Graphs and paths: Finding connections and patterns in data

Peter Wood

Department of Computer Science and Information Systems

Birkbeck, University of London

`ptw@dcs.bbk.ac.uk`

Contents

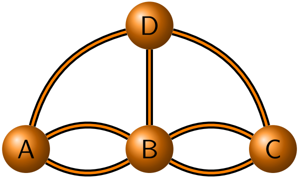
- Introduction
- The complexity of querying graphs (1987–1995)
- Optimising queries on trees (1999–2007)
- Adding flexibility to graph queries (2006–
- Ongoing and future work

Contents

Introduction

- What is a graph and what are they used for?
- What is a query language?
- What are some of the problems we study?

What is a graph?

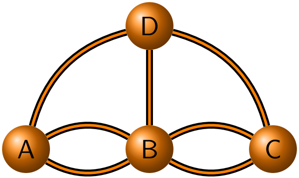


A graph is an abstract representation of some real-world data, where the data consists of connections between entities:

- the entities are denoted by nodes
- the connections are denoted by edges

Graphs have been studied since the 18th century

What is a graph?

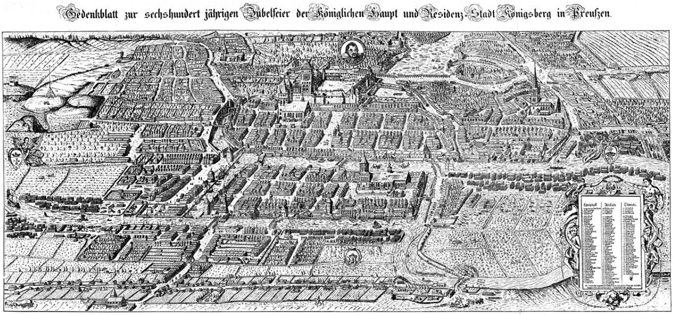


A graph is an abstract representation of some real-world data, where the data consists of connections between entities:

- the entities are denoted by nodes
- the connections are denoted by edges

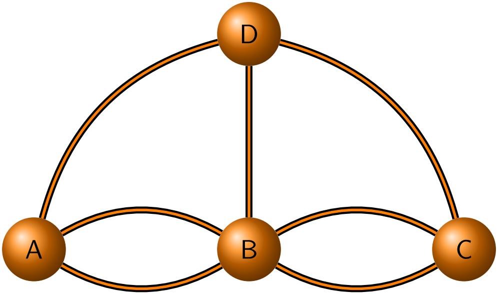
Graphs have been studied since the 18th century

Bridges of Königsberg

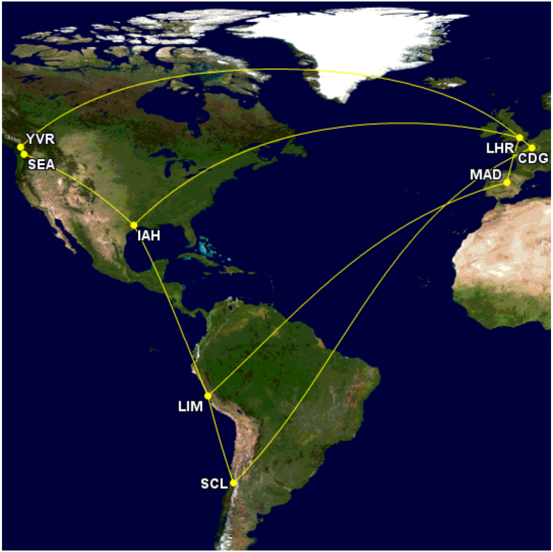


- The city of Königsberg straddled a river containing 2 islands.
- The islands and mainland were connected by 7 bridges.
- Was there a walk that crossed each bridge once and only once?
- In 1736 Leonhard Euler proved that such a walk did not exist.

