Graph Queries: Generation, Evaluation and Learning

Angela Bonifati
University Lyon 1 & CNRS Liris

GraphQ 2017
Venice, Italy
21 March 2017

The ubiquity of graph-shaped data

Graph databases are everywhere

- massive search engines, e-commerce (Google, Yahoo!, Microsoft)
- social networks (e.g., LinkedIn, Facebook, Twitter)
- knowledge graphs (e.g., DBPedia, citation networks)
- experimental data, numerical simulations
- ...

The ubiquity of graph-shaped data

Graph databases are everywhere

- massive search engines, e-commerce (Google, Yahoo!, Microsoft)
- social networks (e.g., LinkedIn, Facebook, Twitter)
- knowledge graphs (e.g., DBPedia, citation networks)
- experimental data, numerical simulations
- ...

E.g. Facebook has 1.39B active users as of 12/2014 with more than 400B edges Ching, Avery, et al. "One trillion edges: Graph processing at facebook-scale." Proceedings of the VLDB Endowment 8.12 (2015): 1804-1815.

The ubiquity of graph-shaped data

Analytics on graph databases increasingly important

- data mining on integrated graph data
- role discovery in social networks
- searching related literature in a citation network
- inspecting large-scale scientific datasets
- ...

Graph queries as building blocks

Graph Analytics = graph mining algorithms + exploratory graph queries

> *A [N]ew paradigm shift is set forth for graph query languages due to their navigational capabilities*
>
> P. Barcelo (*PODS*, 2013)

Navigational properties needed in graph query languages

Navigational properties needed in graph query languages

Classical query languages, such as SQL(1) and Datalog(2), are not suitable due to (1) their limited recursion and (2) their higher data complexity.

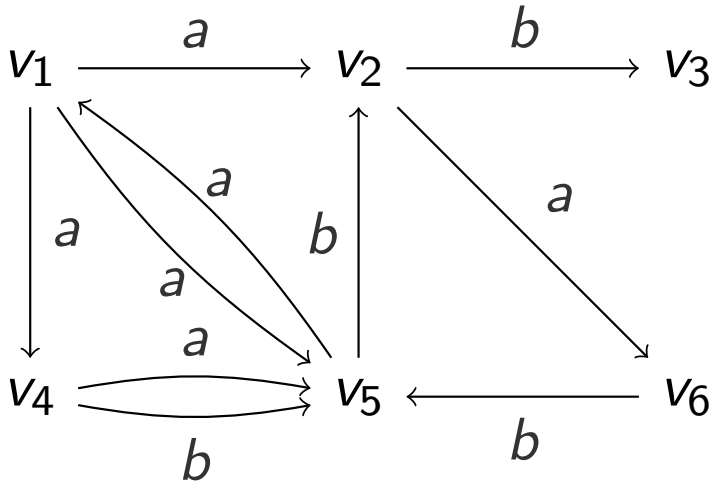
Basic Ingredients: Graph Databases and Queries

Graph databases

db-graph

A db-graph is an edge-labeled graph $G = (V, \Sigma, E)$

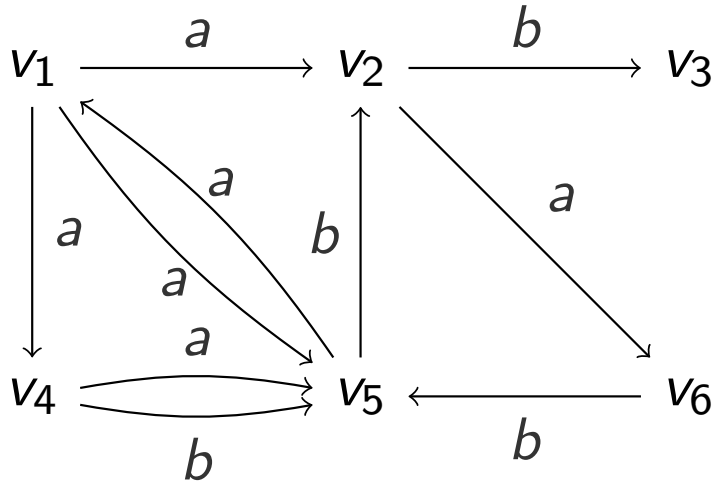
- V is the set of vertices;
- Σ is the set of labels;
- $E \subseteq V \times \Sigma \times V$ is the set of edges.



Edge-labeled Paths

- An edge-labeled path is a sequence $(v_1, a_1, \ldots a_k, v_{k+1})$ with $(v_i, a_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, k\}$.
- The label of a path is the word formed by its edge labels.

Example:

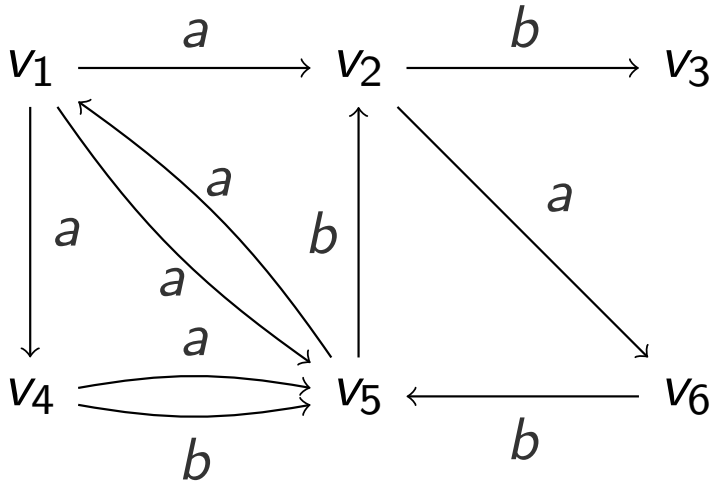


- $p_1 = (v_1, a, v_4, a, v_5, b, v_2)$ is a path from v_1 to v_2 with label aab;
- $p_2 = (v_1, a, v_5, a, v_1, a, v_2)$ is a path from v_1 to v_2 with label aaa.

Edge-labeled Paths

- An edge-labeled path is a sequence $(v_1, a_1, \ldots a_k, v_{k+1})$ with $(v_i, a_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, k\}$.
- The label of a path is the word formed by its edge labels.

Example:

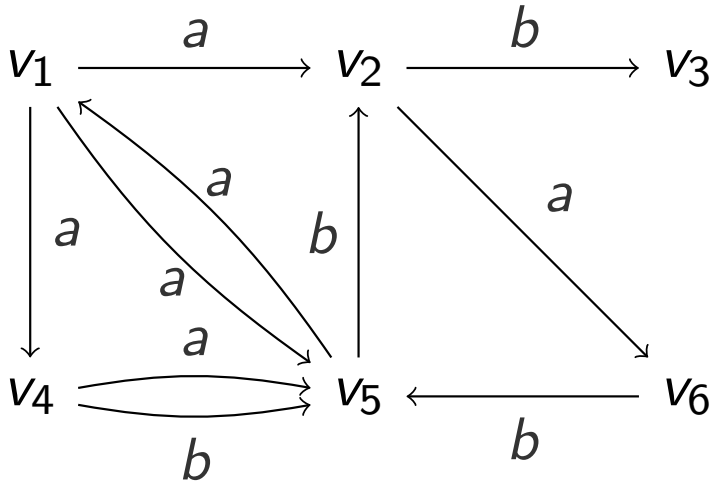


- $p_1 = (v_1, a, v_4, a, v_5, b, v_2)$ is a path from v_1 to v_2 with label aab;
- $p_2 = (v_1, a, v_5, a, v_1, a, v_2)$ is a path from v_1 to v_2 with label aaa.

Edge-labeled Paths

- An edge-labeled path is a sequence $(v_1, a_1, \ldots a_k, v_{k+1})$ with $(v_i, a_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, k\}$.
- The label of a path is the word formed by its edge labels.

Example:

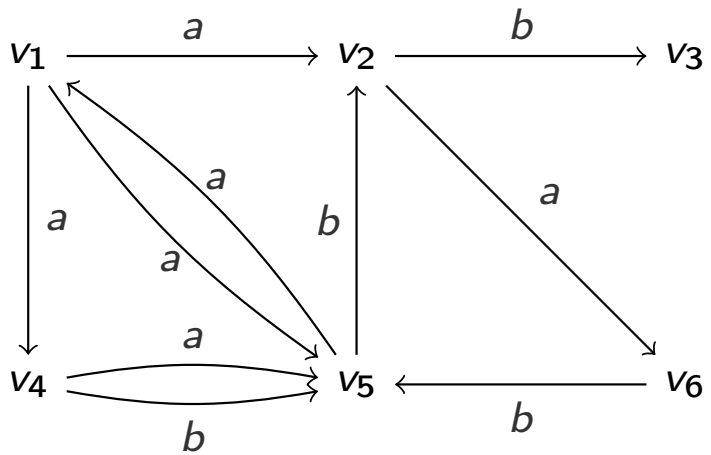


- $p_1 = (v_1, a, v_4, a, v_5, b, v_2)$ is a path from v_1 to v_2 with label aab;
- $p_2 = (v_1, a, v_5, a, v_1, a, v_2)$ is a path from v_1 to v_2 with label aaa.

Regular Path Queries

- A regular path query (RPQ) over the set of edge labels Σ is expressed as a regular expression over Σ.
- The answer $Q(D)$ to an RPQ Q over a database D is the set of pairs of nodes connected in D by a directed path traversing a sequence of edges forming a word in the regular language $L(Q)$ defined by Q.

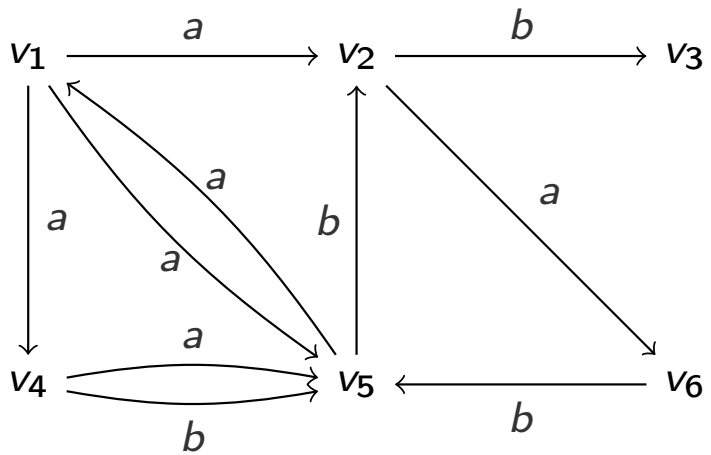
Example:



Regular Path Queries

- A regular path query (RPQ) over the set of edge labels Σ is expressed as a regular expression over Σ.
- The answer $Q(D)$ to an RPQ Q over a database D is the set of pairs of nodes connected in D by a directed path traversing a sequence of edges forming a word in the regular language $L(Q)$ defined by Q.

Example:



- $Q = b*$ is an RPQ returning the pairs of nodes (v_2, v_3), (v_4, v_2), (v_4, v_3), (v_4, v_5), (v_5, v_2), (v_5, v_3), (v_6, v_2), (v_6, v_3), (v_6, v_5).

