gMark:
Schema-Driven Generation of Graphs and Queries

Radu Ciucanu

Université Clermont Auvergne

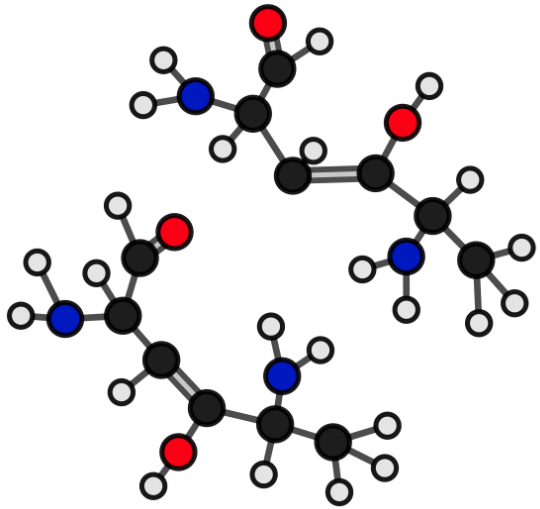
Joint work with colleagues from Univ. Lille, Univ. Lyon, TU Eindhoven

JIRC 2017, Orléans

Why graph data?

Big graph data sets are ubiquitous

- social networks (e.g., LinkedIn, Facebook)
- scientific networks (e.g., Uniprot, PubChem)
- knowledge graphs (e.g., DBPedia)
- ...



Focus is on "things" and their relationships

Why graph databases?

Analytics on big graphs increasingly important

- role discovery in social networks
- identifying interesting patterns in biological networks
- finding important publications in a citation network
- ...

In response to these trends, the past decade has witnessed an explosion of graph data management solutions, e.g.,

- Graph databases such as Neo4j
- Graph analytics platforms such as GraphX
- Triple stores such as Virtuoso
- Datalog engines such as LogicBlox

Why graph database benchmarking?

Benchmark = data sets + query workloads

> *When a field has good benchmarks, we settle debates and the field makes rapid progress.*
>
> D. Patterson (*CACM*, 2012)

Motivated by success stories in relational and XML engineering e.g., TPC and XMark, it is clear that good benchmarks are needed for graph DBs

Graph database benchmarking

LDBC-SNB[1] and WatDiv[2] are current leaders in graph DBMS benchmarking

- LDBC is a fixed-schema and fixed-queries benchmark targeting focused stress-testing of query engineering choke-points
  - social network scenario
- WatDiv is a schema-driven workload-based benchmark targeting broad coverage of query features
  - default schema is products and users scenario

[1] Erling, Averbuch, Larriba-Pey, Chafi, Gubichev, Prat, Pham, and Boncz: *The LDBC social network benchmark: Interactive workload*. SIGMOD'15.

[2] Aluç, Hartig, Özsu, and Daudjee: *Diversified stress testing of RDF data management systems*. ISWC'14.

Synthetic graph and workload generation with gMark

We present gMark, an open-source[1] framework for generation of synthetic graphs and workloads.

Given a graph schema, gMark

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

[1] https://github.com/graphMark/gmark

Why gMark?

We adopt successful aspects of the state of the art

Like WatDiv (and unlike LDBC), gMark is schema-driven,

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

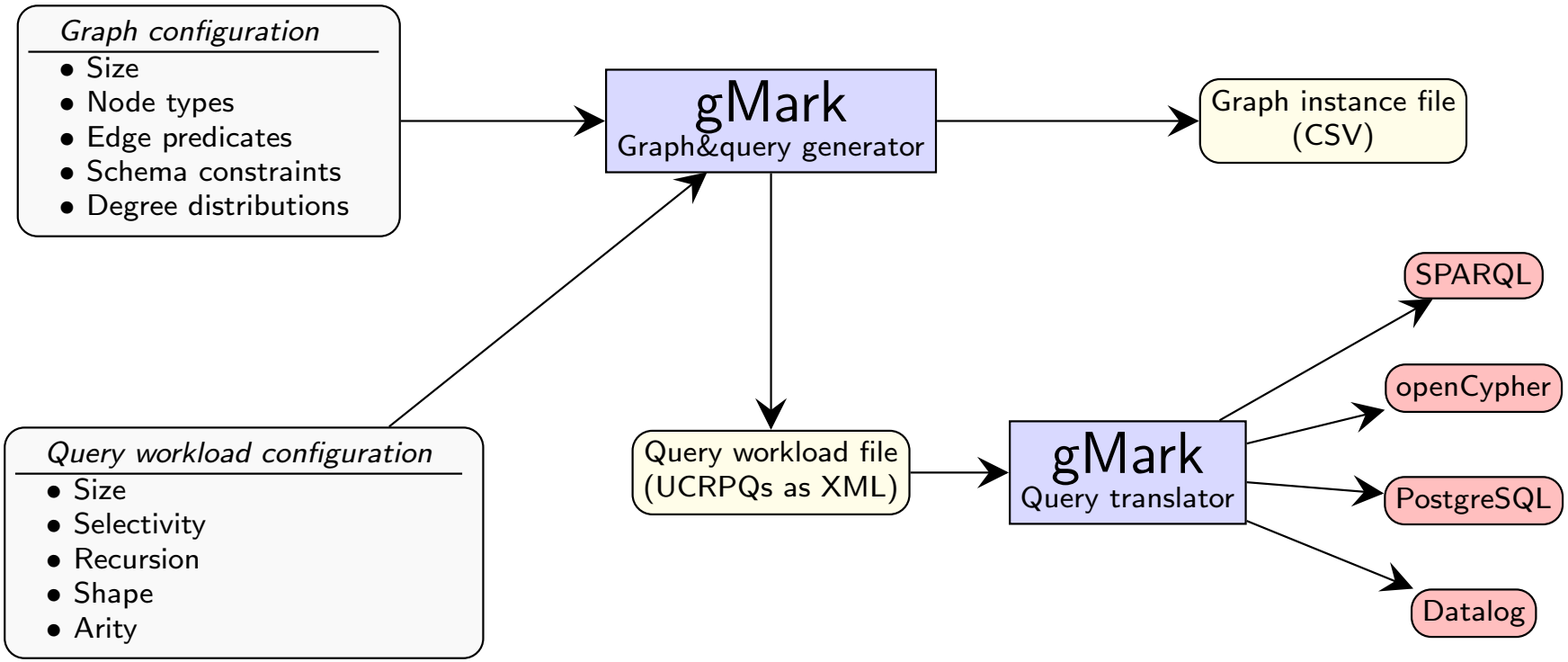
Like LDBC (and unlike WatDiv), gMark supports focused stress-testing of query engineering choke-points, through fine control of query selectivities.

