Center for Semantic Web Research

www.ciws.cl, @ciwschile

Three institutions

PUC, Chile

- Marcelo Arenas (Boss)
  - SW, data exchange, semistrucured data
- Juan Reutter
  - SW, graph DBs, DLs
- Cristian Riveros
  - data exchange, semistr. data
- Jorge Baier
  - planning, search
- Carlos Buil
  - SW

Univ. of Talca

Renzo Angles (SW)

Univ. of Chile

- Pablo Barceló (Deputy)
  - graph DBs, DB theory
- Claudio Gutierrez
  - SW, graph DBs
- Jorge Pérez
  - SW, data exchange
- Aidan Hogan
  - SW, semistr. data
- Bárbara Poblete
  - web mining, SNA
- Benjamín Bustos
  - multimedia

Alberto Mendelzon Workshop (AMW)

AMW 2016
PANAMA CITY

AMW 2016

- Panama City, 6-10 June, 2016
- PC Chairs:

  Altigran Soares da Silva (UFAM), Reinhard Pichler (TU Wien)
- Invited speakers:
  - Diego Calvanese (Data-driven verification)
  - Juliana Freire (Urban data)
  - Lise Getoor (Relational statistical learning)
  - Raghu Ramakrishnan (Big data)
- Long & short submissions (due on Feb. 29th, 2016)
- AMW School (4 tutorials), 4-5 June, 2016
- Very nice environment

Query Languages for Graph DBs: Bridging the Grap Between Theory and Practice

Pablo Barceló
DCC, Universidad de Chile
Center for Semantic Web Research (www.ciws.cl)

BACKGROUND AND OBJECTIVES

Graph databases

Trendy applications:

- Social network analysis
- Semantic web
- Scientific databases
- Software bug localization
- Geo-routing

Graph databases

Trendy applications:

- Social network analysis
- Semantic web
- Scientific databases
- Software bug localization
- Geo-routing

More in general:

- Wherever connections are as important as data

What is a graph database?

A data management system that exposes a graph data model.[1]

[1] *Graph databases*. Robinson, Webber, & Eifrem. O'Reilly, 2013.

What is a graph database?

A data management system that exposes a graph data model.[1]

Several existing graph DB engines and query languages:

- DEX/Sparksee - basic algebra
- IBM System G - Gremlin
- Neo4J - Cypher
- Oracle PGX - PGQL
- RDF stores (Virtuoso, AllegroGraph, Oracle, IBM) - SPARQL

[1] *Graph databases*. Robinson, Webber, & Eifrem. O'Reilly, 2013.

What graph databases are good for?

- Flexible modelling of interconnected data
- Agile evolution of the data model
- Scalable evaluation of join-intensive queries

My personal story

- Since 2009: Working on theory of query languages for graph DBs
- Since 2015: Working group of LDBC for the design of such language

My personal story

- Since 2009: Working on theory of query languages for graph DBs
- Since 2015: Working group of LDBC for the design of such language

Conclusion:

- Theory and practice are more connected than expected

Objectives

- Identify topics of common interest for theoreticians and developers
- Formalize relevant concepts (syntax, semantics, terminology, etc)
- Understand tradeoff expressiveness/efficiency

THE DATA MODEL: PROPERTY GRAPHS

The data model is important as it must be:

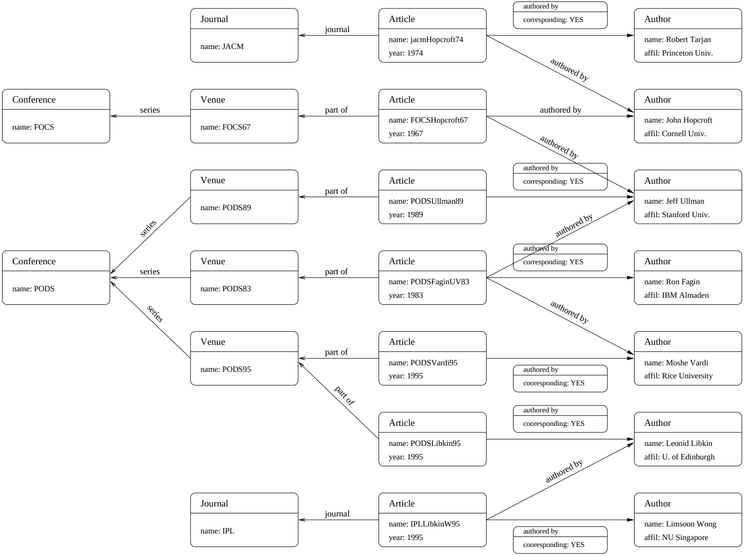
- Flexible enough to accomodate scenarios of practical interest
- Simple enough to allow for a clean presentation
- Expressive enough for theoretical issues to appear in full force