A query language for graphs

UCRPQ: Unions of Conjunctions of Regular Path Queries

– Core constructs of the W3C's SPARQL 1.1, Oracle's PGQL, and and Neo4j's openCypher

– Well understood theoretical properties (e.g., polynomial data complexity)

A query language for graphs

UCRPQ: Unions of Conjunctions of Regular Path Queries
– Core constructs of the W3C's SPARQL 1.1, Oracle's PGQL, and and Neo4j's openCypher
– Well understood theoretical properties (e.g., polynomial data complexity)

UCRPQ includes **recursive queries** (via the Kleene star *), with applications in social networks, bioinformatics, etc.

A query language for graphs

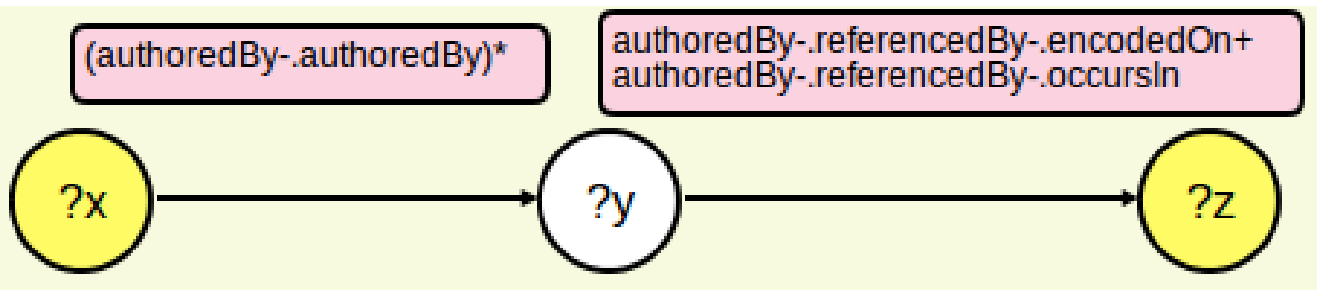
Example of UCRPQ

> *for each researcher, select all of the biological entities (i.e., genes and organisms) relevant to proteins studied in papers authored by people in the researcher's coauthorship network*

A query language for graphs

Example of UCRPQ

for each researcher, select all of the biological entities (i.e., genes and organisms) relevant to proteins studied in papers authored by people in the researcher's coauthorship network



A query language for graphs

Example of UCRPQ

for each researcher, select all of the biological entities (i.e., genes and organisms) relevant to proteins studied in papers authored by people in the researcher's coauthorship network

$$(?x, ?z) \leftarrow (?x, (\mathtt{a}^- \cdot \mathtt{a})^*, ?y), (?y, (\mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{e} + \mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{o}), ?z)$$

(a=authoredBy, r=referencedBy, e=encodedOn, o=occursIn)

Graph queries, from generation to learning

Over the past few years, we have been investigating the ways in which:

- graph queries can be generated, evaluated and inferred.

This talk surveys this work, which is the result of collaborations with my colleagues at CNRS Liris, Eindhoven University of Technology, Université Clermont Auvergne, Université Paris Sud, and Université Lille 3.
Full bibliographic details can be found on the last slides and in the abstract accompanying the talk.

Graph queries, from generation to learning

- part 1: gMark for schema-driven generation of graph instances and queries

Graph queries, from generation to learning

- part 1: gMark for schema-driven generation of graph instances and queries
- part 2: complexity of evaluation of simple regular path queries

Graph queries, from generation to learning

- part 1: gMark for schema-driven generation of graph instances and queries
- part 2: complexity of evaluation of simple regular path queries
- part 3: user-driven inference of regular path queries

Graph queries, from generation to learning

- part 1: gMark for schema-driven generation of graph instances and queries
- part 2: complexity of evaluation of simple regular path queries
- part 3: user-driven inference of regular path queries

Synthetic graph and workload generation with gMark

We present gMark, an open-source framework for generation of synthetic graphs and workloads.

gMark generates UCRPQ $\rightarrow$ the first synthetic workload generator to support recursive queries (and their generation in concrete syntaxes).

Synthetic graph and workload generation with gMark

We present gMark, an open-source framework for generation of synthetic graphs and workloads.

gMark generates UCRPQ $\rightarrow$ the first synthetic workload generator to support recursive queries (and their generation in concrete syntaxes).

gMark has been designed to tailor diverse graph data management scenarios, which are often driven by query workloads, such as multi-query optimization, data integration and database physical design.

Synthetic graph and workload generation with gMark

Given a graph schema, gMark

- generates synthetic instances of the schema (of desired size)
- generates query workloads with targeted structure and runtime behavior (which holds for all instances of the schema)

Why gMark?

We adopt successful aspects of the state of the art

For example, like the Waterloo Diversity Benchmark (ISWC 2014), gMark is schema-driven,

- allowing finely tailored graph instances for specific application domains; and,
- allowing tightly controlled generation of query workloads.

Why gMark?

We adopt successful aspects of the state of the art

For example, like the Waterloo Diversity Benchmark (ISWC 2014), gMark is schema-driven,

- allowing finely tailored graph instances for specific application domains; and,
- allowing tightly controlled generation of query workloads.

and, like the LDBC SNB Interactive (SIGMOD 2015), gMark supports focused stress-testing of query optimization choke-points, through fine control of query parameters such as selectivities.

Why gMark?

New features of gMark include

- support for flexible generation of query workloads including recursive path queries, which are fundamental for graph analytics;

Why gMark?

New features of gMark include

- support for flexible generation of query workloads including recursive path queries, which are fundamental for graph analytics; and,
- query selectivity estimation solution, in a purely instance-independent schema-driven fashion.
  - hence, more scalable, more predictable, and easier to explain/understand.

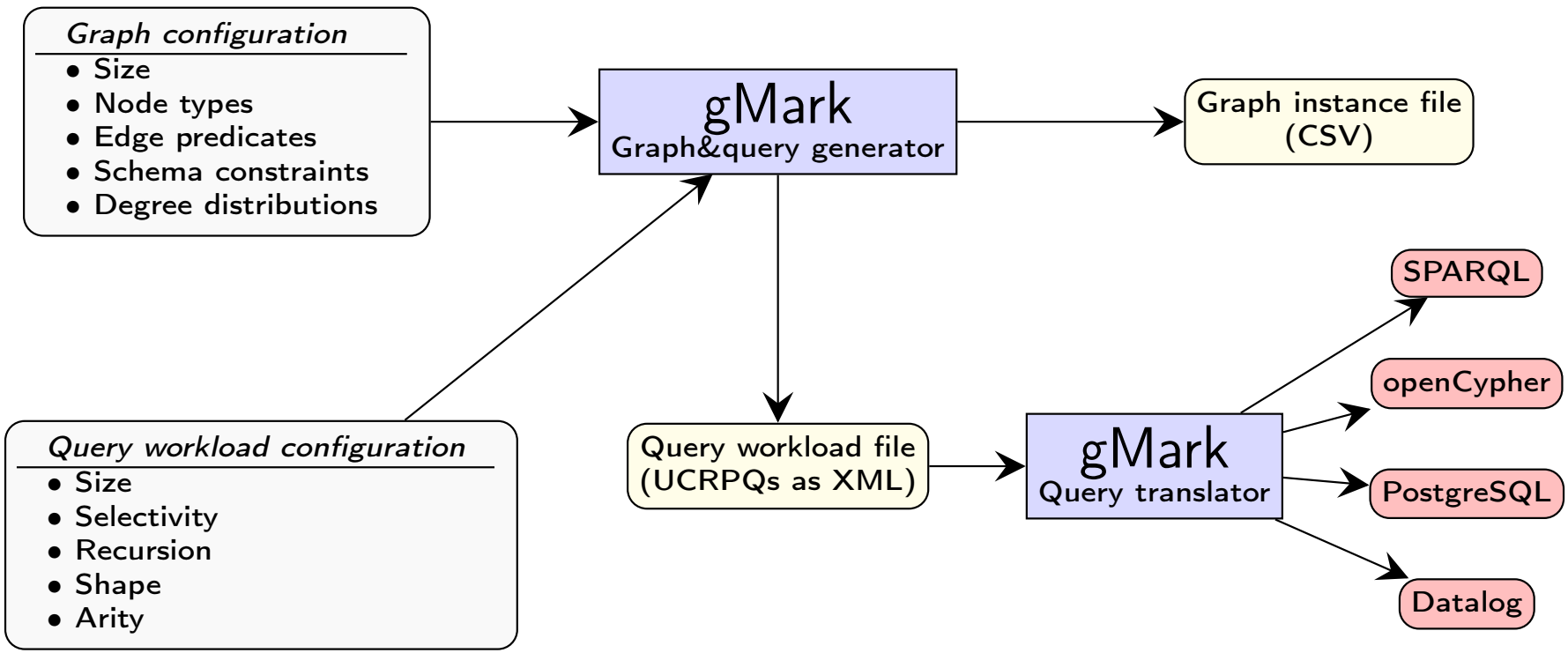
Why gMark?

New features of gMark include

- support for flexible generation of query workloads including recursive path queries, which are fundamental for graph analytics; and,
- query selectivity estimation solution, in a purely instance-independent schema-driven fashion.
  - hence, more scalable, more predictable, and easier to explain/understand.