Bridges of Königsberg as a graph

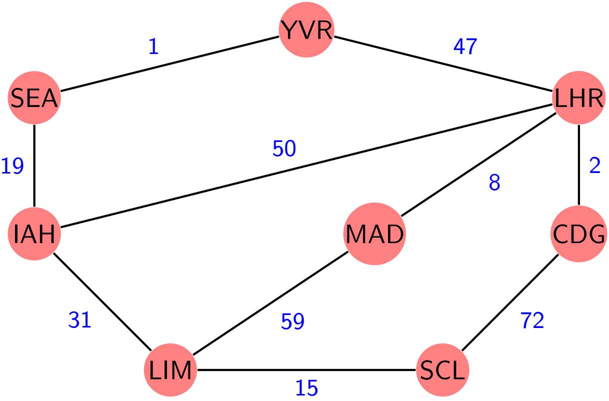


Airline flights



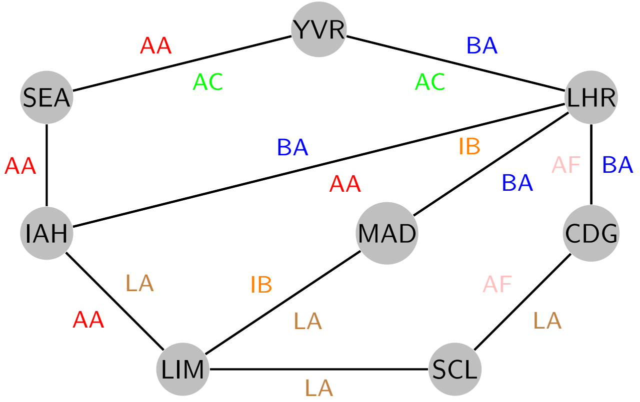
Airline flights as a graph

A graph of airports and flight distances (in hundreds of miles):



Airlines connecting airports

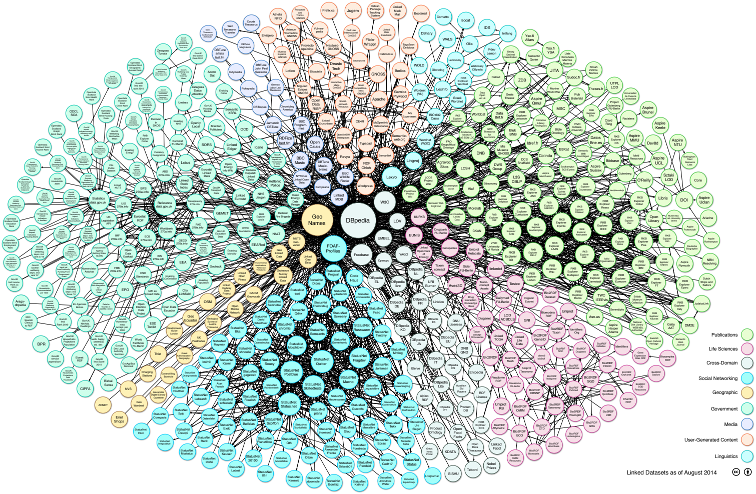
A graph of airports and airlines flying between them:



Large graphs

- the previous examples were of “toy” graphs
- real-life graphs can be very large
- examples include
  - Facebook: 1.65 billion users (nodes) and 280 billion connections (edges)
  - Linked Open Data cloud: 295 datasets, with 31 billion facts

Linked data graph



Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak.

http://lod-cloud.net/

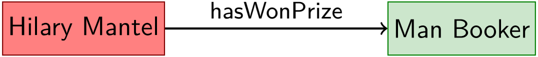
Linked Open Data

Linked Open Data stores facts such as

> “Hilary Mantel won the Man Booker Prize"

as *triples* of the form subject — predicate (or property) — object

e.g. (Hilary Mantel , hasWonPrize, Man Booker) or



This language is called RDF (Resource Description Framework)

What are graphs used for?

- social networks
- knowledge representation (e.g. semantic web)
- transportation and other networks (e.g WWW)
- geographical information
- (hyper)document structure
- semantic associations in criminal investigations
- bibliographic citation analysis
- pathways in biological processes
- computer program analysis
- workflow systems
- data provenance
- ...

Query languages

A query language is a declarative language in which to express an information request from a data set.