The data model is important as it must be:

- Flexible enough to accomodate scenarios of practical interest
- Simple enough to allow for a clean presentation
- Expressive enough for theoretical issues to appear in full force

This is accomplished by the model of property graphs

A property graph



What is a property graph then?

- It is a graph
- It is directed
- It is labeled in nodes and edges
- Nodes and edges can be attributed

What is a property graph then?

- It is a graph
- It is directed
- It is labeled in nodes and edges
- Nodes and edges can be attributed

What is a property graph then?

- It is a graph
- It is directed
- It is labeled in nodes and edges
- Nodes and edges can be attributed

What is a property graph then?

- It is a graph
- It is directed
- It is labeled in nodes and edges
- Nodes and edges can be attributed

What is a property graph then?

- It is a graph
- It is directed
- It is labeled in nodes and edges
- Nodes and edges can be attributed

GRAPH PATTERNS

Graph patterns are:

The basic unit for querying property graphs

Graph patterns are:

The basic unit for querying property graphs

Definition of graph pattern:

- A directed graph
- Nodes are given by variables $x, y, z, \ldots$
- Edges are given by variables $X, Y, Z, \ldots$
- Nodes and edges satisfy label and attribute constraints (selection)
  - E.g., $l(x) =$ Author, $l(Y) =$ authored by & Y@corresponding = YES
- Some of the variables are selected (projection)

Graph patterns are:

The basic unit for querying property graphs

Definition of graph pattern:

- A directed graph
- Nodes are given by variables $x, y, z, \ldots$
- Edges are given by variables $X, Y, Z, \ldots$
- Nodes and edges satisfy label and attribute constraints (selection)
  - E.g., $l(x) =$ Author, $l(Y) =$ authored by & Y@corresponding $=$ YES
- Some of the variables are selected (projection)

Graph patterns are:

The basic unit for querying property graphs

Definition of graph pattern:

- A directed graph
- Nodes are given by variables $x, y, z, \ldots$
- Edges are given by variables $X, Y, Z, \ldots$
- Nodes and edges satisfy label and attribute constraints (selection)
  - E.g., $l(x) =$ Author, $l(Y) =$ authored by & Y@corresponding = YES
- Some of the variables are selected (projection)

Graph patterns are:

The basic unit for querying property graphs

Definition of graph pattern:

- A directed graph
- Nodes are given by variables $x, y, z, \ldots$
- Edges are given by variables $X, Y, Z, \ldots$
- Nodes and edges satisfy label and attribute constraints (selection)
  - E.g., $l(x) =$ Author, $l(Y) =$ authored by & Y@corresponding = YES
- Some of the variables are selected (projection)

Graph patterns are:

The basic unit for querying property graphs

Definition of graph pattern:

- A directed graph
- Nodes are given by variables $x, y, z, \ldots$
- Edges are given by variables $X, Y, Z, \ldots$
- Nodes and edges satisfy label and attribute constraints (selection)
  - E.g., $l(x) = \text{Author}$, $l(Y) = \text{authored by}$ & Y@corresponding $= \text{YES}$
- Some of the variables are selected (projection)

Graph patterns are:

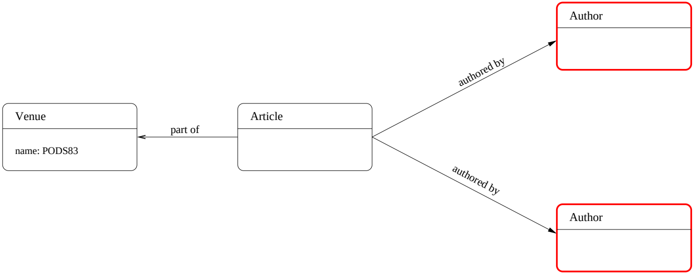
The basic unit for querying property graphs

Definition of graph pattern:

- A directed graph
- Nodes are given by variables $x, y, z, \ldots$
- Edges are given by variables $X, Y, Z, \ldots$
- Nodes and edges satisfy label and attribute constraints (selection)
  - E.g., $l(x) =$ Author, $l(Y) =$ authored by & Y@corresponding = YES
- Some of the variables are selected (projection)

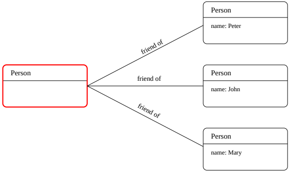
Example of a graph pattern

Find pairs of authors who coauthored a paper in PODS83:



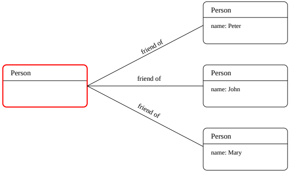
More examples

Get the common friends of Peter, John and Mary:

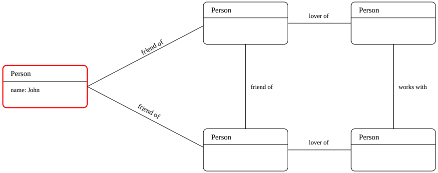


More examples

Get the common friends of Peter, John and Mary:



Find friends of John who are
(1) mutual friends, and (2) have lovers that are colleagues



Evaluation of graph patterns

1. Find all *matchings* of the pattern over the property graph
2. Project over the variables in the output

Evaluation of graph patterns

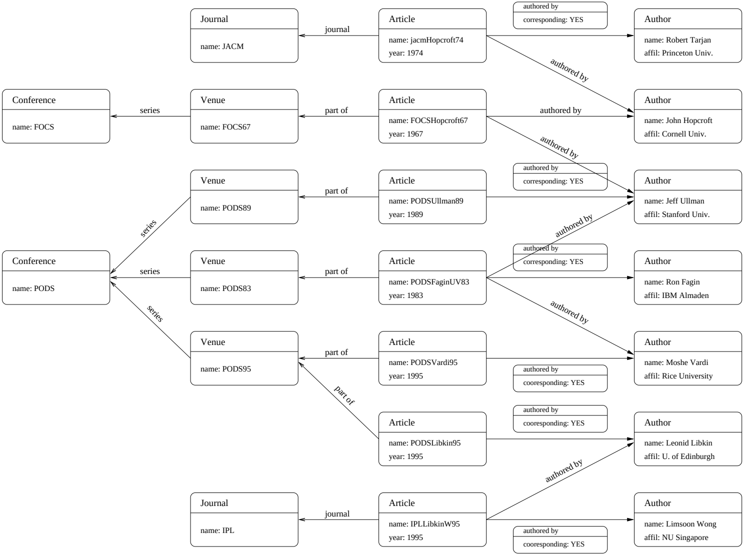
1. Find all *matchings* of the pattern over the property graph
2. Project over the variables in the output