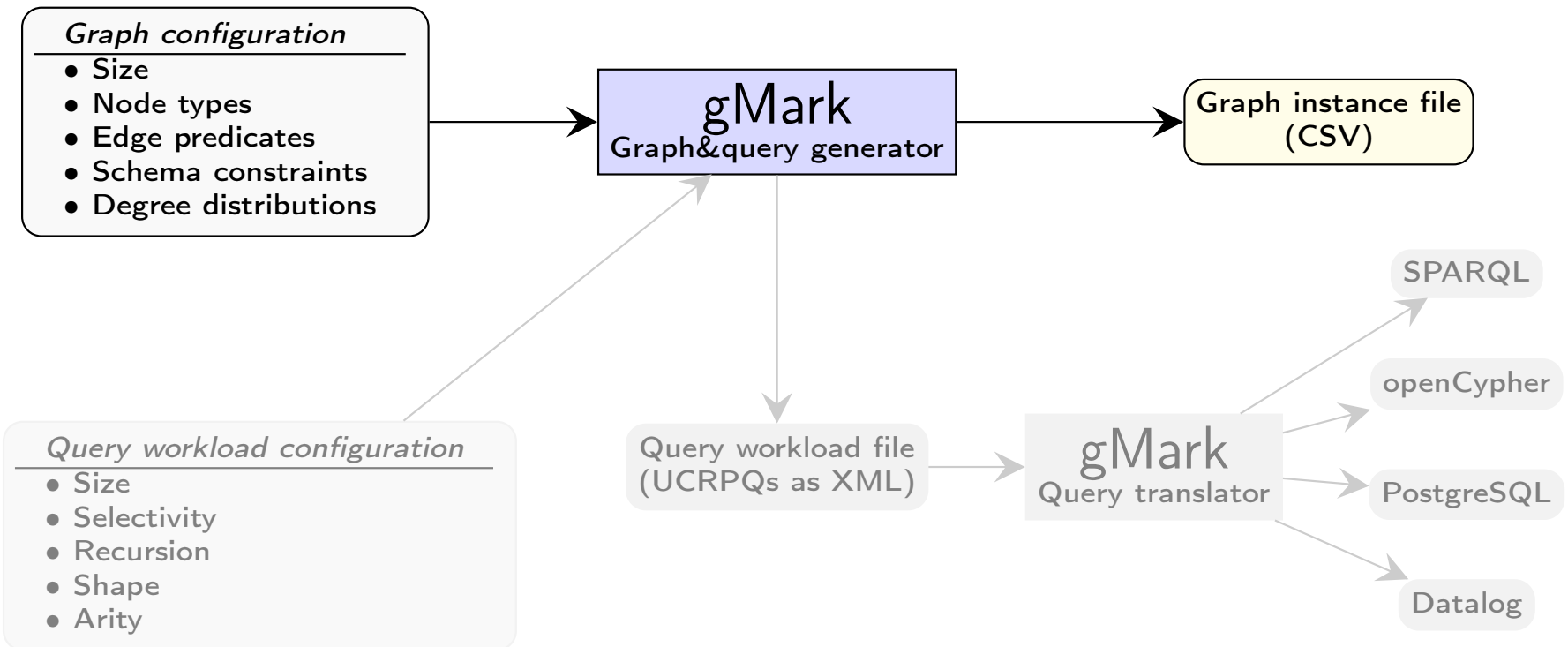
Neither of these are supported in WatDiv and LDBC.

Overview of the gMark workflow



Graph generation

gMark graph generation



Graph configurations

The user can specify in the graph configuration (i.e., graph schema):

- **Size**: # of nodes
- **Node types**: finite set of node labels

  e.g., `author`, `citation`, `journal`

Graph configurations

The user can specify in the graph configuration (i.e., graph schema):

- **Size**: # of nodes
- **Node types**: finite set of node labels

  e.g., `author`, `citation`, `journal`

- **Edge predicates**: finite set of edge labels

  e.g., `authoredBy`, `referencedBy`

Graph configurations

The user can specify in the graph configuration (i.e., graph schema):

- **Size**: # of nodes
- **Node types**: finite set of node labels

  e.g., `author`, `citation`, `journal`

- **Edge predicates**: finite set of edge labels

  e.g., `authoredBy`, `referencedBy`

- **Schema constraints**: proportion of nodes/edges of given type

  e.g., 20% of all nodes are `authors`

- **Degree distributions**: on the in- and out-degree of edge predicates (uniform, normal, zipfian)

  e.g., the out-distribution of `citation` $\xrightarrow{\texttt{authoredBy}}$ `author` is Gaussian with parameters $\mu = 3, \sigma = 1$

Graph configurations: Uniprot schema

Node type	*Constr.*
gene	35%
protein	31%
author	20%
citation	10%
organism	1%
...	...

Node types

Edge predicate	*Constr.*
authoredBy	64%
encodedOn	6%
referencedBy	3%
occursIn	2%
...	...

Edge predicates

source type $\xrightarrow{\textit{predicate}}$ *target type*	*In-distr.*	*Out-distr.*
protein $\xrightarrow{\text{encodedOn}}$ gene	Zipfian	Gaussian
protein $\xrightarrow{\text{occursIn}}$ organism	Zipfian	Uniform
protein $\xrightarrow{\text{referencedBy}}$ citation	Zipfian	Gaussian
citation $\xrightarrow{\text{authoredBy}}$ author	Zipfian	Gaussian
...	...	...

In- and out-degree distributions

Schema-driven graph generation

We have established the intractability of the generation problem

Theorem

Given a graph configuration G, deciding whether or not there exists a graph instance satisfying G is NP-complete.

Hence, gMark follows an heuristic approach in instance generation ($O(n)$), i.e., it attempts to achieve the exact values of the input parameters and relaxes them whenever this is not possible.

Schema-driven graph generation

We have adapted the scenarios of several popular use cases into meaningful gMark configurations, while also adding new gMark features:

- `Bib`: our default bibliographical use-case
- `LSN`: LDBC social network benchmark
- `WD`: WatDiv e-commerce benchmark
- `SP`: SP2Bench DBLP benchmark

Scalability of gMark graph generation

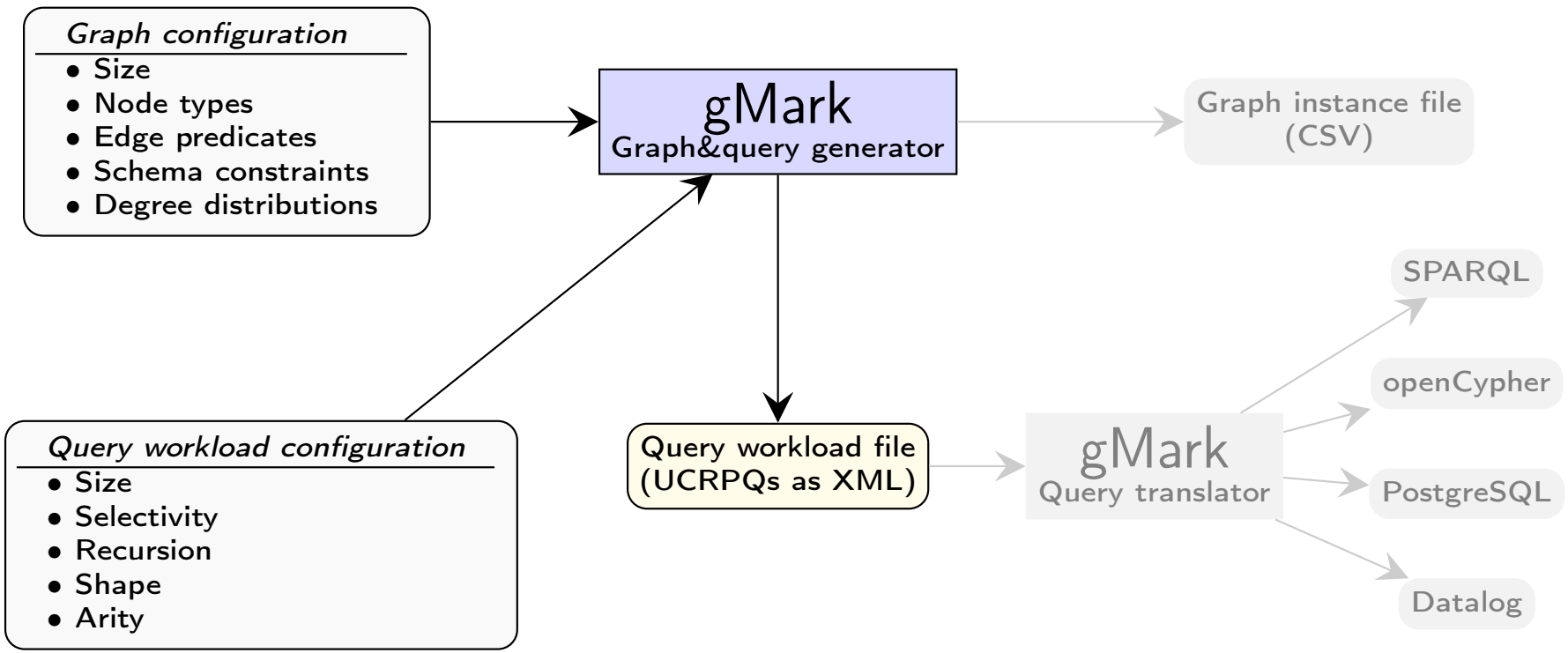
	100K	1M	10M	100M
`Bib`	0m0.057s	0m0.638s	0m8.344s	1m28.725s
`LSN`	0m0.225s	0m1.451s	0m23.018s	3m11.318s
`WD`	0m2.163s	0m25.032s	4m10.988s	113m31.078s
`SP`	0m0.638s	0m7.048s	1m28.831s	15m23.542s

Graph generation times, with varying graph sizes (# nodes)

Generation time depends heavily on density of instances (e.g., `WD` has 100x number of edges than `Bib`)

Query workload generation

gMark query generation



A query language for graphs

Example of UCRPQ

for each researcher, select all of the biological entities (i.e., genes and organisms) relevant to proteins studied in papers authored by people in the researcher's coauthorship network

$$(?x, ?z) \leftarrow (?x, (\mathtt{a}^- \cdot \mathtt{a})^*, ?y), (?y, (\mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{e} + \mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{o}), ?z)$$

(a=authoredBy, r=referencedBy, e=encodedOn, o=occursIn)

A query language for graphs

Example of UCRPQ

for each researcher, select all of the biological entities (i.e., genes and organisms) relevant to proteins studied in papers authored by people in the researcher's coauthorship network

$$(?x, ?z) \leftarrow (?x, (\mathtt{a}^- \cdot \mathtt{a})^*, ?y), (?y, (\mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{e} + \mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{o}), ?z)$$

(a=authoredBy, r=referencedBy, e=encodedOn, o=occursIn)

#rules	1
#conjuncts	2
#disjuncts	1, 2
path lengh	2, 3, 3

Schema-driven workload generation

The user can specify in the query workload configuration:

- **Size**: #queries, #conjuncts/#disjuncts/path length per query
- **Selectivity**: constant, linear, quadratic.
- **Recursion**: probability to generate Kleene star above a conjunct.

