do you think the best algorithm is?
- Recall, you are now reading data from disk
- You have pages pinned in the buffer
- You have auxiliary data structures to keep track of visited nodes
MILLENNIUM DB – PATH QUERIES

- Basically, BFS wins in every experiment
Still holding strong, but DFS/A* do well

Identical graph with only 1% of all the results required
Why would BFS run so well?
  - If we are also returning paths, it is orders of magnitude faster

How many pages need to be pinned in the buffer?
  - DFS: length of the current path (stack height)
  - A*: all top elements on the priority queue
  - BFS: just one (the item on the top of the queue)

Also means parallel executions are feasible for BFS
BFS — OTHER ADVANTAGES

- We can return a single shortest path by default
- With a bit of tweaking can return all shortest paths

- Allows compositionality:
  - What does a graph query return?
  - Let's say a graph
  - But which one?
COMPOSITIONALITY

- Construct the graph:
  - All nodes/edges in fixed size pattern
  - All shortest paths in path queries
  - Assume that SELECT variables are your view of this graph
  - [G-CORE] tried something along these lines

```sql
SELECT ?x, ?y, ?z
MATCH (?x :Person)-[:knows]->(?y :Person),
      (?y)-[:lives_in]->(?z : City),
      (Kevin Bacon)=[(^:actor/:actor)*/:name]>>(?x)
```
LOOKING FORWARD

- GPU transitive closure
  - Seems to be much faster than the classical one

- Shrinking the graph:
  - Remove the noise, work with the paths alone
  - Preliminary results promising, but no paths [AHNRRS21]

- Learning the relevant parts of the graph:
  - Try to learn which part of the graph will be accessed by the query
LOOKING FORWARD

- Understanding indexing schemes:
  - With good guarantees/estimates
  - Which scheme goes with which algorithm/approach

- Query plans based on automata, and not SQL-like plans:
  - Transitive closure approach a good option, but maybe missing the point
  - Compiling query into automata (what is the optimal one?)
  - This line of work was started in [YGG16]
**BOTTOM LINE**

- Do we know how to implement a graph database?
  - Does any existing system support path queries?
  - Do classical algorithms suffice?

- Do we know what we want to implement?
  - What types of queries
  - Which semantics
  - Ongoing standardization efforts

- Seems like there is work to be done 😊
Thank you!
REFERENCES

- [ACP12] Marcelo Arenas, Sebastián Conca, Jorge Pérez: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. WWW 2012.
- [BDRV16] Jorge Baier, Dietrich Daroch, Juan L. Reutter, Domagoj Vrgoč: Property Paths over Linked Data: Can It Be Done and How To Start? COLD@ISWC 2016
- [F12] Wenfei Fan: Graph pattern matching revised for social network analysis. ICDT 2012: 8-21
REFERENCES

- [BDRV17] Jorge Baier, Dietrich Daroch, Juan L. Reutter, Domagoj Vrgoč: Evaluating navigational RDF queries over the Web, HT 2017
REFERENCES


- [G-CORE] Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutiérrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, Hannes Voigt: G-CORE: A Core for Future Graph Query Languages. SIGMOD 2018