Evaluation of graph patterns

1. Find all *matchings* of the pattern over the property graph
2. Project over the variables in the output

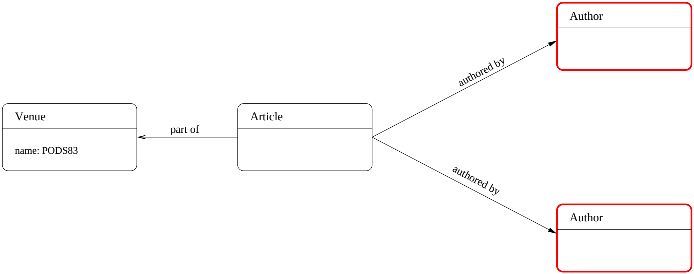
An example of evaluation

The property graph



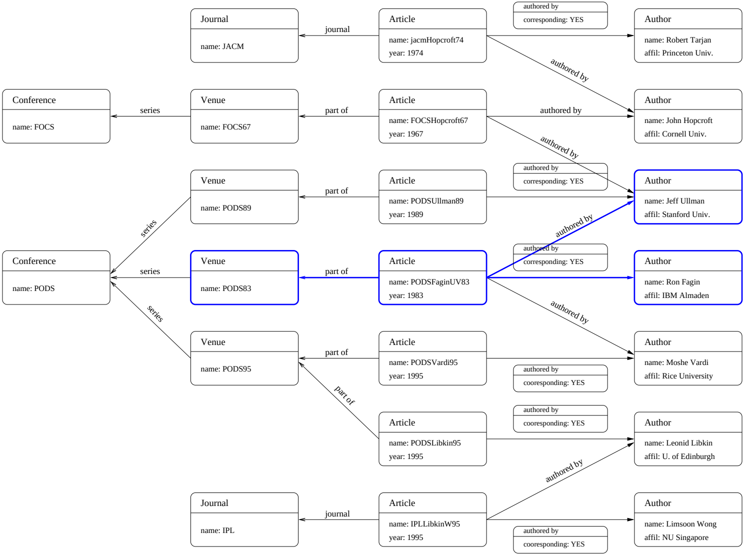
An example of evaluation

The graph pattern



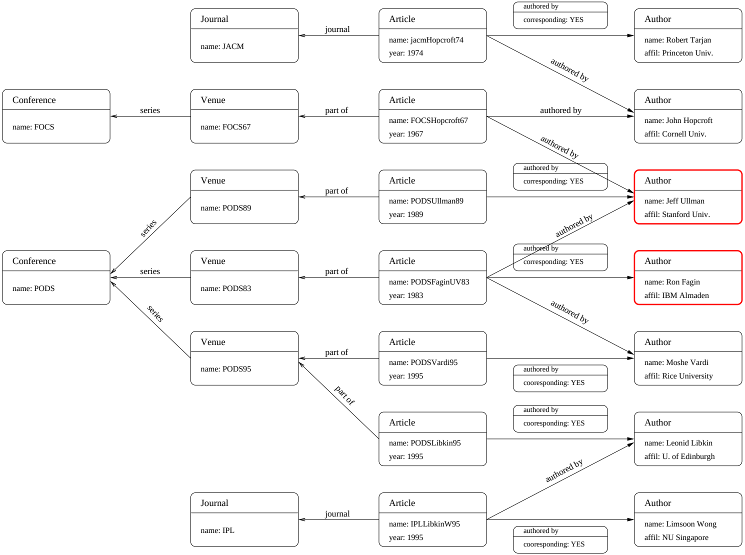
An example of evaluation

A matching



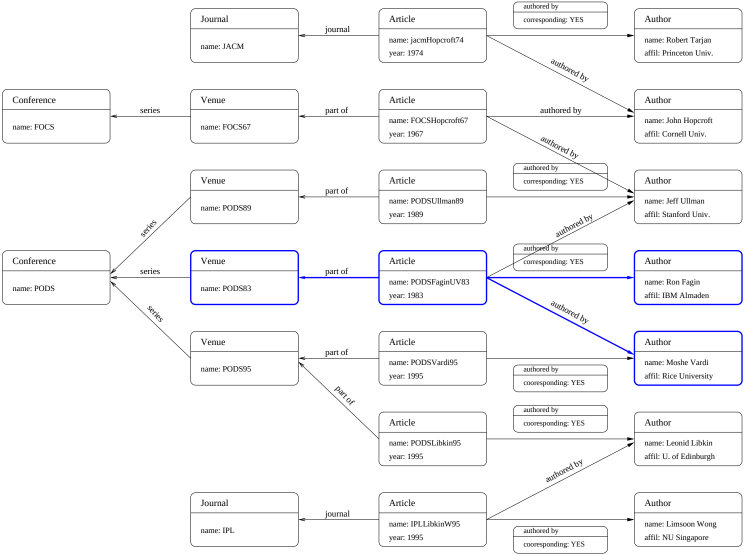
An example of evaluation

Its projection



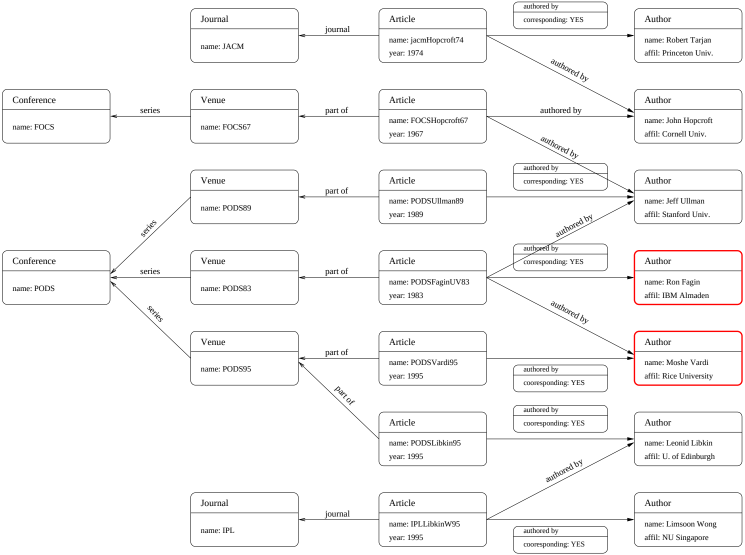
An example of evaluation

Another matching



An example of evaluation

Its projection



But what is a matching?