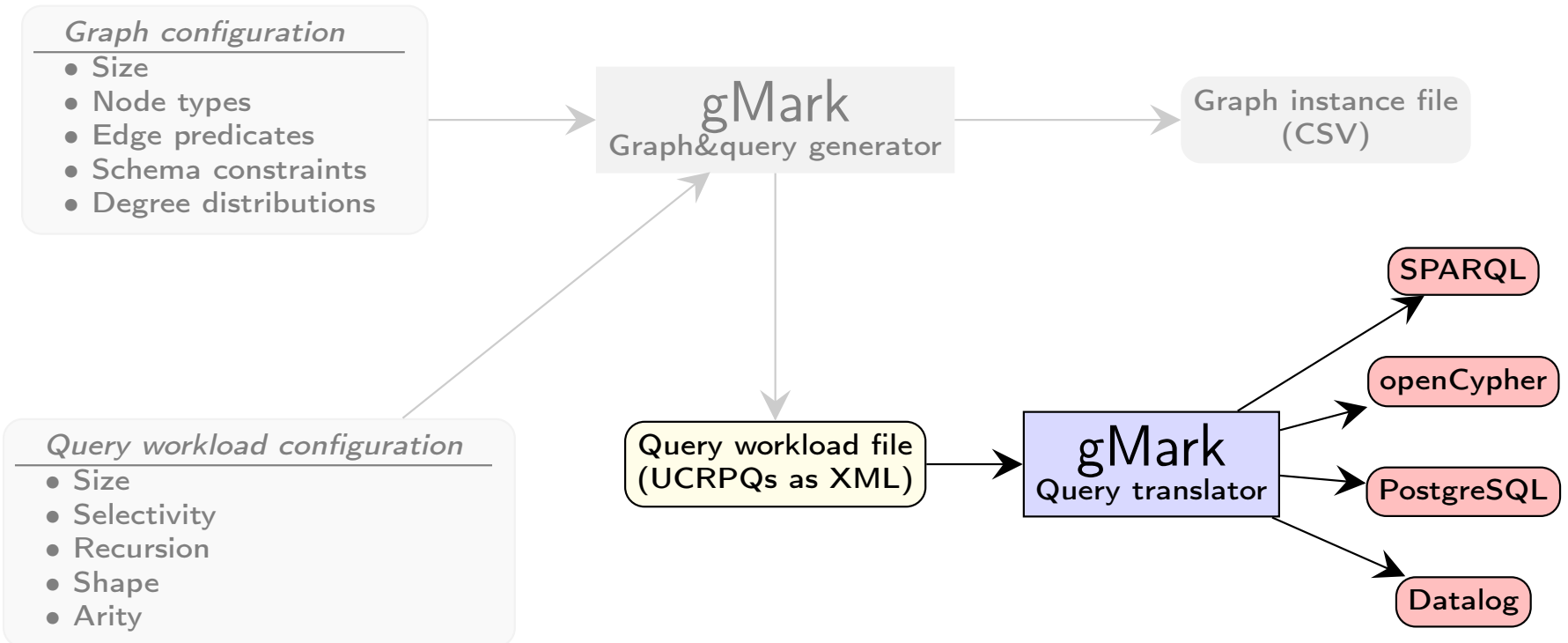
Schema-driven workload generation

The user can specify in the query workload configuration:

- **Size**: #queries, #conjuncts/#disjuncts/path length per query
- **Selectivity**: constant, linear, quadratic.
- **Recursion**: probability to generate Kleene star above a conjunct.
- **Shape**: chain, star, cycle, star-chain.
- **Arity**: arbitrary (including 0 i.e., Boolean).

The graph configuration is also input to the query generator.

gMark query translator



Query translation

UCRPQ: $(?x, ?z) \leftarrow (?x, (\mathtt{a}^- \cdot \mathtt{a})^*, ?y), (?y, (\mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{e} + \mathtt{a}^- \cdot \mathtt{r}^- \cdot \mathtt{o}), ?z)$

Query translation

UCRPQ: $(?x, ?z) \leftarrow (?x, (a^- \cdot a)^*, ?y), (?y, (a^- \cdot r^- \cdot e + a^- \cdot r^- \cdot o), ?z)$

SPARQL

```
PREFIX : <http://example.org/gmark/>
SELECT DISTINCT ?x ?z
WHERE { ?x (^:a/:a)* ?y .
?y ((^:a/^:r/:e)|(^:a/^:r/:o)) ?z .}
```

openCypher

```
MATCH (x)<-[:a]-()-[:a]->(y),
(y)<-[:a]-()<-[:r]-()-[:e]->(z)
RETURN DISTINCT x, z
UNION
MATCH (x)<-[:a]-()-[:a]->(y),
(y)<-[:a]-()<-[:r]-()-[:o]->(z)
RETURN DISTINCT x, z;
```

Datalog

```
g0(x,y)<- edge(x1,a,x0),edge(x1,a,x2),
    x=x0,y=x2.
g0(x,y)<- g0(x,z),g0(z,y).
g1(x,y)<- edge(x1,a,x0),edge(x2,r,x1),
    edge(x2,e,x3),x=x0,y=x3.
g1(x,y)<- edge(x1,a,x0),edge(x2,r,x1),
    edge(x2,o,x3),x=x0,y=x3.
query(x,z)<- g0(x,y),g1(y,z).
```

SQL

```
WITH RECURSIVE c0(src, trg) AS (
SELECT edge.src, edge.src FROM edge
UNION
SELECT edge.trg, edge.trg FROM edge
UNION
SELECT s0.src, s0.trg
FROM (SELECT trg as src, src as trg,
.....
```

Scalability of gMark workload generation

On my laptop, gMark easily generates workloads of one thousand queries for `Bib` in $\sim 0.3s$; `LSN` and `SP` in $\sim 1.5s$; and for the richer `WD` scenario in $\sim 10s$.

Query translation of the thousand queries into all four supported syntaxes for each of the four scenarios required $\sim 0.1s$.

Scalability on recursive query workloads

Example Application. We performed an extensive performance study of four state-of-the-art systems under the four use-case schemas.

Our main finding was that performance on queries containing recursive path navigation (i.e., RPQs) was typically impractical

- indicates the need for further study of the engineering of this basic class of graph queries

part 1: Conclusions and future work

Recap

Novel contributions of gMark

- schema-driven graph and query-workload generation, featuring instance-independent selectivity estimation;
- finely controlled query workload-centered approach
  - versus query-centered approaches – nb. both are valid and needed!
- discovery of the performance difficulties of existing graph DBMS's on evaluating a basic class of graph queries
  - Regular Path Queries

Recap

Novel contributions of gMark

- schema-driven graph and query-workload generation, featuring instance-independent selectivity estimation;
- finely controlled query workload-centered approach
  - versus query-centered approaches – nb. both are valid and needed!
- discovery of the performance difficulties of existing graph DBMS's on evaluating a basic class of graph queries
  - Regular Path Queries

https://github.com/graphMark/gmark

Looking ahead to gMark v2.0

Extensions/Wishlist.

- properties of the generation
  - stability of nodes across the generation process[1]

[1] Come to our EDBT17 poster session: W. van Leeuwen, A. Bonifati, G. Fletcher, and N. Yakovets. *Stability notions in synthetic graph generation: a preliminary study*.

Looking ahead to gMark v2.0

Extensions/Wishlist.

- properties of the generation
  - stability of nodes across the generation process[1]
- richer queries
  - support of constants in queries
  - additional query shapes
  - extensions of selectivity estimation to higher arity queries,

[1] Come to our EDBT17 poster session: W. van Leeuwen, A. Bonifati, G. Fletcher, and N. Yakovets. *Stability notions in synthetic graph generation: a preliminary study*.

Looking ahead to gMark v2.0

Extensions/Wishlist.

- properties of the generation
  - stability of nodes across the generation process[1]
- richer queries
  - support of constants in queries
  - additional query shapes
  - extensions of selectivity estimation to higher arity queries,
- richer schemas
  - configuration parameter completion,
  - schema constructs for correlated structure

[1] Come to our EDBT17 poster session: W. van Leeuwen, A. Bonifati, G. Fletcher, and N. Yakovets. *Stability notions in synthetic graph generation: a preliminary study*.

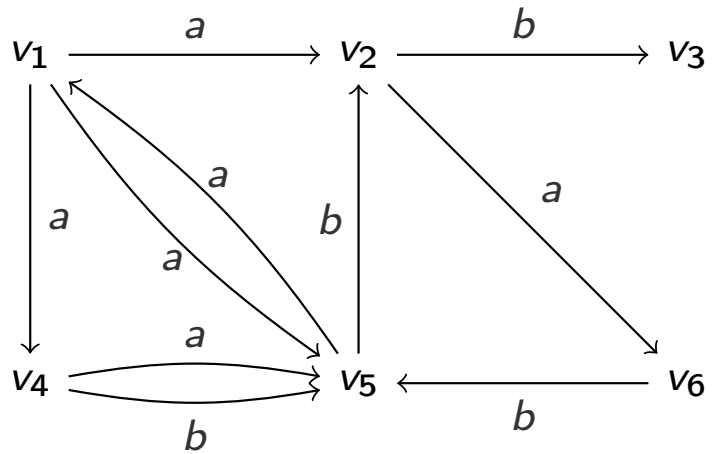
Graph queries, from generation to learning

- part 1: gMark for schema-driven generation of graph instances and queries
- part 2: complexity of evaluation of simple regular path queries
- part 3: user-driven inference of regular path queries

Paths

- A path is a sequence $(v_1, a_1, \ldots a_k, v_{k+1})$ with $(v_i, a_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, k\}$.
- A path is simple if it does not contain cycles (repetitions of vertices).
- The label of a path is the word formed by its edge labels.
- An L-labeled path is a path whose label belongs to L.

Example:

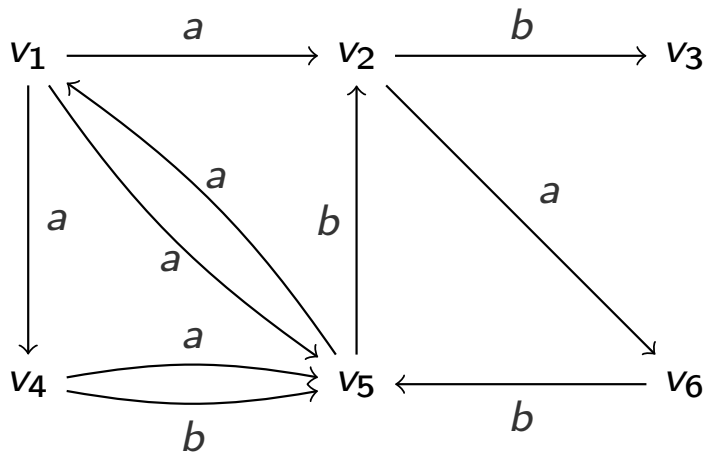


- $p_1 = (v_1, a, v_4, a, v_5, b, v_2)$ is a simple path from v_1 to v_2 with label aab;
- $p_2 = (v_1, a, v_5, a, v_1, a, v_2)$ is a (non simple) path from v_1 to v_2 with label aaa.

Paths

- A path is a sequence $(v_1, a_1, \ldots a_k, v_{k+1})$ with $(v_i, a_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, k\}$.
- A path is simple if it does not contain cycles (repetitions of vertices).
- The label of a path is the word formed by its edge labels.
- An L-labeled path is a path whose label belongs to L.

Example:

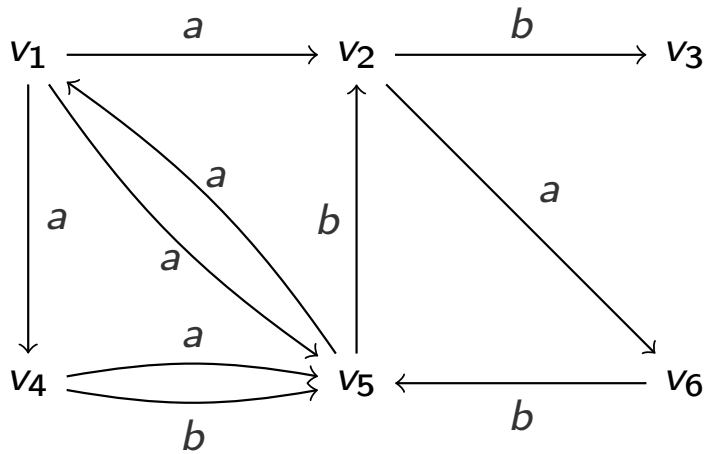


- $p_1 = (v_1, a, v_4, a, v_5, b, v_2)$ is a simple path from v_1 to v_2 with label aab;
- $p_2 = (v_1, a, v_5, a, v_1, a, v_2)$ is a (non simple) path from v_1 to v_2 with label aaa.

