Query Languages for Graph Databases

Peter T. Wood

School of Computer Science and Information Systems
Birkbeck, University of London
ptw@dcs.bbk.ac.uk

Birkbeck UNIVERSITY OF LONDON
Educating busy Londoners

Third Alberto Mendelzon International Workshop on Foundations of Data Management

Outline

Outline

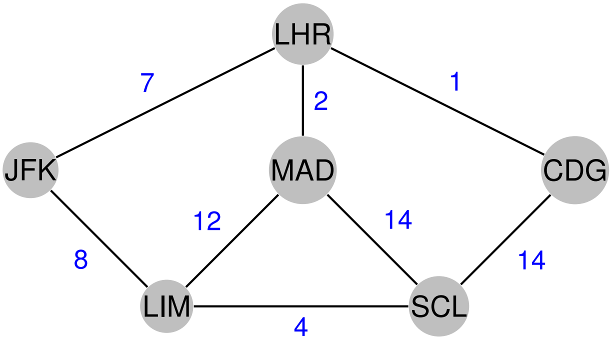
Motivation

Graphs are widely used for representing data

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

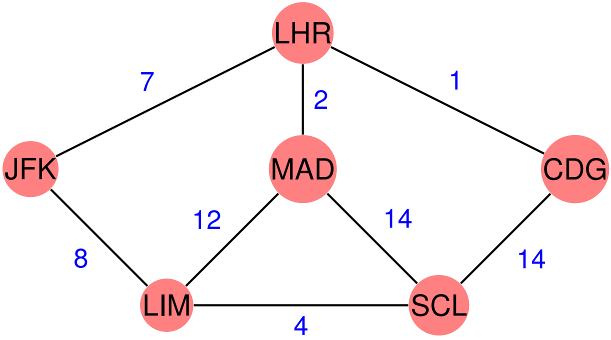
Example

A graph of cities and flight durations:



Example

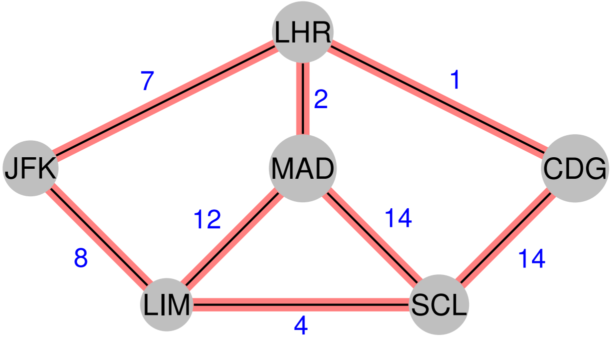
A graph of cities and flight durations:



Nodes

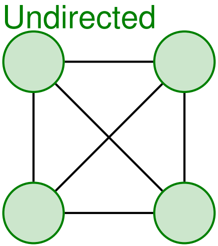
Example

A graph of cities and flight durations:

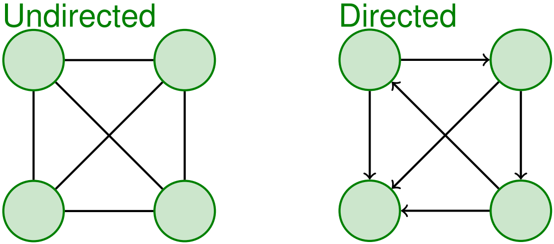


Edges

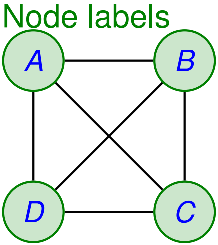
Types of Edges



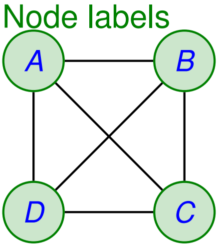
Types of Edges

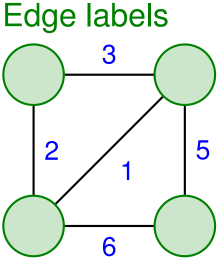


Types of labels

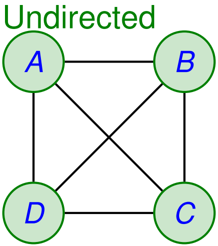


Types of labels

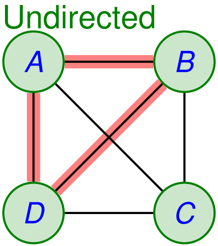




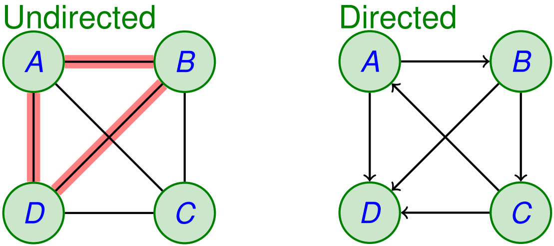
Cyclic graphs



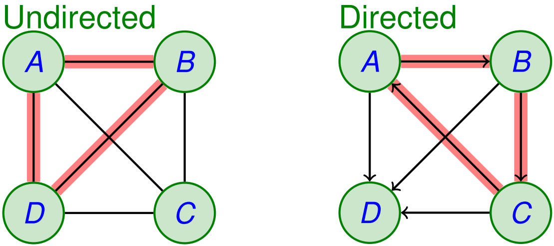
Cyclic graphs



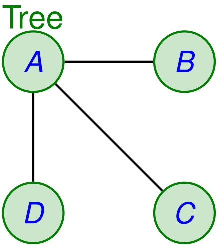
Cyclic graphs



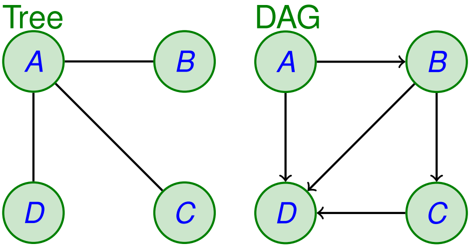
Cyclic graphs



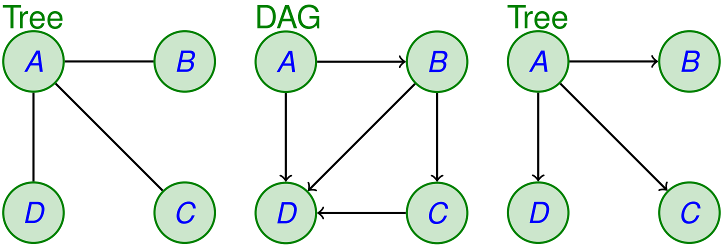
Acyclic graphs



Acyclic graphs



Acyclic graphs



Formal graph definition

For our purposes:

- database comprises a single labelled (multi-)graph G
- (finite) set of nodes N with identifiers drawn from an infinite vocabulary V
- (finite) set of (directed) edges E
- incidence function $\phi : E \mapsto N \times N$ (allows multi-edges)
- edge labelling function $\lambda : E \mapsto \Sigma$
- Σ is a finite alphabet

So $G = (N, E, V, \Sigma, \phi, \lambda)$

Outline

Background

- Graphs
- Personal interest
- Uses of graphs

Graph models and query languages

- G, G^+ and Graphlog
- Lore/Lorel
- YAGO/NAGA
- Other models and languages

Query Functionality

- Overview
- Graph pattern matching
- Path finding
- Edge and path variables
- Aggregation
- Approximate matching and ranking

Many years ago . . .

- PhD on "Queries on Graphs" (1988)
- supervised by Alberto Mendelzon

Many years ago . . .

- PhD on “Queries on Graphs” (1988)
- supervised by Alberto Mendelzon

More recently

- querying RDF (allowing for query relaxation and ranking)
- ranking approximate answers to semantic web queries
- investigating operators for finding/manipulating paths
- . . . with Pablo Barcelo and Carlos Hurtado (Chile) and Alex Poulovassilis (Birkbeck)

Outline

Transportation and other networks

- airline, train, bus … networks
- communication networks
- planning networks—single source and sink, acyclic

Transportation and other networks

- airline, train, bus . . . networks
- communication networks
- planning networks—single source and sink, acyclic

Typical queries:

- reachability: can I get from a to b?
- shortest path: find the quickest/shortest route from a to b
- reliability/capacity of paths
- critical path

Knowledge representation

- semantic networks
- conceptual graphs
- RDF/S, OWL
- ontologies
- taxonomies
- ...

Knowledge representation

- semantic networks
- conceptual graphs
- RDF/S, OWL
- ontologies
- taxonomies
- ...

Typical queries:

- instance and subclass relationships
- finding connections between entities
- ...

Program/workflow analysis

- nodes are program points or agents/products
- edges are program or workflow steps
- often single source and sink nodes
- also data provenance applications

Program/workflow analysis

- nodes are program points or agents/products
- edges are program or workflow steps
- often single source and sink nodes
- also data provenance applications

Typical queries:

- reachability of code
- variables used before defined
- deadlock/livelock
- what agents/processes/products were involved in producing something

Biological applications

- metabolic pathways
- gene regulatory networks
- protein interaction networks
- ...

Biological applications

- metabolic pathways
- gene regulatory networks
- protein interaction networks
- ...

Typical queries include:

- path existence
- subgraph isomorphism
- k-shortest paths
- neighbourhood queries
- approximate matching
- ...

(see https://hpcrd.lbl.gov/staff/olken/graphdm/graphdm.htm)

Outline

Background
- Graphs
- Personal interest
- Uses of graphs

Graph models and query languages
- G, G^+ and Graphlog
- Lore/Lorel
- YAGO/NAGA
- Other models and languages

Query Functionality
- Overview
- Graph pattern matching
- Path finding
- Edge and path variables
- Aggregation
- Approximate matching and ranking

G, G^+ and Graphlog

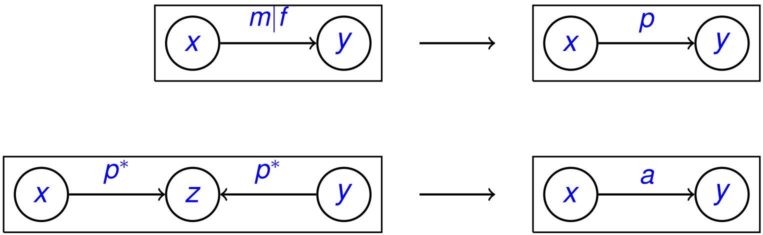
- from the 1980s [Cruz, Mendelzon and Wood, 1987] [Cruz, Mendelzon and Wood, 1988] [Consens and Mendelzon, 1989]
- developed at University of Toronto
- data model is a labelled, directed graph
- in G and G^+, query is a set of pairs of pattern graphs and summary graphs
- pattern graph nodes are labelled with variables or constants
- pattern graph edges are labelled with regular expressions over edge labels and variables
- Graphlog adds edge inversion, negation, distinguished edge and different semantics

G, G^+ Example

- given a graph
  - nodes representing people
  - edges labelled with *m* (for *motherOf*) and *f* (for *fatherOf*)