- "declarative" means that we shouldn't need to specify precisely *how* the information is to be retrieved
- that is left up to the (database) system to work out

e.g. What is the length of the shortest route (path) from LHR to LIM

Query languages

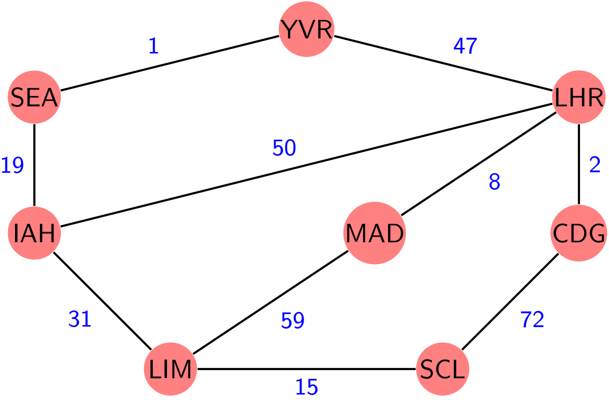
A query language is a declarative language in which to express an information request from a data set.

- “declarative” means that we shouldn’t need to specify precisely *how* the information is to be retrieved
- that is left up to the (database) system to work out

e.g. What is the length of the shortest route (path) from LHR to LIM

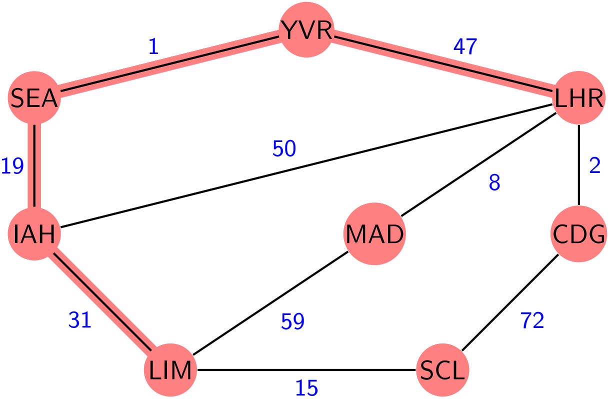
Airline flights as a graph

A graph of airports and flight distances (in hundreds of miles):



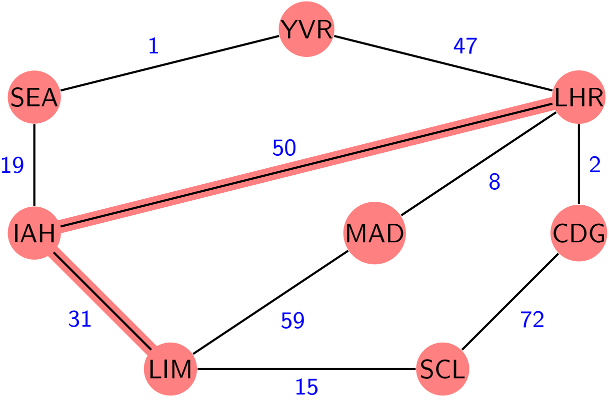
Airline flights as a graph

A graph of airports and flight distances (in hundreds of miles):



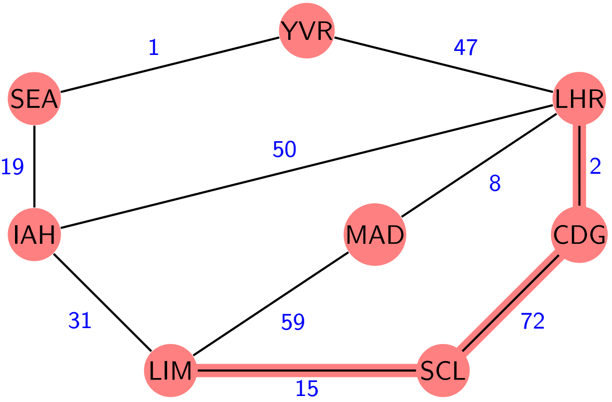
Airline flights as a graph

A graph of airports and flight distances (in hundreds of miles):