Paths

- A path is a sequence $(v_1, a_1, \ldots a_k, v_{k+1})$ with $(v_i, a_i, v_{i+1}) \in E$ for every $i \in \{1, \ldots, k\}$.
- A path is simple if it does not contain cycles (repetitions of vertices).
- The label of a path is the word formed by its edge labels.
- An L-labeled path is a path whose label belongs to L.

Example:



- $p_1 = (v_1, a, v_4, a, v_5, b, v_2)$ is a simple path from v_1 to v_2 with label aab;
- $p_2 = (v_1, a, v_5, a, v_1, a, v_2)$ is a (non simple) path from v_1 to v_2 with label aaa.

Regular path queries

RPQ(L)

- A db-graph G, two nodes x and y
- Is there an L-labeled path from x to y?

RSPQ(L)

- A db-graph G, two nodes x and y
- Is there an L-labeled simple path from x to y?

Applications of RSPQ

- Sparql [Losemann and Martens 2012, Arenas et al 2012]
- Semantic web
- Biological networks
- Wireless networks
- Transportation problems

Complexity [Mendelzon and Wood 95]

- RPQ(L) is polynomial for combined complexity (both in the size of the graph and the regular expression);
- RSPQ(L) is NP-complete for data complexity.
  - For example, if we fix $L = a^*ba^*$ or $L = (aa)^*$.

Results already known

RSPQ(L) is polynomial (combined complexity):

- when L is closed by subwords [Mendelzon and Wood 95];
- over acyclic graphs [Mendelzon and Wood 95];
- over outerplanar graphs [Nedev and Wood 2000];
- over bounded treewidth graphs [Barrett et al 2000];

Our goal

Question

For which languages L, RSPQ(L) is tractable?

Our main result

Theorem (Main theorem)

Let L be a regular language. Then:

- *RSPQ(L) is polynomial if $L \in \mathcal{C}_{tract}$;*
- *RSPQ(L) is NP-complete if $L \notin \mathcal{C}_{tract}$.*

Our main result

Theorem (Main theorem)

Let L be a regular language. Then:

- *RSPQ(L) is polynomial if $L \in \mathcal{C}_{tract}$;*
- *RSPQ(L) is NP-complete if $L \notin \mathcal{C}_{tract}$.*

Theorem (Refinement)

Let L be a regular language. Then:

- *RSPQ(L) is AC^0 if L is finite;*
- *RSPQ(L) is NL-complete if $L \in \mathcal{C}_{tract}$ and L is infinite;*
- *RSPQ(L) is NP-complete if $L \notin \mathcal{C}_{tract}$.*

Our main result

Theorem (Main theorem)

Let L be a regular language. Then:

- *RSPQ(L) is polynomial if $L \in \mathcal{C}_{tract}$;*
- *RSPQ(L) is NP-complete if $L \notin \mathcal{C}_{tract}$.*

Theorem (Refinement)

Let L be a regular language. Then:

- *RSPQ(L) is AC^0 if L is finite;*
- *RSPQ(L) is NL-complete if $L \in \mathcal{C}_{tract}$ and L is infinite;*
- *RSPQ(L) is NP-complete if $L \notin \mathcal{C}_{tract}$.*

$AC^0 \subseteq L \subseteq NL \subseteq P \subseteq NP \subseteq$ PSPACE.

AC^0: definable by a first-order formula.

NL: non deterministic logspace.

NP: non deterministic polynomial time.

Definition

A regular language belongs to $\mathcal{C}_{tract}$ if there is a constant M such that for every words w_l, w_1, w_m, w_2, w_r (w_1, w_2 non empty), it holds

$$w_l w_1^M w_m w_2^M w_r \in L \Rightarrow w_l w_1^M w_2^M w_r \in L$$

Examples of languages not in $\mathcal{C}_{tract}$:

- $L = a^* b c^*$:
  $a^M b c^M \in L$ and $a^M c^M \notin L$.
- $L = (aa)^*$:
  $a(aa)^M a(aa)^M \in L$ and $a(aa)^M (aa)^M \notin L$
- $L = \{\text{words with no occurrence of } aa\}$:
  $(ba)^M b(ab)^M \in L$ and $(ba)^M (ab)^M \notin L$.

Alternative characterization

Definition by regular expressions

$\mathcal{C}_{tract}$ is the class of languages definable by Ψ_{tr}-expressions.

- Ψ_{tr}-terms:
  - $w + \epsilon$ for $w \in \Sigma^*$.
  - $A^{\geqslant k} + \epsilon$ for $A \subseteq \Sigma$.
- Ψ_{tr}-sequence: concatenation of a sequence of terms. The first and last vertices of the sequence are words and the others are Ψ_{tr}-terms;
- Ψ_{tr}-expression: disjunction of Ψ_{tr}-sequences.

Example

- $L = abb^*(ab + \epsilon)a^*ab$
- $L = ab(a^*(a + c)^* + d + \epsilon)(aaa^* + \epsilon)b$

part 2: Conclusions and future work

Concluding remarks and perspectives

- We have charted the tractability frontier of RSPQ.
- A future direction is devoted to:
  - generalize the classification to context-free languages (CFL) for which RSPQ(L) is tractable: $\{a^n b^m | n \neq m\}$.

Concluding remarks and perspectives

- We have charted the tractability frontier of RSPQ.
- A future direction is devoted to:
  - generalize the classification to context-free languages (CFL) for which RSPQ(L) is tractable: $\{a^n b^m | n \neq m\}$.
  - consider special classes of graphs like planar digraphs.

Concluding remarks and perspectives

- We have charted the tractability frontier of RSPQ.
- A future direction is devoted to:
  - generalize the classification to context-free languages (CFL) for which RSPQ(L) is tractable: $\{a^n b^m | n \neq m\}$.
  - consider special classes of graphs like planar digraphs.
  - does it exist a trichotomy for RPQ: AC^0, L-complete, NL-complete ?

Graph queries, from generation to learning

- part 1: gMark for schema-driven generation of graph instances and queries
- part 2: complexity of evaluation of simple regular path queries
- part 3: user-driven inference of regular path queries

Motivation

- Specifying a database query is a challenging task for non-expert users
  - **Unfamiliar** with language formalisms
- In the context of **graph databases**, the problem becomes even harder:
  - There is no clear distinction between instances and **schemas**.
  - The instances do not carry proper **metadata**.
  - The instances are usually of large **size** and difficult to visualize.
- Traditional query specification paradigms for non-expert users e.g., **query by example**[2] become unfeasible.

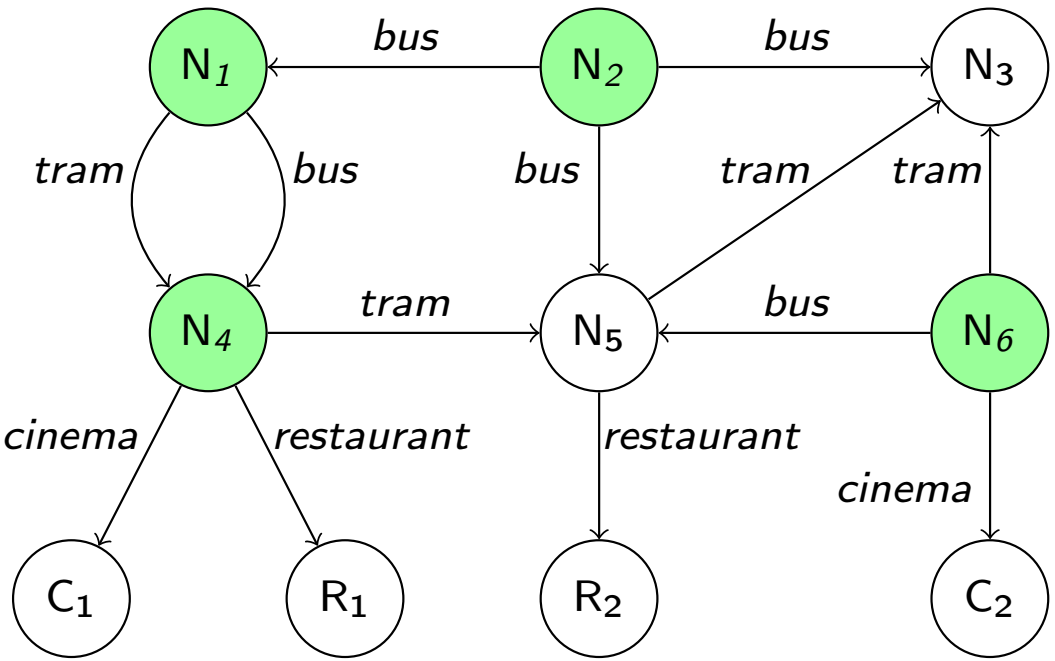
[2] Zloof. Query by example. *AFIPS'75*.

Regular Path queries on graph databases

- We focus on **regular path queries** (pq) that select nodes having at least one path in the language of a given **regular expression**
- Example: "select neighborhoods from which one can reach a cinema via public transportation"

$(tram + bus)^* \cdot cinema$

$N_1 \xrightarrow{tram} N_4 \xrightarrow{cinema} C_1$
$N_2 \xrightarrow{bus} N_1 \xrightarrow{tram} N_4 \xrightarrow{cinema} C_1$
$N_4 \xrightarrow{cinema} C_1$
$N_6 \xrightarrow{cinema} C_2$



Learning path queries on graph databases

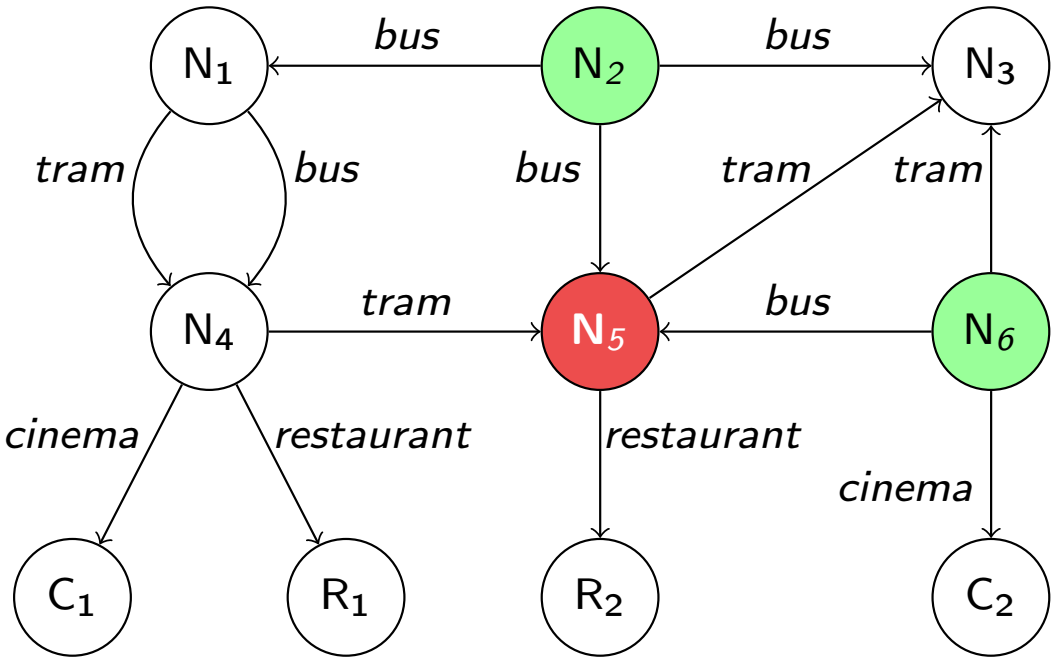
- **Input:** **positive** and **negative** node **examples**
- **Output:** the query that "the user has in mind"