G, G^+ Example

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



Outline

Lore/Lorel

- from the 1990s [Abiteboul et al., 1997]
- developed at Stanford
- Lore: Lightweight Object Repository
- Lorel: Lore query language
- for semistructured data
  - no predefined schema
  - may be heterogeneous
- uses Object Exchange Model (OEM)
- Lore/Lorel can be viewed as extension of ODMG model/OQL

Lore model

- data model is graph with two types of nodes
  - complex objects
  - atomic objects (values) with no outgoing edges
- each node has a unique oid
- each edge is labelled with a string
- graph has a number of named nodes (entry points)
- every node must be reachable from a named node

Lorel example

- graph representing a restaurant guide
- find addresses of restaurants with a given zipcode

  select *Guide*.*restaurant*.*address*
  where *Guide*.*restaurant*.*address*.*zipcode* = 92310
- *Guide* is a named node
- *restaurant*, *address* and *zipcode* are edge labels

Outline

Background
Graphs
Personal interest
Uses of graphs
Graph models and query languages
G, G^+ and Graphlog
Lore/Lorel
YAGO/NAGA
Other models and languages
Query Functionality
Overview
Graph pattern matching
Path finding
Edge and path variables
Aggregation
Approximate matching and ranking
Summary

YAGO/NAGA

- from the 2000s [Weikum et al., 2009]
- developed at Max Planck Institute for Informatics
- YAGO: Yet Another Great Ontology
- NAGA: Not Another Google Answer
- semantic search engine for web derived knowledge
- combines DB and IR
- 26 relationships between entities derived using information extraction
  - e.g., *isA*, *bornInYear*, *hasWonPrize*, *locatedIn*, . . .

NAGA model

- data model is directed, weighted multigraph
- nodes represent entities
- edges represent relationships
- weights represent confidence of extracted facts
- query is a connected, directed graph
- each edge labelled with a regular expression over edge labels or a variable or *connect* keyword
- answers are ranked by
  - informativeness
  - confidence
  - compactness

NAGA examples

- graph representing information on people and films
- in which films did a governor act?

  X isA governor
  X actedIn Y
  Y isA film
- *X* and *Y* are node variables
- *isA* and *actedIn* are relationships (edge labels)
- what do Albert Einstein and Niels Bohr have in common?

  Albert_Einstein connect Niels_Bohr
- *Albert_Einstein* and *Niels_Bohr* are node labels
- asks for paths connecting nodes—ranked

Outline

Other graph data models and query languages

- Functional Data Model
- Logical Data Model
- O2
- GOOD, GDM
- Strudel and StruQL
- G-BASE, Gram, GraphDB, GRAS
- hypergraphs, hypernode model, hygraphs
- RDF/S and SPARQL

See Survey of Graph Database Models [Angles and Gutierrez, 2008]

Outline

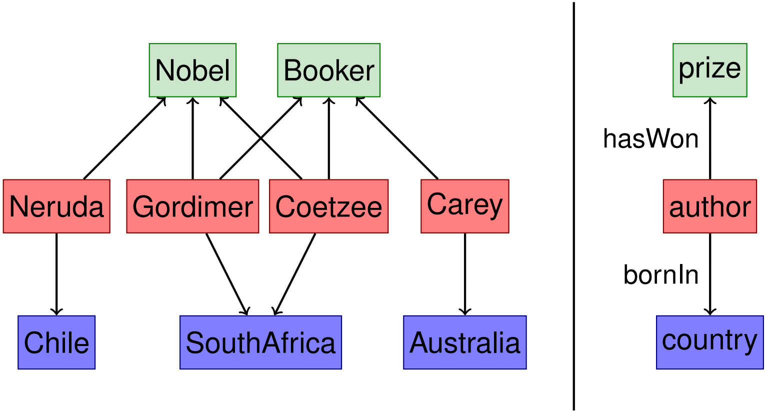
Query functionality

- graph pattern matching
- path finding
- edge label variables
- negation
- path variables
- aggregation
- approximate matching and ranking
- (disjunction)

Outline

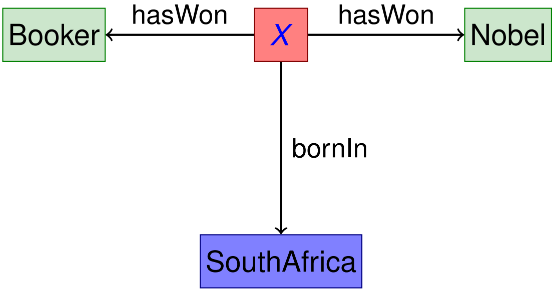
Example graph

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



Example query

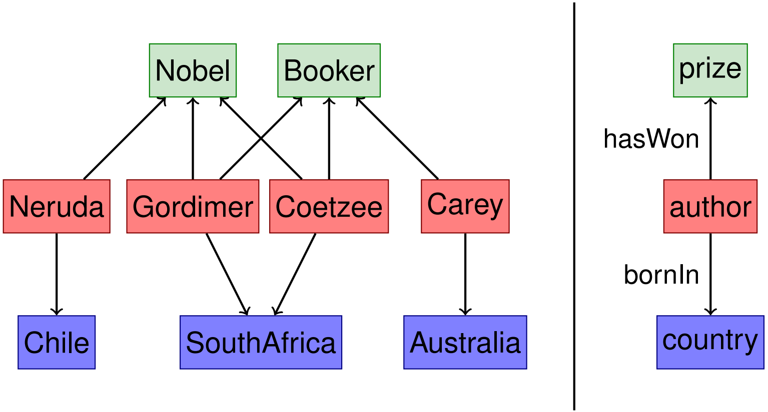
Which authors born in South Africa have won both the Nobel Prize in Literature and the Man Booker prize?