But what is a matching?

A mapping from:

- nodes of the pattern to nodes of the graph, and
- edges of the pattern to edges of the graph

which preserves the structure of the pattern in the graph

Different notions of matching

Homomorphism: No restriction on the mapping

- $\neq$ nodes in the pattern can map to the same node in the graph

Different notions of matching

Homomorphism: No restriction on the mapping

- $\neq$ nodes in the pattern can map to the same node in the graph

Mostly studied in database theory (PODS)

Different notions of matching

Isomorphism: Mapping is injective

- $\neq$ elements in the pattern map to $\neq$ elements in the graph

Different notions of matching

Isomorphism: Mapping is injective

- $\neq$ elements in the pattern map to $\neq$ elements in the graph

Mostly studied in database systems (SIGMOD)

Different notions of matching

Edge-injective: Self-describing

- $\neq$ edges in the pattern map to $\neq$ edges in the graph

Different notions of matching

Edge-injective: Self-describing

- $\neq$ edges in the pattern map to $\neq$ edges in the graph

Implemented in some graph DB engines (Neo4J)

Is there a matching?

Is there a matching?

An NP-complete problem

Is there a matching?

An NP-complete problem

How to address this problem?

Solution 1: Restriction on graph patterns

In many applications, graph patterns are *tame*:

- Homomorphism/isomorphism can be solved efficiently for them

Solution 1: Restriction on graph patterns

In many applications, graph patterns are *tame*:

- Homomorphism/isomorphism can be solved efficiently for them

Tame: The underlying graph is almost acyclic

- Bounded *treewidth* (database/graph theory)

Solution 2: Heuristics for real-world datasets

Structural optimization techniques for reducing search space:

- Join ordering, prunning, indexes (database systems)

Solution 2: Heuristics for real-world datasets

Structural optimization techniques for reducing search space:

- Join ordering, prunning, indexes (database systems)

Real databases have structure that can be exploited

Solution 3: Inexact evaluation

Use weaker forms of matching that can be evaluated efficiently:

- Bisimulations, approximations (database theory/systems)

Solution 3: Inexact evaluation

Use weaker forms of matching that can be evaluated efficiently:

- Bisimulations, approximations (database theory/systems)

Compromise the quality of the answer in favor of efficiency

Solution 4: Use different notions of complexity

Graphs and patterns are different beasts:

- Graphs are BIG, patterns are small

Solution 4: Use different notions of complexity

Graphs and patterns are different beasts:

- Graphs are BIG, patterns are small

Assume pattern is fixed (data complexity/database theory):

- Matching can be solved very efficiently

Solution modifiers on graph patterns

Relational operations:

- Union
- Difference
- Cartesian product

Solution modifiers on graph patterns

Relational operations:

- Union
- Difference
- Cartesian product

The language becomes relational complete

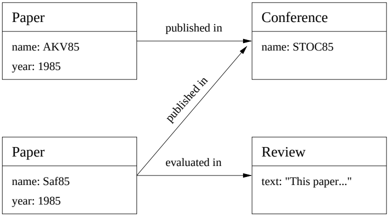
OPTIONAL: An important solution modifier

Allows to match parts of the data only if available

- Important in the context of semistructured data
- Developed by the RDF community
- Corresponds to *left-outer join* in relational algebra