Consistent queries:

$(tram + bus)^* \cdot cinema$

bus

$\vdots$



Contributions

Learning from a set of examples (a fixed set of examples is given)

- General **framework** for learning path queries
- Intractability of **consistency checking** (even for restricted classes)
- **Learnability** (with **abstain**) of pq

Learning from user interactions (the algorithm interacts with the user)

- Characterization of what means for a node to be **informative**
- Practical **strategies** of presenting nodes to the user

Experimental evaluation of our algorithms

- Real **biological** and **synthetic** datasets

Learning from a set of examples

Learning with polynomial time and data

Ideally, we would like a Gold[3]-style learning algorithm:

- **Polynomial**
- **Sound** – return a consistent query or *null* if no such query exists
- **Complete** – able to learn any query from its characteristic sample

Problem:

- Intractability of **consistency checking**
  - **PSPACE-complete** for **general** pq
  - **NP-complete** for **restrictions** (queries of the form $a_1 \cdot \ldots \cdot a_n$)
- Proof techniques from **definability** problems[4] (binary semantics)

---

[3] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 1978.

[4] T. Antonopoulos, F. Neven, and F. Servais. Definability problems for graph query languages. *ICDT*, 2013.

Learning model with abstain

Learning with polynomial time and data

- **Polynomial**
- ~~**Sound** – return a consistent query or *null* if no such query exists~~
- **Complete** – able to learn any query from its characteristic sample

Learning model with abstain

- Always return in **polynomial time**
- If a consistent query cannot be efficiently constructed, the algorithm **abstains** from answering
- If a **polynomial characteristic sample** is provided, the learning algorithm is guaranteed to return the goal query

Learning algorithm

Idea

- for each positive node select the path that "made the user label it"
- construct the disjunction of such paths
- generalize consistently with the examples

Learning algorithm

Idea

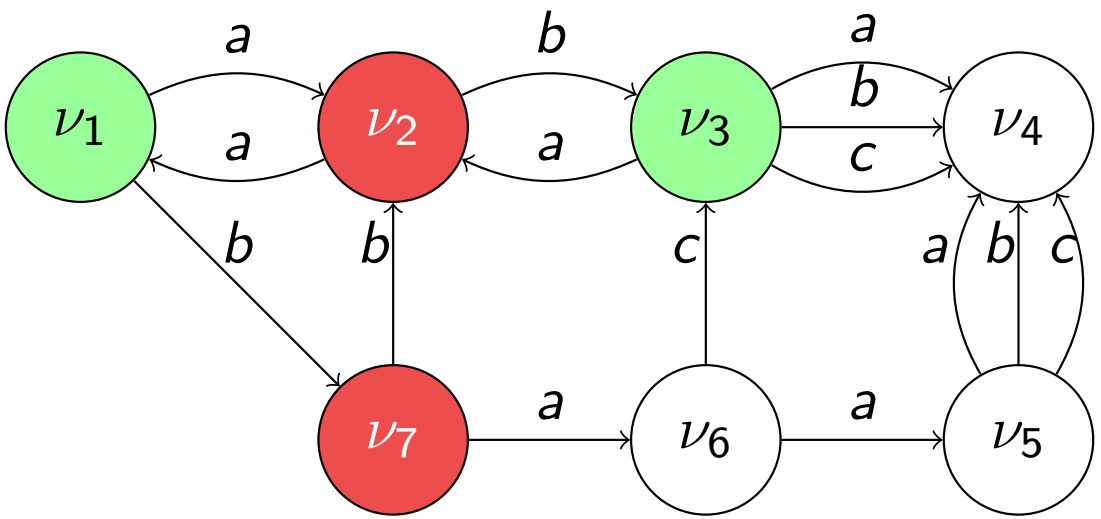
- for each positive node select the path that "made the user label it"
- construct the disjunction of such paths
- generalize consistently with the examples

Step 1 – Selecting smallest consistent paths (SCPs)

- for each positive node select its **smallest consistent path** (**SCP**)

ν_1 – abc

ν_3 – c

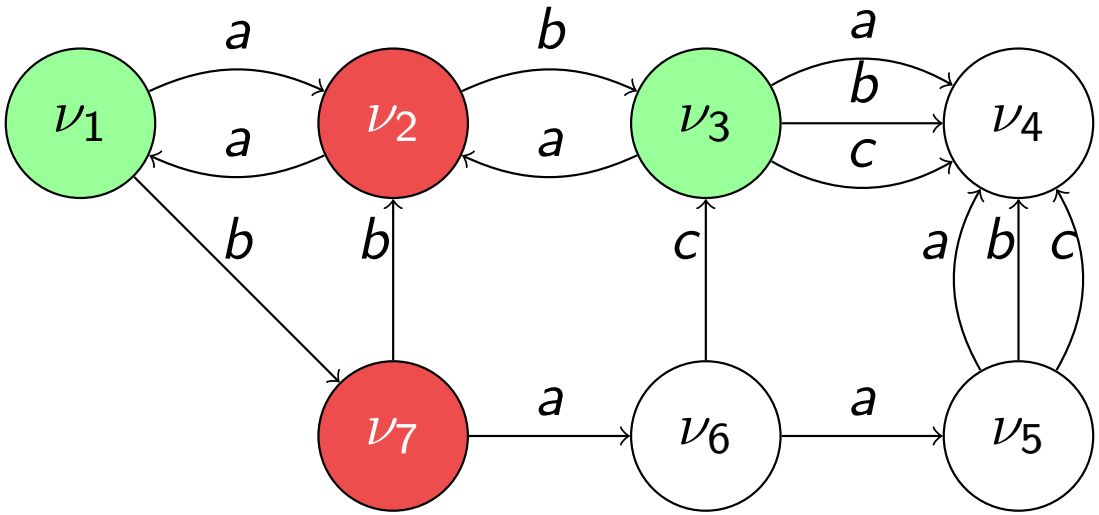


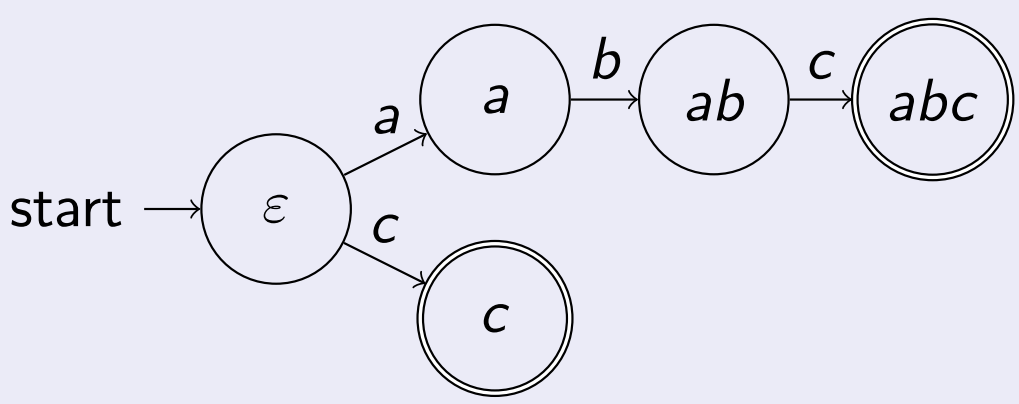
Learning algorithm

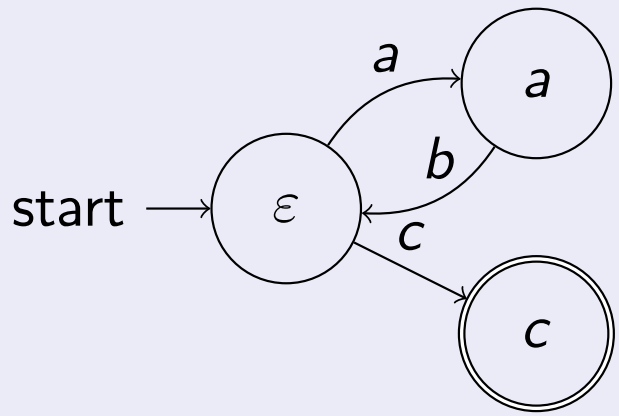
Step 2 – Generalizing SCPs by **state merges** à la **RPNI**[5]

ν_1 – abc

ν_3 – c





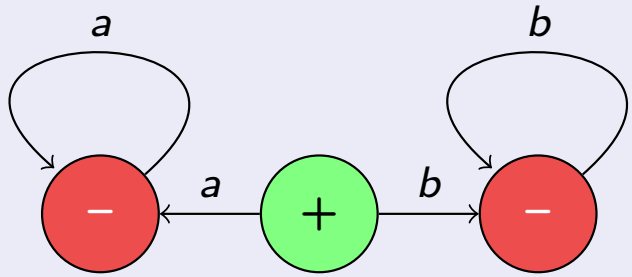


[5] J. Oncina and P. García. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, 1992.