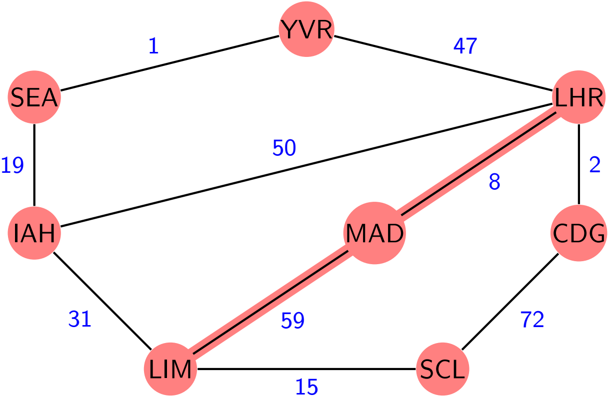
Airline flights as a graph

A graph of airports and flight distances (in hundreds of miles):



Airline flights as a graph

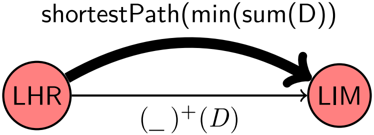
A graph of airports and flight distances (in hundreds of miles):



Shortest path algorithm

```
Function Dijkstra(Graph, source)
dist[LHR] $\leftarrow$ 0
create node set Q
foreach node $v$ in Graph do
    if $v \neq$ source then  dist[v] $\leftarrow$ INFINITY
    $Q$.add_with_priority($v$, dist[$v$])
while Q is not empty do
    $u \leftarrow$ Q.extract_min()
    foreach neighbour v of u do
        $alt \leftarrow$ dist[$u$] + length($u$, $v$)
        if $alt <$ dist[$v$] then
            dist[$v$] $\leftarrow alt$
            Q.decrease_priority($v$, $alt$)
return dist[]
```

Shortest path as a query



shortestPath(min(sum(D)))

LHR

LIM

$(\_)^{+}(D)$

- variable D collects up the distance values
- sum adds up the distances along a path
- min asks for the minimum summed distance

Problems to study

3 main topics of investigation:

- what are appropriate query language constructs?
- how efficiently can queries be evaluated?
- how can query evaluation be optimised?

Measures of efficiency

- we can run queries to see how fast they execute
- we can study their computational complexity
  - polynomial-time algorithms ☺
  - NP-complete (or worse) problems — intractable ☹
    - we can try to find polynomial-time subclasses (occurring in practice)
    - we can try to find approximation algorithms or heuristics
    - we can claim that the problem size is small enough

Measures of efficiency

- we can run queries to see how fast they execute
- we can study their computational complexity
  - polynomial-time algorithms 🙂
  - NP-complete (or worse) problems — intractable ☹
    - we can try to find polynomial-time subclasses (occurring in practice)
    - we can try to find approximation algorithms or heuristics
    - we can claim that the problem size is small enough

Measures of efficiency

- we can run queries to see how fast they execute
- we can study their computational complexity
  - polynomial-time algorithms 🙂
  - NP-complete (or worse) problems — intractable 🙁
    - we can try to find polynomial-time subclasses (occurring in practice)
    - we can try to find approximation algorithms or heuristics
    - we can claim that the problem size is small enough

Contents

- Introduction
- The complexity of querying graphs (1987–1995)
- Optimising queries on trees (1999–2007)
- Adding flexibility to graph queries (2006–
- Ongoing and future work

Many years ago . . .

- my PhD was about "Queries on Graphs"
- one contribution was the use of regular expressions to specify the paths of interest
- also contributions in terms of complexity of query evaluation

25 years later

- regular expressions were added to SPARQL 1.1, the standard query language for RDF

Many years ago . . .

- my PhD was about "Queries on Graphs"
- one contribution was the use of regular expressions to specify the paths of interest
- also contributions in terms of complexity of query evaluation

25 years later

- regular expressions were added to SPARQL 1.1, the standard query language for RDF

Regular expressions

Used to express *patterns* of interest, using 3 (or more) operators:

- "choice" (|): a parent is a father or mother

  (father | mother)

- "sequence" (·): a grandparent is a parent of a parent

  (parent · parent)

- "repetition" (∗): an ancestor is a parent of a parent of a . . .

  (parent)∗

These can be combined: so (mother | father)∗ specifies any sequence of mother and/or father relationships.