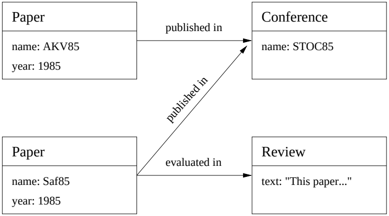
An example with OPTIONAL

The property graph



An example with OPTIONAL

The property graph

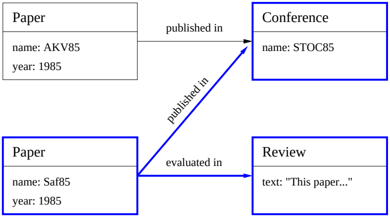


The pattern with optional

$$(x, \text{published\_in}, y) \text{ OPTIONAL } (y, \text{evaluated\_in}, z)$$

An example with OPTIONAL

A matching

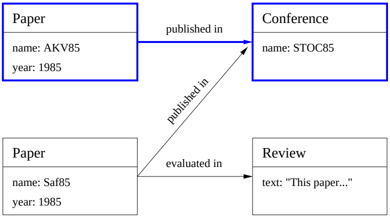


The pattern with optional

$$(x, \text{published\_in}, y) \text{ OPTIONAL } (x, \text{evaluated\_in}, z)$$

An example with OPTIONAL

Another matching



The pattern with optional

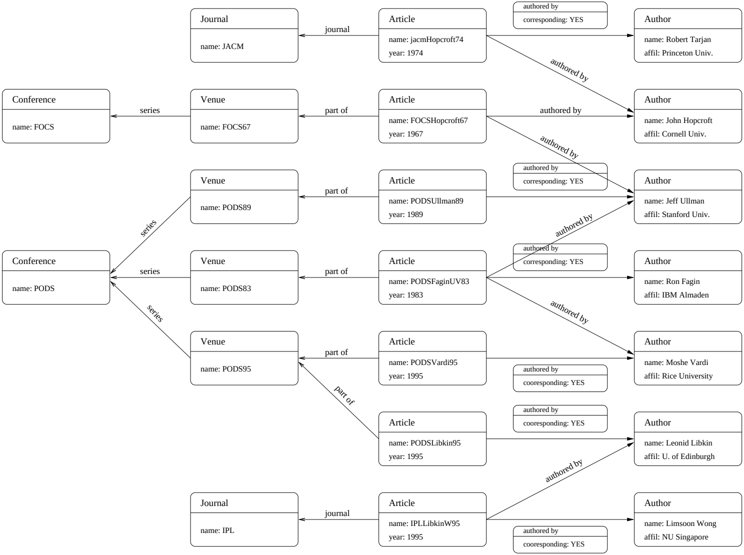
$$(x, \text{published\_in}, y) \text{ OPTIONAL } (x, \text{evaluated\_in}, z)$$

Conclusion

- Graph patterns are a versatile and simple language for querying PGs
- Graph pattern evaluation comes in different flavors
- This problem is challenging (theory/practice)
- Different operators can be added in order to increase expressiveness

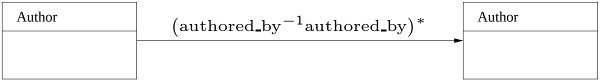
Graphs are there to be navigated

Recall our property graph?



Graphs are there to be navigated

Find pairs of authors linked by a coauthorship sequence



Do practical query languages navigate?

Very little:

- Check if there is a directed path between two nodes (DFS)

Do practical query languages navigate?

Very little:

- Check if there is a directed path between two nodes (DFS)

The previous query cannot be expressed (save for a few exceptions)

Do practical query languages navigate?

Very little:

- Check if there is a directed path between two nodes (DFS)

The previous query cannot be expressed (save for a few exceptions)

No support for regular path queries (database theory, RDF, DL)

- Is there a directed path whose label satisfies a regex?

Are RPQs harder to evaluate?

Are RPQs harder to evaluate?

Not really (in theory):

- Convert the regex into an automata
- Take the cross product of the property graph and the automata
- Check if there is a path from an initial to a final state (DFS)

Are RPQs harder to evaluate?

Not really (in theory):

- Convert the regex into an automata
- Take the cross product of the property graph and the automata
- Check if there is a path from an initial to a final state (DFS)

Cost is linear in the size of the data and the regex

But, what is the semantics?

Is there a path or a simple path?

- Database theory concentrates on the former (why?)
- Graph DB engines implement the latter (why?)

But, what is the semantics?

Is there a path or a simple path?

- Database theory concentrates on the former (why?)
- Graph DB engines implement the latter (why?)

But, what is the semantics?

Is there a path or a simple path?

- Database theory concentrates on the former (why?)
- Graph DB engines implement the latter (why?)

But, what is the semantics?

Is there a path or a simple path?

- Database theory concentrates on the former (why?)
- Graph DB engines implement the latter (why?)

Our algorithm evaluates RPQs under arbitrary path semantics

But, what is the semantics?

Is there a path or a simple path?

- Database theory concentrates on the former (why?)
- Graph DB engines implement the latter (why?)