Bound the length of SCPs

Problem 1 – Inconsistent sample

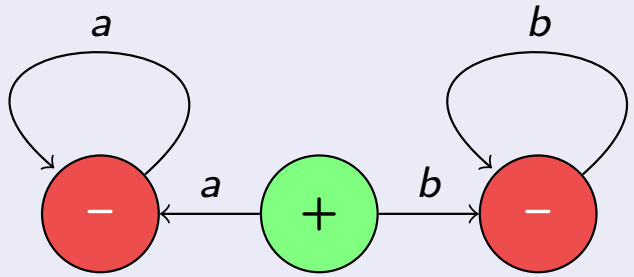
we may enumerate an infinite set of paths and never halt



Bound the length of SCPs

Problem 1 – Inconsistent sample

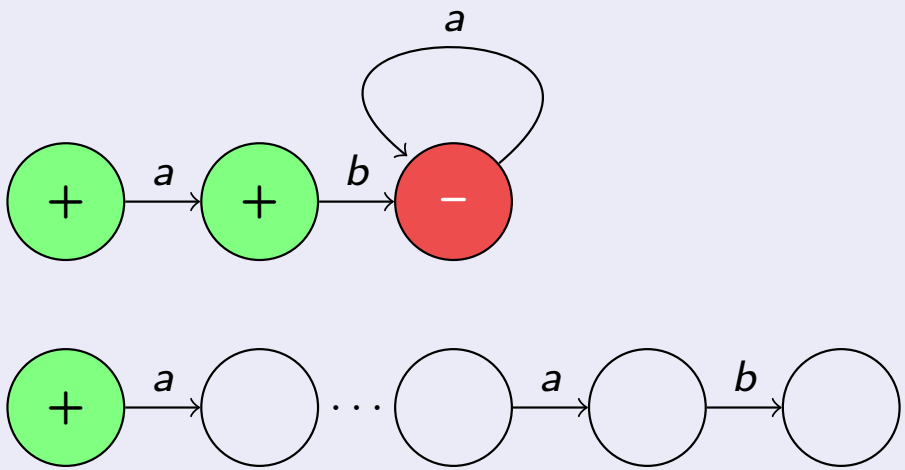
we may enumerate an infinite set of paths and never halt



Problem 2 – Very long SCPs

Goal query: $a^* \cdot b$

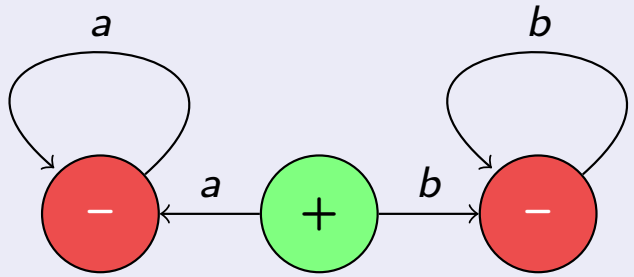
SCPs: b, ab, ~~$aa \ldots ab$~~



Bound the length of SCPs

Problem 1 – Inconsistent sample

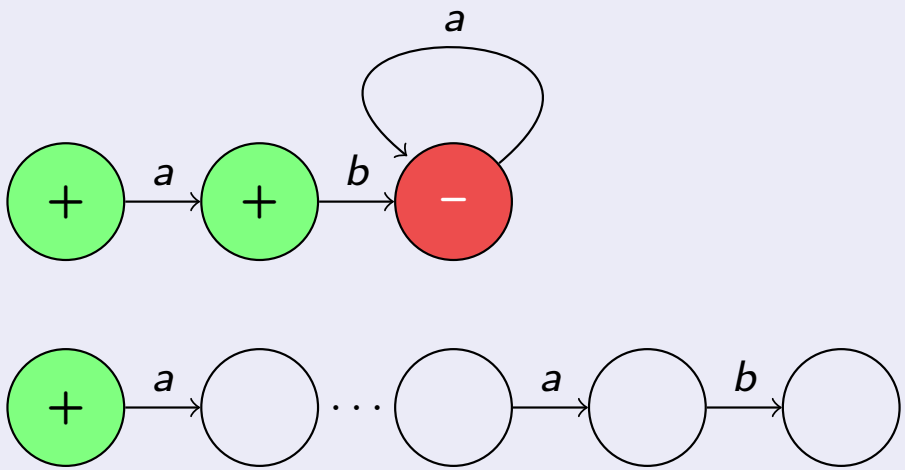
we may enumerate an infinite set of paths and never halt



Problem 2 – Very long SCPs

Goal query: $a^* \cdot b$

SCPs: b, ab, ~~$aa \ldots ab$~~



Learning algorithm

1. select SCPs shorter than a fixed parameter k if they exist
2. generalize SCPs

Learnability result

Choosing the good k

- Let P_+ be the set of paths to be selected in an input sample, the parameter k should be $\geqslant$ than the longest path in P_+.
- The longest path in P_+ is bounded by $2 \times n + 1$ (where n is the number of states in the canonical DFA of the goal)[a].
- Let $\text{pq}^{\leqslant n}$ the path queries with canonical DFAs of at most n states.

[a] J. Oncina and P. García. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, 1992.

Theorem

$\text{pq}^{\leqslant n}$ are **learnable** (with abstain in polynomial time and data) using the learning algorithm with k set to $2 \times n + 1$.

Learning from user interactions

Learning from user interactions

Algorithm

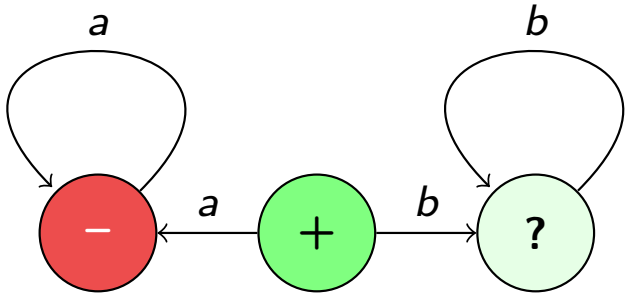
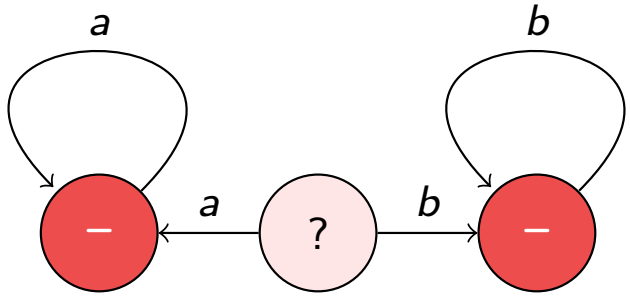
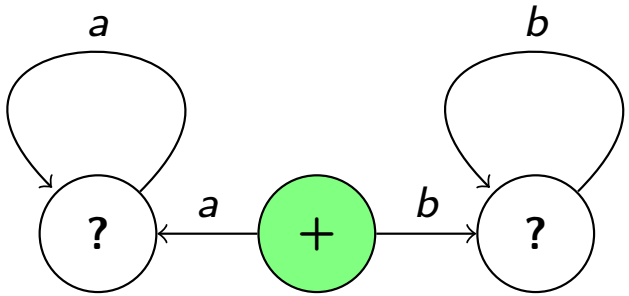
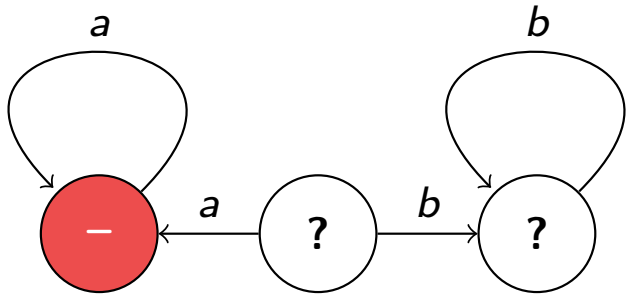
start with an empty sample
while there are informative nodes left
    choose a node according to a strategy
    ask the user to **label** the chosen node
    learn a query q
    if the user is satisfied by the output of q
        return q

Problems

- What means for a node to be informative?
- What is a good strategy of presenting nodes to the user?

Informative and uninformative nodes

A node is **uninformative** with a **label** if labeling it otherwise leads to an inconsistent sample.

Uninformative with +	Uninformative with −
	
Informative nodes	**Informative nodes**
	

Complexity

Deciding whether a node is informative is PSPACE-complete.

Practical strategies

Idea

- Look at k-**paths** – paths of nodes of length bounded by k
- A node is k-**uninformative** if all its k-paths are covered by negatives

  no $\rightarrow$ the node is informative and becomes a candidate next node

  yes $\rightarrow$ the current k does not permit to decide the informativeness

Practical strategies

Idea

- Look at k-**paths** – paths of nodes of length bounded by k
- A node is k-**uninformative** if all its k-paths are covered by negatives

  no → the node is informative and becomes a candidate next node

  yes → the current k does not permit to decide the informativeness

Strategies

1. A **randomly** chosen k-informative node (kR)
2. The node with the **smallest** number of non-covered k-paths (kS)

Setup of experiments

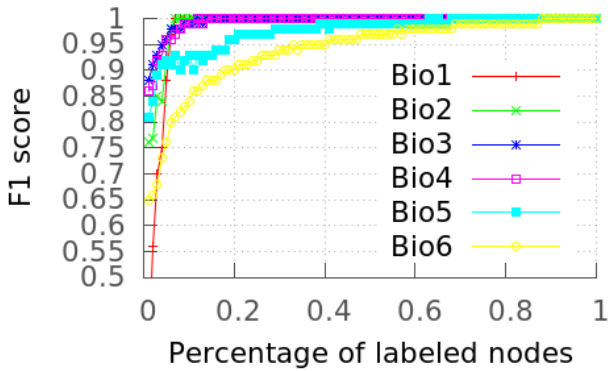
Datasets

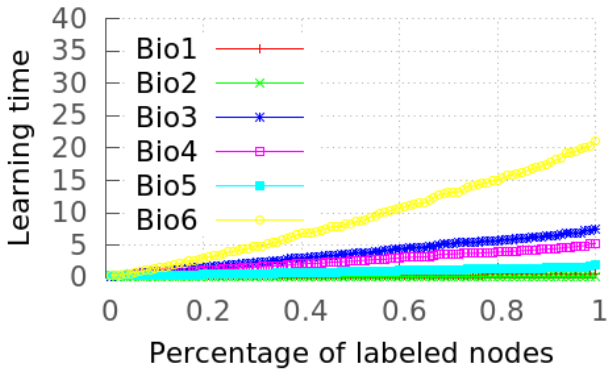
- **Biological** datasets
  - Biological graph of $\approx 3k$ nodes, 6 queries from biological research
- **Synthetic** datasets
  - Generate **scale-free** graphs (as Internet, social and biological graphs)
  - Varying sizes: $10k$, $20k$, $30k$

Experimental settings

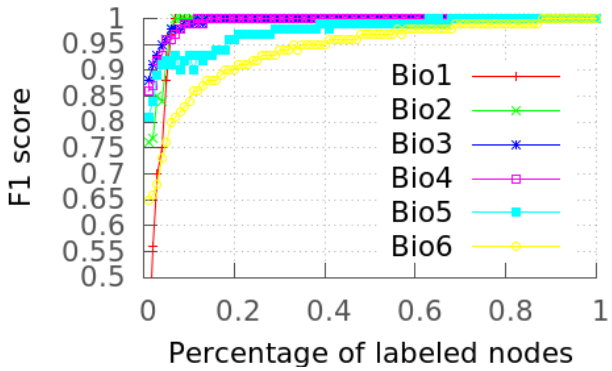
- **Static** experiments
  - Take randomly some nodes, label them, and run algorithm on them
  - Measure the **F1 score** and the **learning time** (s)
- **Interactive** experiments
  - Start with an empty set of examples
  - Measure the **number of examples (+/−)** and the **time** necessary for F1 score 1.