X is a *variable*

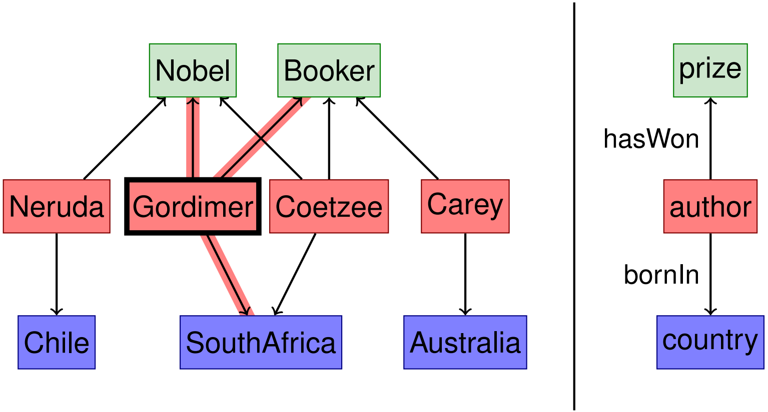
Matching subgraphs

Two matching subgraphs



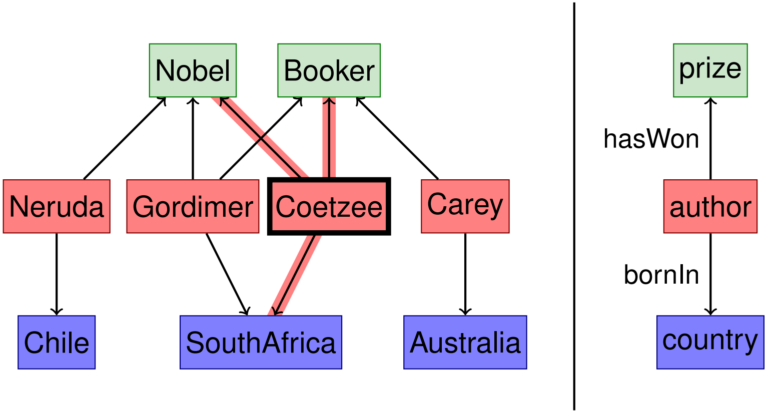
Matching subgraphs

Two matching subgraphs



Matching subgraphs

Two matching subgraphs



Query answers

Depending on the query language and whether the database is a set of graphs or a single graph, answers mights be the

- set of graphs in which a match is found (e.g. biological applications)
- set of matching subgraphs (NAGA)
- set of variable bindings for each variable (most others)

Forms of query expression

- similar to SQL/OQL (Lorel, RQL):

  select X
  from $X.hasWon\ Y$, $X.hasWon\ Z$, $X.bornIn\ W$
  where $Y = Nobel$ and $Z = Booker$ and
  $W = SouthAfrica$

  W, X, Y and Z are variables

- conjunctive query (similar to NAGA and others):

$$\begin{aligned}(X) \quad \leftarrow \quad & (X, hasWon, Nobel), \\ & (X, hasWon, Booker), \\ & (X, bornIn, SouthAfrica)\end{aligned}$$

Query evaluation problem

Given a query expression Q and a graph (database) G, is $Q(G)$ non-empty?

- *Combined complexity*: both Q and G are part of the input
- *Query complexity*: input is Q while G is fixed
- *Data complexity*: input is G while Q is fixed

Often consider data complexity since graphs are assumed to be large while query expressions are assumed to be short

Complexity of query evaluation

For graph pattern matching, the complexity is the same as

- relational conjunctive queries
- subgraph isomorphism

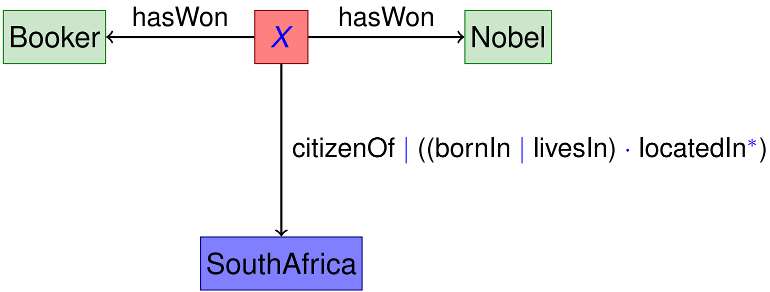
namely

- NP-complete in terms of query and combined complexity
- PTIME in terms of data complexity

Query and combined complexity are in PTIME if the variables in the query satisfy an *acyclicity* condition [Yannakakis, 1981]
But can still be exponential if output all variable bindings

Outline

More flexible matching



South African if a citizen or born or lives in a place located there

Regular expressions

Regular expression over alphabet Σ of edge labels:

- ϵ (empty string) is a regular expression
- any label in Σ is a regular expression
- if r_1 and r_2 are regular expressions, then so are
  - $(r_1|r_2)$ (alternation)
  - $(r_1 \cdot r_2)$ (concatenation)
- if r is regular expression, then so is r^* (closure)
- may also use a^- to mean traversal of edge labelled a in the *reverse* direction
- r^+ is shorthand for $(r \cdot r^*)$
- $r?$ is shorthand for $(r|\epsilon)$
- Σ is shorthand for $(a_1| \cdots |a_n)$ if $\Sigma = \{a_1, \ldots, a_n\}$

Regular languages

Language $L(r)$ (set of sequences of labels) denoted by r is given by:

- ϵ denotes $\{\epsilon\}$
- $a \in \Sigma$ denotes $\{a\}$
- $(r_1 | r_2)$ denotes $L(r_1) \cup L(r_2)$
- $(r_1 \cdot r_2)$ denotes $L(r_1) \cdot L(r_2)$
- r^* denotes $L(r)^*$