Regular expressions

Used to express *patterns* of interest, using 3 (or more) operators:

- "choice" ($|$): a parent is a father or mother

  (father $|$ mother)

- "sequence" ($\cdot$): a grandparent is a parent of a parent

  (parent $\cdot$ parent)

- "repetition" ($*$): an ancestor is a parent of a parent of a ...

  (parent)$*$

These can be combined: so (mother $|$ father)$*$ specifies any sequence of mother and/or father relationships.

Regular expressions

Used to express *patterns* of interest, using 3 (or more) operators:

- “choice” ($|$): a parent is a father or mother

  (father $|$ mother)

- “sequence” ($\cdot$): a grandparent is a parent of a parent

  (parent $\cdot$ parent)

- “repetition” ($*$): an ancestor is a parent of a parent of a . . .

  (parent)$*$

These can be combined: so (mother $|$ father)$*$ specifies any sequence of mother and/or father relationships.

Regular expressions

Used to express *patterns* of interest, using 3 (or more) operators:

- "choice" ($|$): a parent is a father or mother

  (father | mother)

- "sequence" ($\cdot$): a grandparent is a parent of a parent

  (parent $\cdot$ parent)

- "repetition" ($*$): an ancestor is a parent of a parent of a . . .

  (parent)$*$

These can be combined: so (mother | father)$*$ specifies any sequence of mother and/or father relationships.

Regular expressions

Used to express *patterns* of interest, using 3 (or more) operators:

- "choice" ($|$): a parent is a father or mother

  (father $|$ mother)

- "sequence" ($\cdot$): a grandparent is a parent of a parent

  (parent $\cdot$ parent)

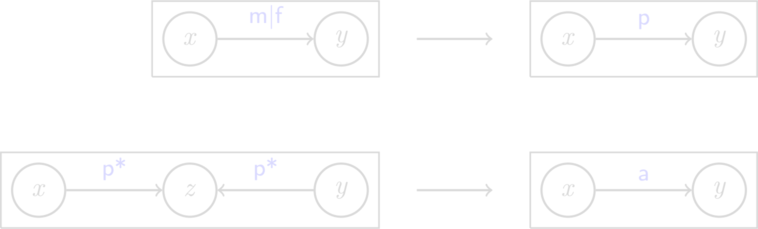
- "repetition" ($*$): an ancestor is a parent of a parent of a ...

  (parent)$*$

These can be combined: so (mother $|$ father)$*$ specifies any sequence of mother and/or father relationships.

Example of a query

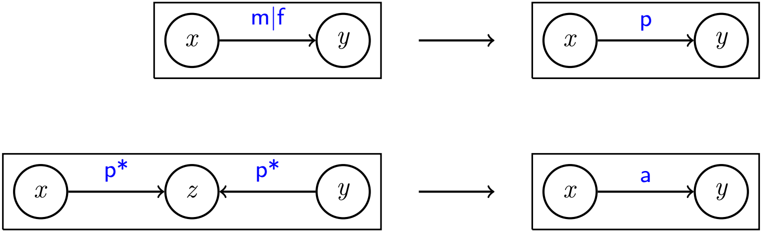
- given a graph
  - nodes representing people
  - edges labelled with m (for motherOf) or f (for fatherOf)
- following query finds parents followed by pairs of people who have a common ancestor



x, y and z are *variables*, for matching nodes in the graph

Example of a query

- given a graph
  - nodes representing people
  - edges labelled with m (for motherOf) or f (for fatherOf)
- following query finds parents followed by pairs of people who have a common ancestor



x, y and z are *variables*, for matching nodes in the graph

Regular simple path queries

- a path p is simple if no node is repeated on p
- REGULAR SIMPLE PATH PROBLEM

  Given a graph, a pair of nodes x and y and a regular expression r, is there a simple path from x to y satisfying r?
- REGULAR SIMPLE PATH PROBLEM is NP-complete, even for fixed expressions ☹
- the complexity is in the size of the *graph*

Regular simple path queries

- a path p is simple if no node is repeated on p
- Regular Simple Path Problem

  Given a graph, a pair of nodes x and y and a regular expression r, is there a simple path from x to y satisfying r?
- Regular Simple Path Problem is NP-complete, even for fixed expressions ☹
- the complexity is in the size of the *graph*