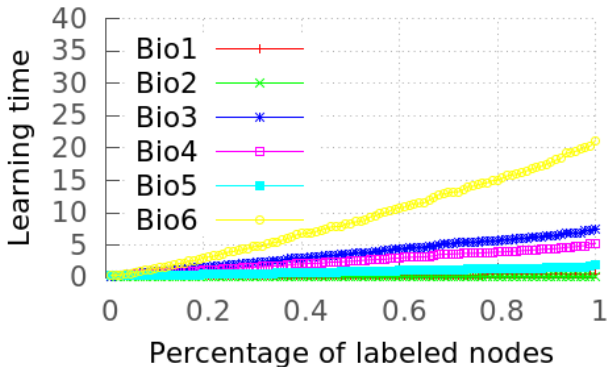
Summary of static experiments

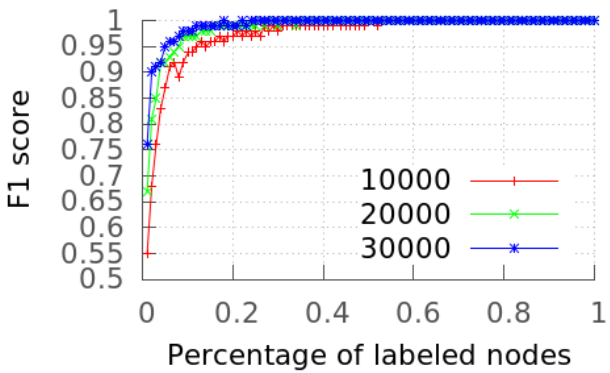


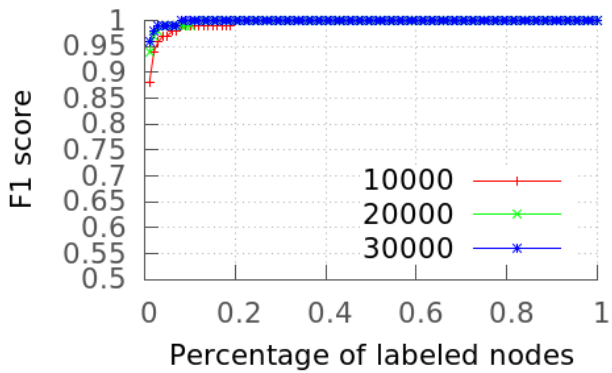


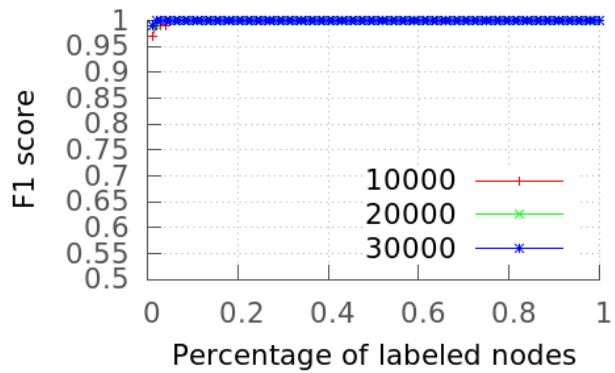
Summary of static experiments

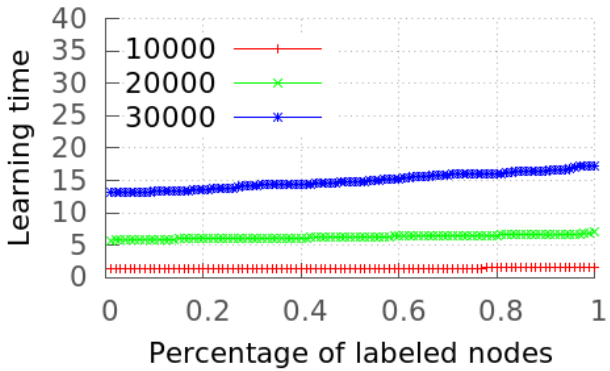


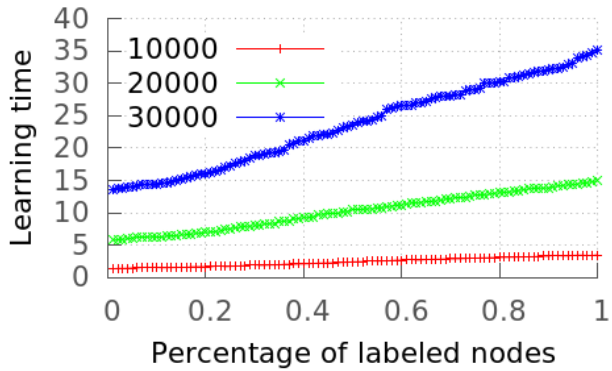


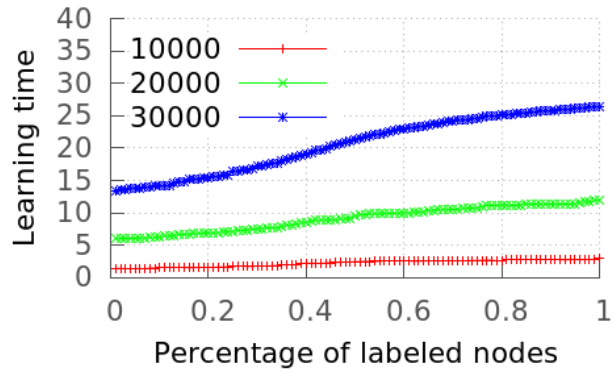












Summary of interactive experiments

Dataset	*Bio query* / *Graph size*	*Labels needed for F1 score = 1* **without interactions**	*Interactive strategy*	*Labels needed for F1 score = 1* **with interactions**	*Time betwe interaction* (*seconds*)
Bio queries	bio_1	7%	kR	0.06%	0.19
			kS	0.06%	0.33
	bio_2	7%	kR	1.78%	0.26
			kS	3.13%	0.48
	bio_3	66%	kR	1.24%	0.34
			kS	1.49%	0.45
	bio_4	12%	kR	1.32%	0.23
			kS	0.22%	0.53
	bio_5	87%	kR	7.7%	3.45
			kS	7.39%	3.79
	bio_6	12%	kR	1.18%	0.24
			kS	0.35%	0.3

Summary of interactive experiments

Dataset	*Bio query* / *Graph size*	*Labels needed for F1 score = 1* **without interactions**	*Interactive strategy*	*Labels needed for F1 score = 1* **with interactions**	*Time betwe interaction* (*seconds*)
Bio queries	bio_1	7%	kR	0.06%	0.19
			kS	0.06%	0.33
	bio_2	7%	kR	1.78%	0.26
			kS	3.13%	0.48
	bio_3	66%	kR	1.24%	0.34
			kS	1.49%	0.45
	bio_4	12%	kR	1.32%	0.23
			kS	0.22%	0.53
	bio_5	87%	kR	7.7%	3.45
			kS	7.39%	3.79
	bio_6	12%	kR	1.18%	0.24
			kS	0.35%	0.3
Synt. query syn_1	10000	51%	kR	0.15%	1.33
			kS	0.17%	1.35
	20000	26%	kR	0.07%	5.83
			kS	0.06%	5.92
	30000	22%	kR	0.04%	13.5
			kS	0.04%	13.95
Synt. query syn_2	10000	20%	kR	0.38%	1.57
			kS	0.36%	1.58
	20000	11%	kR	0.23%	6.63
			kS	0.22%	6.78
	30000	8%	kR	0.17%	15.24
			kS	0.16%	15.38

part 3: Conclusions and future work

Conclusions

- We studied the problem of **learning path queries on graphs**:
  - **Static** – a fixed set of examples is given
    - ⋆ General framework for learning pq
    - ⋆ Intractability of consistency checking
    - ⋆ Learnability (with abstain) of pq
  - **Interactive** – the algorithm interacts with the user
    - ⋆ Characterization of the "informativeness" of a node
    - ⋆ Practical strategies of presenting nodes to the user
- We have validated experimentally our algorithms
  - Real biological graph
  - Synthetic datasets

Future work

- Sample the initial graph and learn on a representative subgraph
- Evaluate our approach for n-ary queries
- Explore new measures of informativeness

Results

- Schema-driven generation of graph instances and queries (PVLDB16, ICDE17, TKDE17, EDBT17) [Bagan et al., 2016, Bagan et al., 2017a, Bagan et al., 2017b, Leuween et al., 2017]
- Graph queries evaluation (PODS13,ongoing) [Bagan et al., 2013, Bagan et al., 2017c]
- Learning graph queries (Data4U14, EDBT15, EDBT15a, TODS16) [Bonifati et al., 2014, Bonifati et al., 2015b, Bonifati et al., 2015a, Bonifati et al., 2016]

Graph Queries: What is left to be done?

Open issues

- Data engineering around RPQ (query optimization)
- RPQ with data values
  - each node in the graph has a data value to test in query filters
- Extension to more sophisticates graph models, such as the Property Graph Model

Open issues

- Data engineering around RPQ (query optimization)
- RPQ with data values
  - each node in the graph has a data value to test in query filters
- Extension to more sophisticates graph models, such as the Property Graph Model

https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model

Thanks for your attention.

References I

[Bagan et al., 2016] Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G. H. L., Lemay, A., and Advokaat, N. (2016).

Generating Flexible Workloads for Graph Databases.

PVLDB, 9(13):1447–1460.

[Bagan et al., 2017a] Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G. H. L., Lemay, A., and Advokaat, N. (In press, 2017a).

gMark: Schema-Driven Generation of Graphs and Queries.

IEEE Trans. on Knowl. Data Eng.

[Bagan et al., 2017b] Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G. H. L., Lemay, A., and Advokaat, N. (To appear, 2017b).

gMark: Schema-Driven Generation of Graphs and Queries (Extended Abstract).

In *Proceedings of the International Conference on Data Engineering ICDE 2017, San Diego, USA.*

[Bagan et al., 2017c] Bagan, G., Bonifati, A., Fletcher, G. H. L., and Kheddouci, H. (In preparation, 2017c).

Labelling Schemes for RPQs and beyond.

References II

[Bagan et al., 2013] Bagan, G., Bonifati, A., and Groz, B. (2013).

A trichotomy for regular simple path queries on graphs.

In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA*, pages 261–272.

[Bonifati et al., 2015a] Bonifati, A., Ciucanu, R., and Lemay, A. (2015a).

Interactive Path Query Specification on Graph Databases.

In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium*, pages 505–508.

[Bonifati et al., 2015b] Bonifati, A., Ciucanu, R., and Lemay, A. (2015b).

Learning Path Queries on Graph Databases.

In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium*, pages 109–120.

[Bonifati et al., 2014] Bonifati, A., Ciucanu, R., Lemay, A., and Staworko, S. (2014).

A Paradigm for Learning Queries on Big Data.

In *Proceedings of the First International Workshop on Bringing the Value of "Big Data" to Users, Data4U@VLDB 2014, Hangzhou, China*.

References III

[Bonifati et al., 2016] Bonifati, A., Ciucanu, R., and Staworko, S. (2016).

Learning Join Queries from User Examples.

ACM Trans. Database Syst., 40(4):24:1–24:38.

[Leuween et al., 2017] Leuween, W. V., Bonifati, A., Fletcher, G. H. L., and Yakovets, N. (To appear, 2017).

Stability notions in synthetic graph generation: a preliminary study.

In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy*.