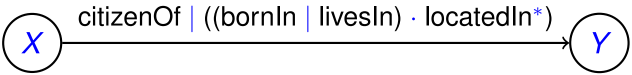
Paths satisfying regular expressions

Given a graph $G = (N, E, V, \Sigma, \phi, \lambda)$

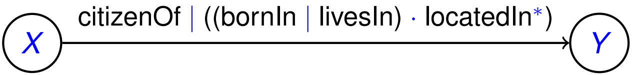
- a *path* p is a sequence of edges $(e_1, e_2, \ldots, e_n)$ such that, for each $1 \leq i \leq n$, if $\phi(e_i) = (x, y)$, then $\phi(e_{i+1}) = (y, z)$ for some $x, y, z \in N$
- the *path label* of p is given by $\lambda(e_1) \cdot \lambda(e_2) \cdots \lambda(e_n)$ and is denoted $\lambda(p)$
- path p *satisfies* regular expression r if $\lambda(p) \in L(r)$

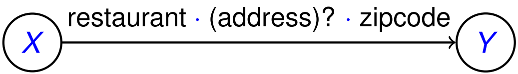
Regular path query: given r and G, find all pairs of nodes (x, y) in G such there is a path from x to y which satisfies r

Examples of regular path queries

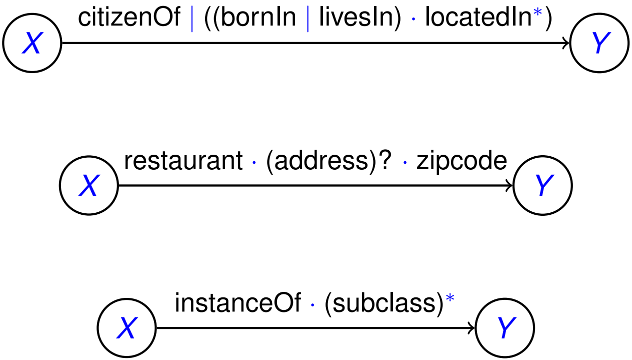


Examples of regular path queries

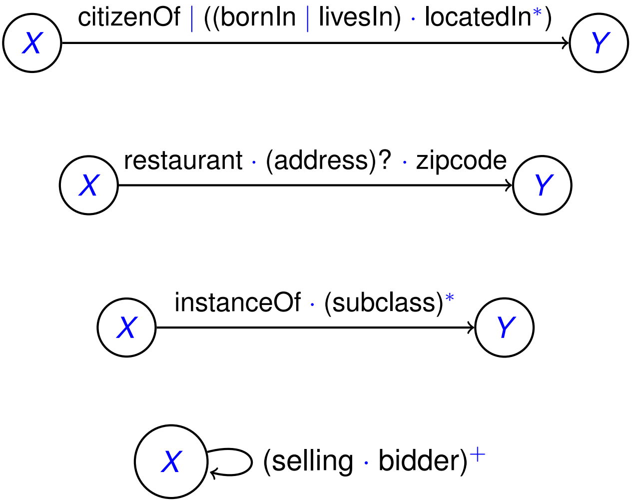




Examples of regular path queries



Examples of regular path queries

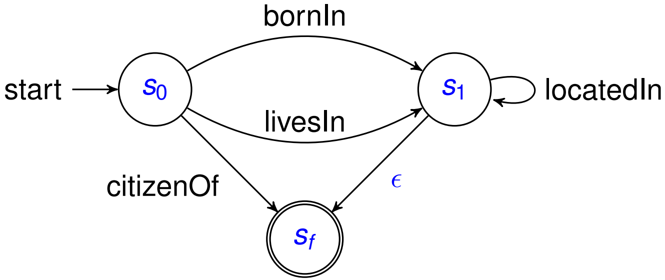


Complexity of regular path query evaluation

- REGULAR PATH PROBLEM
  Given graph G, pair of nodes x and y and regular expression r, is there a path from x to y satisfying r?
- algorithm:
  - construct a nondeterministic finite automaton (NFA) M accepting $L(r)$
  - assume M has initial state s_0 and final state s_f
  - consider G as an NFA with initial state x and final state y
  - form the “intersection” I of M and G
  - check if there is a path from (s_0, x) to (s_f, y)
- Each step can be done in PTIME, so REGULAR PATH PROBLEM has PTIME combined complexity

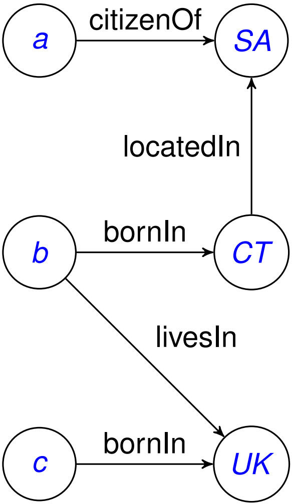
Regular path query evaluation

NFA M for $r = \text{citizenOf} \mid ((\text{bornIn} \mid \text{livesIn}) \cdot \text{locatedIn}^*)$



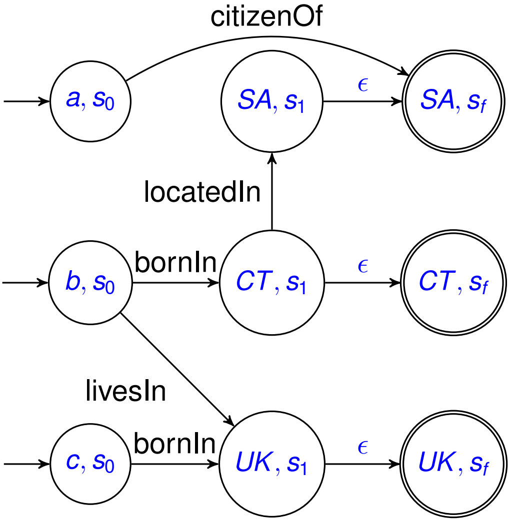
Regular path query evaluation

Graph G:



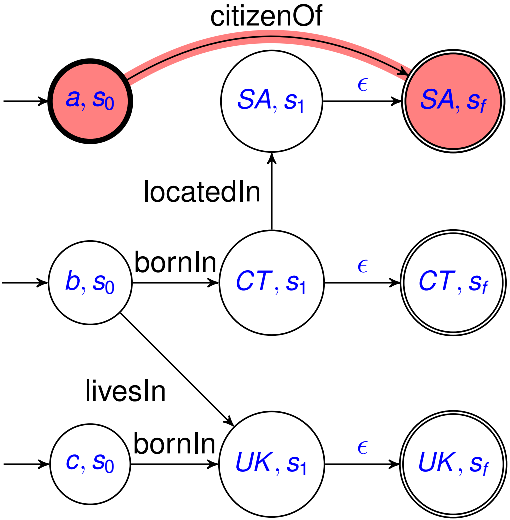
Regular path query evaluation

Intersection of *G* and *M*



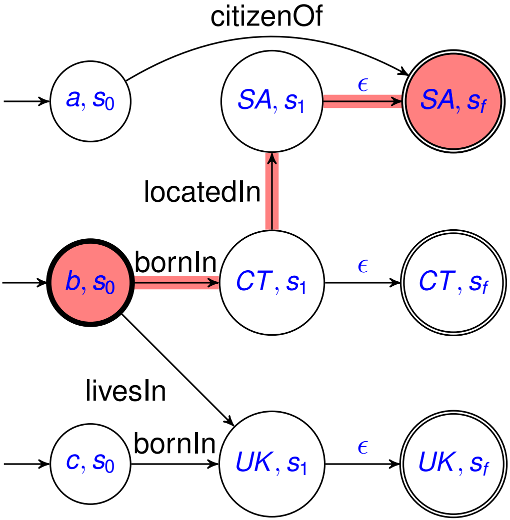
Regular path query evaluation

Intersection of G and M



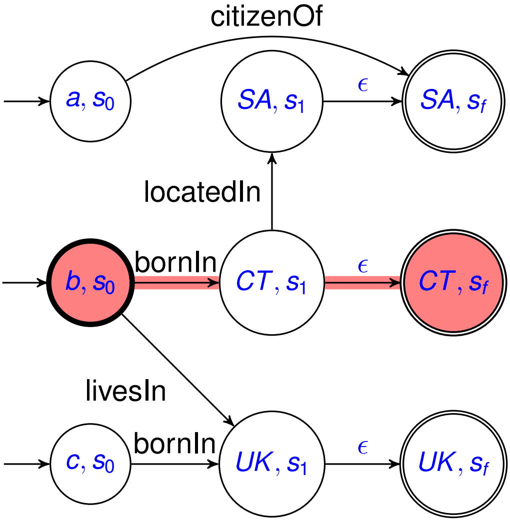
Regular path query evaluation

Intersection of G and M



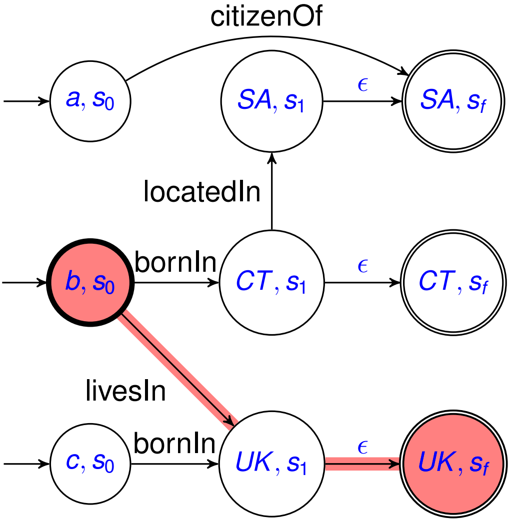
Regular path query evaluation

Intersection of G and M



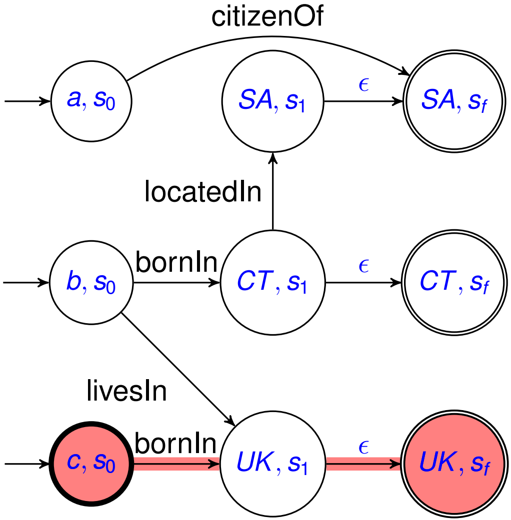
Regular path query evaluation

Intersection of G and M



Regular path query evaluation

Intersection of G and M



Regular path query evaluation

Alternatively can translate

$$citizenOf \mid ((bornIn \mid livesIn) \cdot locatedIn^*)$$

to Datalog (as done by Graphlog, e.g.)

$$\begin{aligned}
assoc(X, Y) &\leftarrow bornIn(X, Y) \\
assoc(X, Y) &\leftarrow livesIn(X, Y) \\
partOf(X, Y) &\leftarrow locatedIn(X, Y) \\
partOf(X, Y) &\leftarrow locatedIn(X, Z), partOf(Z, Y) \\
answer(X, Y) &\leftarrow citizenOf(X, Y) \\
answer(X, Y) &\leftarrow assoc(X, Y) \\
answer(X, Y) &\leftarrow assoc(X, Z), partOf(Z, Y)
\end{aligned}$$

Regular simple path queries

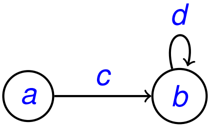
- path p is *simple* if no node is repeated on p
- REGULAR SIMPLE PATH PROBLEM
  Given graph G, pair of nodes x and y and regular expression r, is there a *simple* path from x to y satisfying r?