Regular simple path queries

- a path p is simple if no node is repeated on p
- Regular Simple Path Problem

  Given a graph, a pair of nodes x and y and a regular expression r, is there a simple path from x to y satisfying r?
- Regular Simple Path Problem is NP-complete, even for fixed expressions 🙁
- the complexity is in the size of the *graph*

Regular simple path problem

- solvable in polynomial time when restricted to regular expressions closed under abbreviations (or deletions) 🙂
- if we take a sequence satisfying regular expression r and delete some labels from the sequence, we get another sequence satisfying r
- many commonly used regular expressions, such as p* and (m|f)* are closed under abbreviations

Contents

- Introduction
- The complexity of querying graphs (1987–1995)
- Optimising queries on trees (1999–2007)
- Adding flexibility to graph queries (2006–
- Ongoing and future work

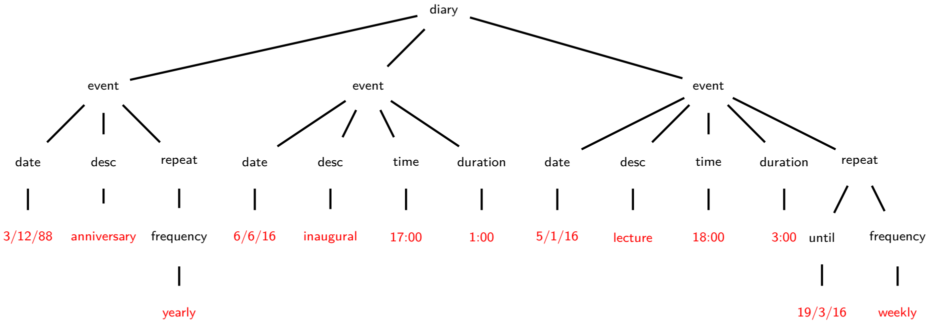
Trees

- trees are commonly used for representing data
- one language for trees is called XML (eXtensible Modelling Language)
- XML is used for data exchange between systems/applications
- e.g., HESA (Higher Education Statistics Agency) requires student data to be submitted in XML

Calendar data in XML

```
<diary>
  <event>
    <date>3/12/88</date> <description>anniversary</description>
    <repeat><frequency>yearly</frequency></repeat>
  </event>
  <event>
    <date>6/6/16</date> <description>inaugural</description>
    <time>17:00</time> <duration>1:00</duration>
  </event>
  <event>
    <date>5/1/2016</date> <description>lecture</description>
    <time>18:00</time> <duration>3:00</duration>
    <repeat><until>19/3/2016</until><frequency>weekly</frequency></repeat>
  </event>
</diary>
```

Calendar data as a tree



A query language for XML

- XPath (XML Path language) is a query language for XML
- it is part of many other languages for processing XML
- it makes sense to study optimisation of XPath queries
- particularly in the presence of schema information
- a schema is a set of constraints that restricts the permissable relationships between data items
- one way to define schemas for XML is to use Document Type Definitions (DTDs)

A query language for XML

- XPath (XML Path language) is a query language for XML
- it is part of many other languages for processing XML
- it makes sense to study optimisation of XPath queries
- particularly in the presence of schema information
- a schema is a set of constraints that restricts the permissable relationships between data items
- one way to define schemas for XML is to use Document Type Definitions (DTDs)

DTD example

Consider an XML DTD for our diary application:

$$\begin{array}{lcl} \text{diary} & \rightarrow & (\text{event})^* \\ \text{event} & \rightarrow & (\text{date} \cdot \text{description} \cdot (\text{time} \cdot \text{duration})?, \text{repeat}?) \\ \text{repeat} & \rightarrow & ((\text{until} \mid \text{occurrences})? \cdot \text{frequency}) \end{array}$$

- note the use of regular expressions above
- ? means that what precedes it is *optional*
- some constraints specified by the DTD are:
  - every event must have a date
  - every event with a time must have a duration
  - every event has at most one description