Our algorithm evaluates RPQs under arbitrary path semantics

Is it possible to use it under a simple path semantics?

RPQs under simple paths

Is there a simple path whose label satisfies a regex?

- This problem is NP-complete even if the regex is fixed!
- High complexity only dependent on the size of the data

RPQs under simple paths

Is there a simple path whose label satisfies a regex?

- This problem is NP-complete even if the regex is fixed!
- High complexity only dependent on the size of the data

RPQs under simple paths

Is there a simple path whose label satisfies a regex?

- This problem is NP-complete even if the regex is fixed!
- High complexity only dependent on the size of the data

RPQs under simple paths

Is there a simple path whose label satisfies a regex?

- This problem is NP-complete even if the regex is fixed!
- High complexity only dependent on the size of the data

Example: Consider the RPQ $(aa)^*$

1. It asks whether there is a simple path of even length from x to y
2. This problem is NP-complete

Adding RPQs to graph patterns

Give rise to the class of conjunctive RPQs

- Has received considerable attention in theory
- (Essentially) unexplored from a practical point of view
- Challenging because of matching and RPQ evaluation

Adding RPQs to graph patterns

Give rise to the class of conjunctive RPQs

- Has received considerable attention in theory
- (Essentially) unexplored from a practical point of view
- Challenging because of matching and RPQ evaluation

Adding RPQs to graph patterns

Give rise to the class of conjunctive RPQs

- Has received considerable attention in theory
- (Essentially) unexplored from a practical point of view
- Challenging because of matching and RPQ evaluation

Adding RPQs to graph patterns

Give rise to the class of conjunctive RPQs

- Has received considerable attention in theory
- (Essentially) unexplored from a practical point of view
- Challenging because of matching and RPQ evaluation

Are paths the only form of navigation?

Are paths the only form of navigation?

No! We can allow stronger forms of recursion (DB theory, DL, MC):

- Navigate with branching
  (nested regexs, similar evaluation to RPQs)
- Allow transitive closure on top of conjunctive RPQs
  (regular queries, harder to evaluate)
- Allow arbitrary recursion
  (datalog, very hard to evaluate, non-parallelizable)

Are paths the only form of navigation?

No! We can allow stronger forms of recursion (DB theory, DL, MC):

- Navigate with branching
  (nested regexs, similar evaluation to RPQs)
- Allow transitive closure on top of conjunctive RPQs
  (regular queries, harder to evaluate)
- Allow arbitrary recursion
  (datalog, very hard to evaluate, non-parallelizable)

Are paths the only form of navigation?

No! We can allow stronger forms of recursion (DB theory, DL, MC):

- Navigate with branching
  (nested regexs, similar evaluation to RPQs)
- Allow transitive closure on top of conjunctive RPQs
  (regular queries, harder to evaluate)
- Allow arbitrary recursion
  (datalog, very hard to evaluate, non-parallelizable)

Are paths the only form of navigation?

No! We can allow stronger forms of recursion (DB theory, DL, MC):

- Navigate with branching
  (nested regexs, similar evaluation to RPQs)
- Allow transitive closure on top of conjunctive RPQs
  (regular queries, harder to evaluate)
- Allow arbitrary recursion
  (datalog, very hard to evaluate, non-parallelizable)

Conclusions (really, questions)

- Are RPQs useful in practice?
- Can they be implemented?
- Under which semantics?
- What about conjunctive RPQs?
- Or even stronger forms of recursion?

Conclusions (really, questions)

- Are RPQs useful in practice?
- Can they be implemented?
- Under which semantics?
- What about conjunctive RPQs?
- Or even stronger forms of recursion?

Conclusions (really, questions)

- Are RPQs useful in practice?
- Can they be implemented?
- Under which semantics?
- What about conjunctive RPQs?
- Or even stronger forms of recursion?

Conclusions (really, questions)

- Are RPQs useful in practice?
- Can they be implemented?
- Under which semantics?
- What about conjunctive RPQs?
- Or even stronger forms of recursion?

Conclusions (really, questions)

- Are RPQs useful in practice?
- Can they be implemented?
- Under which semantics?
- What about conjunctive RPQs?
- Or even stronger forms of recursion?

Conclusions (really, questions)

- Are RPQs useful in practice?
- Can they be implemented?
- Under which semantics?
- What about conjunctive RPQs?
- Or even stronger forms of recursion?