Regular simple path queries

- path p is *simple* if no node is repeated on p
- REGULAR SIMPLE PATH PROBLEM
  Given graph G, pair of nodes x and y and 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 [Mendelzon and Wood, 1989]

Regular simple path queries

- path p is *simple* if no node is repeated on p
- REGULAR SIMPLE PATH PROBLEM
  Given graph G, pair of nodes x and y and 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 [Mendelzon and Wood, 1989]
- there can be a path from x to y satisfying r but no simple path satisfying r, e.g., $r = (c \cdot d)^*$



Outline

Edge label variables

- what relationship(s) exist between *Coetzee* and *SouthAfrica*?

$$X \leftarrow (Coetzee, X, SouthAfrica)$$

  a "schema-level" query
- answers might be: $\{bornIn, livesIn, citizenOf\}$
- find people X and things Y such that X is related Y in the same way as *Coetzee* is related to Y

$$(X, Y) \leftarrow (Coetzee, Z, Y), (Y, Z^-, X)$$

  superscript $^-$ indicates traversal in *reverse* direction

Edge label variables

- program analysis example: database transaction
- graph:
  - nodes represent points in a program
  - special nodes *start* and *end*
  - edges represent operations, e.g., $lock(b)$ and $unlock(b)$ of some data item b
- is it the case that a transaction tries to lock the same item more than once (not two-phase)?

$$\leftarrow (start, (\Sigma^* \cdot lock(X) \cdot \Sigma^* \cdot lock(X) \cdot \Sigma^*), end)$$

- Σ^* matches any sequence of edge labels
- sometimes called *parameterised* regular expressions

Negation

- program analysis: *def* and *use* of program variables
- to find program points that immediately follow a use of an uninitialized variable

$$Y \leftarrow (start, (\neg def(X))^* \cdot use(X), Y)$$

Negation

- program analysis: *def* and *use* of program variables
- to find program points that immediately follow a use of an uninitialized variable

$$Y \leftarrow (start, (\neg def(X))^* \cdot use(X), Y)$$

- to find only the first use of each uninitialized variable along each path

$$\begin{aligned} Y, Z \quad \leftarrow \quad & (start, ((\neg(def(X) \mid use(X)))^*), Y), \\ & (Y, use(X), Z) \end{aligned}$$

Path variables

- may want to know path(s) connecting two nodes:
  - linked data on the web (DBPedia, Freebase)
  - link analysis in criminal networks
  - data provenance
- given regular expression r and variable X, use $(r)\%X$ to bind path matching r to X
- edge label variable X is a special case where first occurrence is equivalent to $(\Sigma)\%X$
- paths connecting *Coetzee* and *Gordimer* given by

$$X \leftarrow (Coetzee, ((\Sigma|\Sigma^-)^*)\%X, Gordimer)$$

  answers: $bornIn \cdot bornIn^-$ and $hasWon \cdot hasWon^-$
- Lorel uses @, not %; NAGA uses *connect* keyword

Path variables

- find entities X and Y such that *Coetzee* is connected to Y in the same way as X is connected to Y

$$(X, Y) \leftarrow (\mathit{Coetzee}, (\Sigma^*)\%Z, Y), (X, Z, Y)$$

- similar to regular expressions with backreferencing, e.g., in *egrep* (Unix) and in Perl
- membership problem is NP-complete [Aho, 1980]; data complexity is PTIME
- in general, can denote non-context-free languages, e.g., $\{ww \mid w \in \Sigma^*\}$ as above
- can also do local binding: $((\Sigma\%X) \cdot X)^*$

Outline

Motivation

To be able to answer traditional graph queries like

- degree of a node
- distance between pairs of nodes
- eccentricity of a node
- diameter, radius and centre of a graph

Motivation

To be able to answer traditional graph queries like

- degree of a node
- distance between pairs of nodes
- eccentricity of a node
- diameter, radius and centre of a graph

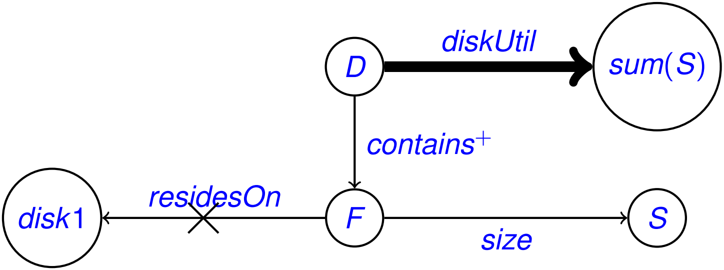
and applications like

- shortest path
- most reliable path
- critical path
- bill of materials
- ...