DTD example

Consider an XML DTD for our diary application:

$$\begin{array}{lcl}
\text{diary} & \rightarrow & (\text{event})^* \\
\text{event} & \rightarrow & (\text{date} \cdot \text{description} \cdot (\text{time} \cdot \text{duration})?, \text{repeat}?) \\
\text{repeat} & \rightarrow & ((\text{until} \mid \text{occurrences})? \cdot \text{frequency})
\end{array}$$

- note the use of regular expressions above
- ? means that what precedes it is *optional*
- some constraints specified by the DTD are:
  - every event must have a date
  - every event with a time must have a duration
  - every event has at most one description

Query satisfiability

- an XPath query may be unsatisfiable (with respect to a DTD)
- this means that the answer to the query will always be empty
- e.g., asking for diary events which repeat **both** until some date **and** some number of occurrences
- checking satisfiability first can yield savings in overall query processing time

XPath satisfiability results

(joint work with PhD student Manizheh Montazerian)

- we showed that checking satisfiability, even for simple XPath fragments, is NP-complete 🙁
- we also examined several real-world DTDs and discovered a new property, called covering, which most of them preserved
- we showed that query satisfiability for a fragment of XPath can be tested in polynomial time when the underlying DTD is covering 🙂

Contents

- Introduction
- The complexity of querying graphs (1987–1995)
- Optimising queries on trees (1999–2007)
- Adding flexibility to graph queries (2006–
- Ongoing and future work

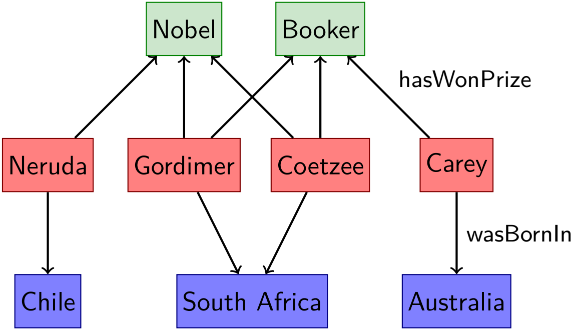
Flexible querying

(joint work with Alex Poulovassilis, Andrea Calì, Carlos Hurtado, Petra Selmer, Riccardo Frosini)

- with LOD, many heterogeneous data sets are available
- the standard query language is called SPARQL
- regular expressions (property paths in SPARQL 1.1 (2013)) already allow for some flexibility
- but when users don't know the vocabulary or structure, they may need help
- in particular, they may pose unsatisfiable queries

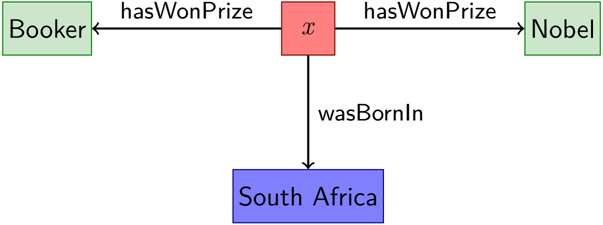
Example LOD graph

Graph of authors, prizes they have won, and countries where they were born:



Example SPARQL query

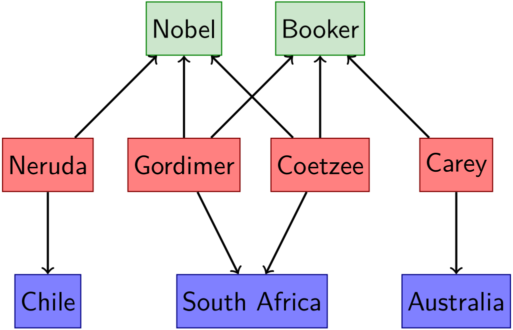
Which authors born in South Africa have won both the Nobel Prize in Literature and the Man Booker prize?