RETURNING PATHS

Paths in the output

Query languages such as Cypher & PGQL allow:

- Return one path
- Return one shortest path (DFS)
- Return all paths
- Return all shortest paths

Paths in the output

Query languages such as Cypher & PGQL allow:

- Return one path
- Return one shortest path (DFS)
- Return all paths
- Return all shortest paths

But, what does it mean to return all paths?

... there can be infinitely many

Paths in the output

Query languages such as Cypher & PGQL allow:

- Return one path
- Return one shortest path (DFS)
- Return all paths
- Return all shortest paths

But, what does it mean to return all paths?

... there can be infinitely many

Systems return simple paths

... but there can be exponentially many

Even more interesting for RPQs

- Return a shortest path whose label satisfies a regex
- And all shortest paths
- Return all paths whose label satisfies a regex
- And all paths

A solution from database theory

Instead of returning all paths ...

return a *compact* representation of them

A solution from database theory

Instead of returning all paths ...

return a *compact* representation of them

Compact representation: A property graph with all paths in the output

- Can be constructed efficiently (in data) for arbitrary paths
- Impossible for simple paths

A solution from database theory

Instead of returning all paths ...

return a *compact* representation of them

Compact representation: A property graph with all paths in the output

- Can be constructed efficiently (in data) for arbitrary paths
- Impossible for simple paths

A solution from database theory

Instead of returning all paths ...

return a *compact* representation of them

Compact representation: A property graph with all paths in the output

- Can be constructed efficiently (in data) for arbitrary paths
- Impossible for simple paths

Conclusions

- Returning paths is difficult under all interpretations
- "All paths" can be compactly represented, but simple paths cannot
- The right semantics still needs to be settled

UNGROUPING

Paths can appear in the output

Paths can appear in the output

We can *ungroup* them

- List the nodes that appear in them

Paths can appear in the output

We can *ungroup* them

- List the nodes that appear in them

We can check if a path visits all nodes

- Travelling salesman problem!

Paths can appear in the output

We can *ungroup* them

- List the nodes that appear in them

We can check if a path visits all nodes

- Travelling salesman problem!

Paths can appear in the output

We can *ungroup* them

- List the nodes that appear in them

We can check if a path visits all nodes

- Travelling salesman problem!

Interaction with other operators expresses even more complex properties

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

Complexity of evaluation is astronomical (in data)

- (Severely) restricting the language leads to efficiency

A query language for paths

- Variables for paths and nodes
- Can check if a node belongs to a path
- Can check if the label of a path satisfies a regex
- Closure under quantification over nodes and paths

Complexity of evaluation is astronomical (in data)

- (Severely) restricting the language leads to efficiency

Question

- What ungropuing can do and should do?

FINAL THOUGHTS

Final thoughts

- Exciting times for studying graph DBs (theory/practice)
- Lots of fine tuning needed
- Some issues still unexplored:
  - Comparing paths
  - Ranking of answers
  - Constraints

MANY THANKS

Bibliography

Surveys:

- P. Wood. Query languages for graph databases. SIGMOD Record, 2012.
- P. Barceló. Querying graph databases. PODS, 2013.

Regular path queries:

- A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. SIAM J. on Comp, 1995.
- D. Calvanese, G. de Giacomo, M. Lenzerini, M. Vardi. Reasoning on regular path queries. SIGMOD Record, 2003.

Conjunctive regular path queries:

- D. Calvanese, G. de Giacomo, M. Lenzerini, M. Vardi. Containment of conjunctive regular path queries with inverse. KR, 2000.

Bibliography

Queries with more expressive recursion:

- P. Barceló, J. Pérez, J. Reutter. Relative expressiveness of nested regular expressions. AMW, 2012.
- J. Reutter, M. Romero, M. Vardi. Regular queries on graph databases. ICDT, 2015.

Queries that compare nodes:

- L. Libkin, D,. Vrgoc. Regular path queries on graphs with data. ICDT, 2012.
- G. Fletcher, M. Gyssens, D. Leinders, et al. Relative expressive power of navigational querying on graphs. ICDT, 2011.
- P. Barceló, G. Fontaine, A. W. Lin. Expressive path queries over graphs with data. LMCS, 2015.

Queries with paths as first-class citizens and negation:

- P. Barceló, L. Libkin, A. W. Lin, P. Wood. Expressive path queries over graph-structured data. ACM TODS, 2012.