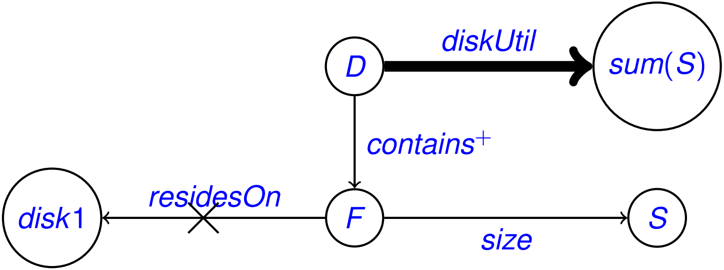
need operators such as count, min, max, sum

Aggregation in Graphlog

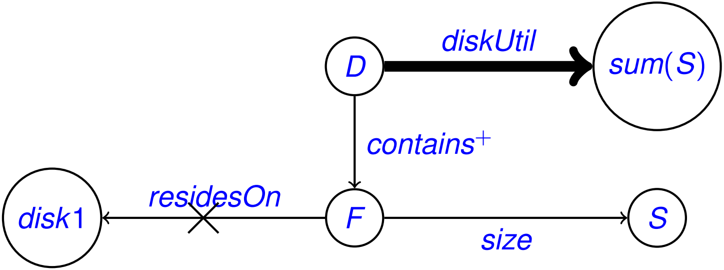
- aggregate terms are allowed in label of distinguished edge or distinguished node
- following query computes, for each directory D, the total file space used by all contained files and sub-directories, other than those residing on $disk1$



From Graphlog to Datalog



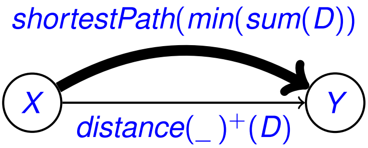
From Graphlog to Datalog



$$containsPlus(X, Y) \leftarrow contains(X, Y)$$
$$containsPlus(X, Y) \leftarrow contains(X, Z), containsPlus(Z, Y)$$
$$diskUtil(D, sum(S)) \leftarrow containsPlus(D, F), size(F, S), \neg residesOn(F, disk1)$$

Graphlog example

- we might also want to summarise values *along* a path and then aggregate
- following query computes the length of the shortest path between each pair of nodes



- *D* is called a *collecting* variable
- *sum* is used to summarise distances along a path
- *min* is used to aggregate the summarised distances
- query evaluation is in PTIME if summarisation and aggregation operators form a closed semiring [Consens and Mendelzon, 1990]

Outline

Motivation

- users may not be familiar with graph structure/constraints
- may formulate queries which return no answers or too few answers, e.g.
  - expression *course* · *student* when correct path is *student* · *course*
  - expression *restaurant* · *zipcode* when *address* is required between them
- can perform approximate matching of paths
- rank results in terms of "closeness" to original query

Approximate matching

- can modify the user's original query (regular expression *r*)
- one way is to apply *edit operations* to $L(r)$
  - insertions
  - deletions
  - substitutions
  - transpositions
  - invertions
- each operation may have a different cost
- somewhat related to user *preferences*
  - prepared to substitute *train* by *bus* but at cost 2

Approximate matching algorithm

- for conjunctive regular path queries
- can use algorithms from approximate string matching
- incrementally build an *approximate* NFA
- perform incremental joins for conjuncts
- PTIME combined complexity if conjuncts are acyclic and fixed number of head variables
- in general, can transform NFA using a *regular transducer*
- see [Hurtado, Poulovassilis and Wood, 2009]

Summary

- motivated that graph-based data is widely used and available
- a brief high-level overview of some query languages for graph databases
- focussed on query language functionality
- some discussion of query evaluation algorithms
- some complexity results mentioned

Issues not covered

Many issues not covered

- other languages
- more query evaluation strategies, e.g., using indexes
- graphs with schemas
- query optimisation, e.g., containment
- …

References

S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener.
The LOREL query language for semistructured data.
Int. J. on Digital Libraries, 1(1):68–88, April 1997.

A. V. Aho.
Pattern matching in strings.
In R. V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 325–347. Academic Press, 1980.

R. Angles and C. Gutierrez.
Survey of graph database models.
ACM Comput. Surv., 40(1):1–39, 2008.

M. P. Consens and A. O. Mendelzon.
Expressing structural hypertext queries in GraphLog.
In *Proc. Second ACM Conf. on Hypertext*, pages 269–292, 1989.

References

- M. P. Consens and A. O. Mendelzon.
  Low complexity aggregation in GraphLog and Datalog.
  In *Proc. 3rd Int. Conf. on Database Theory*, pages 379–394, 1990.
- I. F. Cruz, A. O. Mendelzon, and P. T. Wood.
  A graphical query language supporting recursion.
  In *ACM SIGMOD Int. Conf. on Management of Data*, pages 323–330, 1987.
- I. F. Cruz, A. O. Mendelzon, and P. T. Wood.
  G^+: Recursive queries without recursion.
  In *Proc. 2nd Int. Conf. on Expert Database Systems*, pages 355–368, 1988.

References

- C. A. Hurtado, A. Poulovassilis, and P. T. Wood.
  Ranking Approximate Answers to Semantic Web Queries.
  In *Proc. 6th European Semantic Web Conference*, pages 263–277, 2009.
- A. O. Mendelzon and P. T. Wood.
  Finding regular simple paths in graph databases.
  In *Proc. 15th Int. Conf. on Very Large Data Bases*, pages 185–193, 1989.
- G. Weikum, G. Kasneci, M. Ramanath, and F. Suchanek.
  Database and information-retrieval methods for knowledge discovery.
  Commun. ACM, 52(4):56–64, 2009.

References

M. Yannakakis.
Algorithms for acyclic database schemes.
In *Proc. 7th Int. Conf. on Very Large Data Bases*, pages 82–94, 1981.