x is a *variable*

Matching answers

Two matchings for variable x: Gordimer and Coetzee


Nobel

Booker

Neruda

Gordimer

Coetzee

Carey

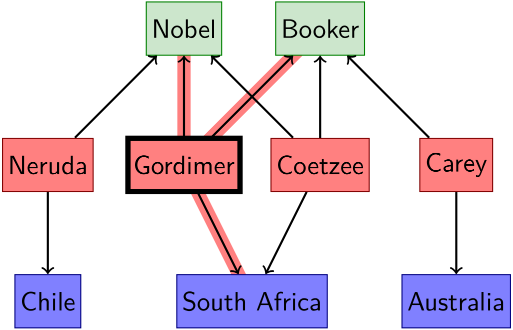
Chile

South Africa

Australia

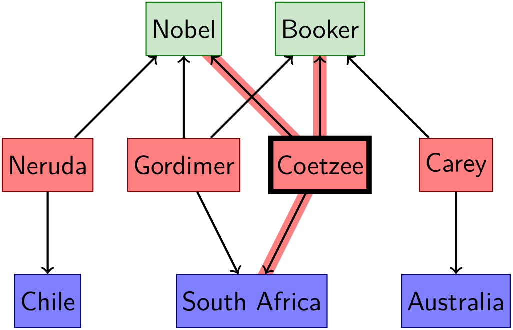
Matching answers

Two matchings for variable x: Gordimer and Coetzee



Matching answers

Two matchings for variable x: Gordimer and Coetzee



- in the *real* data set, wasBornIn connects a person to a *city*, not a country
- so the previous query will return no answers
- a city is connected to a country by an edge labelled with isLocatedIn
- so the correct regular expression to connect x to South Africa is: wasBornIn · isLocatedIn
- the query system can automatically make changes to a query so as to help the user find relevant information

Approximate matching

- the user's original query is modified by applying edit operations to the regular expressions, e.g.,
  - insertions
  - deletions
  - substitutions
- so isLocatedIn would be automatically inserted after wasBornIn
- such edit operations have also been used for DNA sequence alignment
- each operation may have a different cost associated with it
- answers to queries are returned in ranked order, in increasing total cost from the original query

Approximate matching

- the user's original query is modified by applying edit operations to the regular expressions, e.g.,
  - insertions
  - deletions
  - substitutions
- so isLocatedIn would be automatically inserted after wasBornIn
- such edit operations have also been used for DNA sequence alignment
- each operation may have a different cost associated with it
- answers to queries are returned in ranked order, in increasing total cost from the original query

Optimisation

- applying edit operations generates many additional queries
- so we have been investigating ways to speed up query execution
- one is to construct an approximate summary schema from the data
- this allows the system to detect and ignore unsatisfiable queries

Schema summary optimisation

- basic idea is to record all the path sequences up to a certain maximum length that actually appear in the data
- choose a small maximum length, e.g. 2
- only an approximation: if we discover two paths $\mathsf{a} \cdot \mathsf{b}$ and $\mathsf{b} \cdot \mathsf{c}$ in the data, the summary records that $\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{c}$ is a possible path, even if it doesn't actually appear in the data
- nevertheless, if a path does not appear in the summary, then it does not appear in the data
- this allows the system to check for unsatisfiable queries
- and save time by not executing them

Schema summary optimisation

- basic idea is to record all the path sequences up to a certain maximum length that actually appear in the data
- choose a small maximum length, e.g. 2
- only an approximation: if we discover two paths $a \cdot b$ and $b \cdot c$ in the data, the summary records that $a \cdot b \cdot c$ is a possible path, even if it doesn't actually appear in the data
- nevertheless, if a path does not appear in the summary, then it does not appear in the data
- this allows the system to check for unsatisfiable queries
- and save time by not executing them

Sample schema summary results

- for one query with a maximum edit cost set to 3,

  112 additional queries were generated via edit operations
- using a schema summary on a data set of 6.7M facts,

  59 of these queries were unsatisfiable
- without the summary, the queries did not complete execution within 8 hours
- with a schema summary of size 2, they completed in under 4 seconds

Sample schema summary results

- for one query with a maximum edit cost set to 3,
  112 additional queries were generated via edit operations
- using a schema summary on a data set of 6.7M facts,
  59 of these queries were unsatisfiable
- without the summary, the queries did not complete execution within 8 hours
- with a schema summary of size 2, they completed in under 4 seconds

Contents

Ongoing and future work

- further investigation of suitable query constructs, particularly for social network applications
- returning recommended paths to users, rather than simply nodes
- provide explanations to users about how answers to queries were derived
- other methods to improve query execution time
- use mappings between LOD data sets to allow for automatic rewriting of queries in distributed scenarios