Why gMark?

Unlike both WatDiv and LDBC, gMark

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

Overview of the gMark workflow

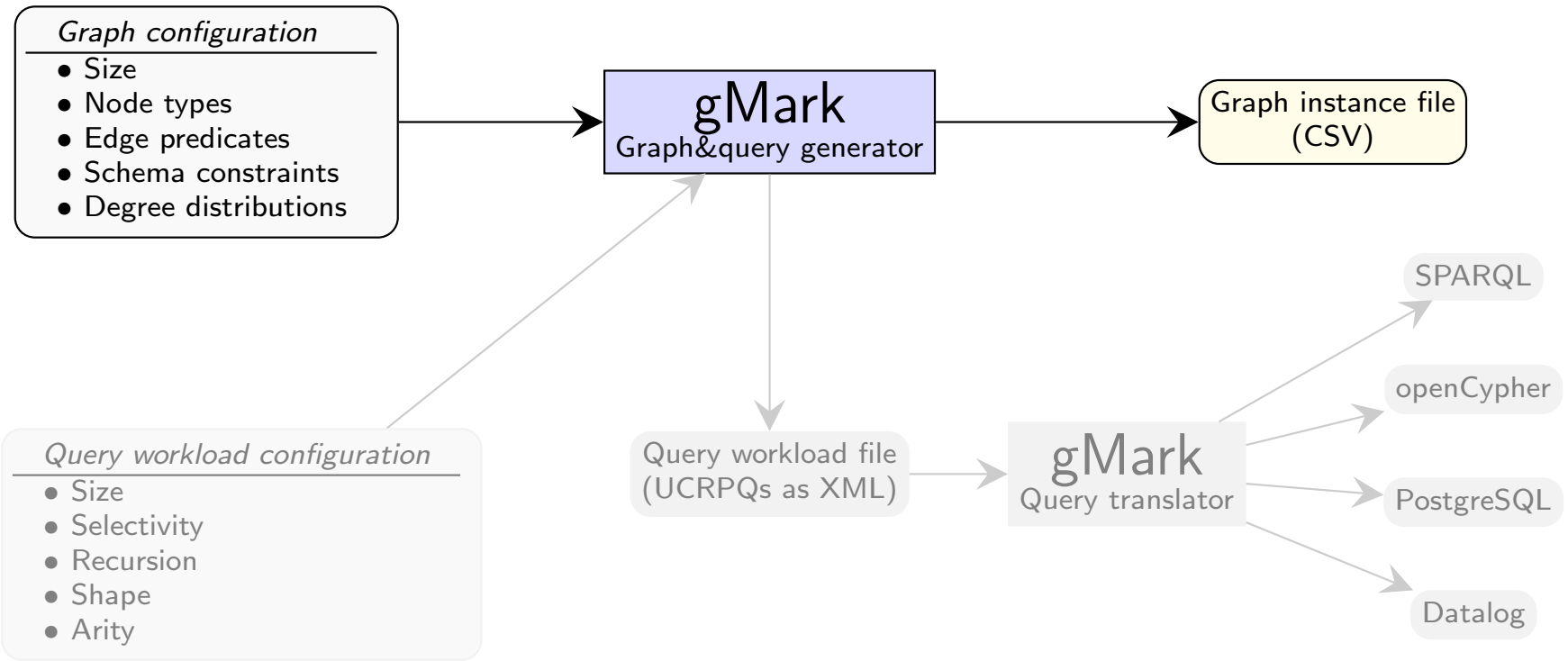


gMark: Schema-Driven Generation of Graphs and Queries

1. Graph Generation
2. Query Generation
3. Scalability Study of Current Graph Databases
4. Evolving Graph Generation

gMark: Schema-Driven Generation of Graphs and Queries

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`

- **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.*
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 a 'best-effort' strategy in instance generation ($\mathcal{O}(n)$), 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 adapted the scenarios of 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

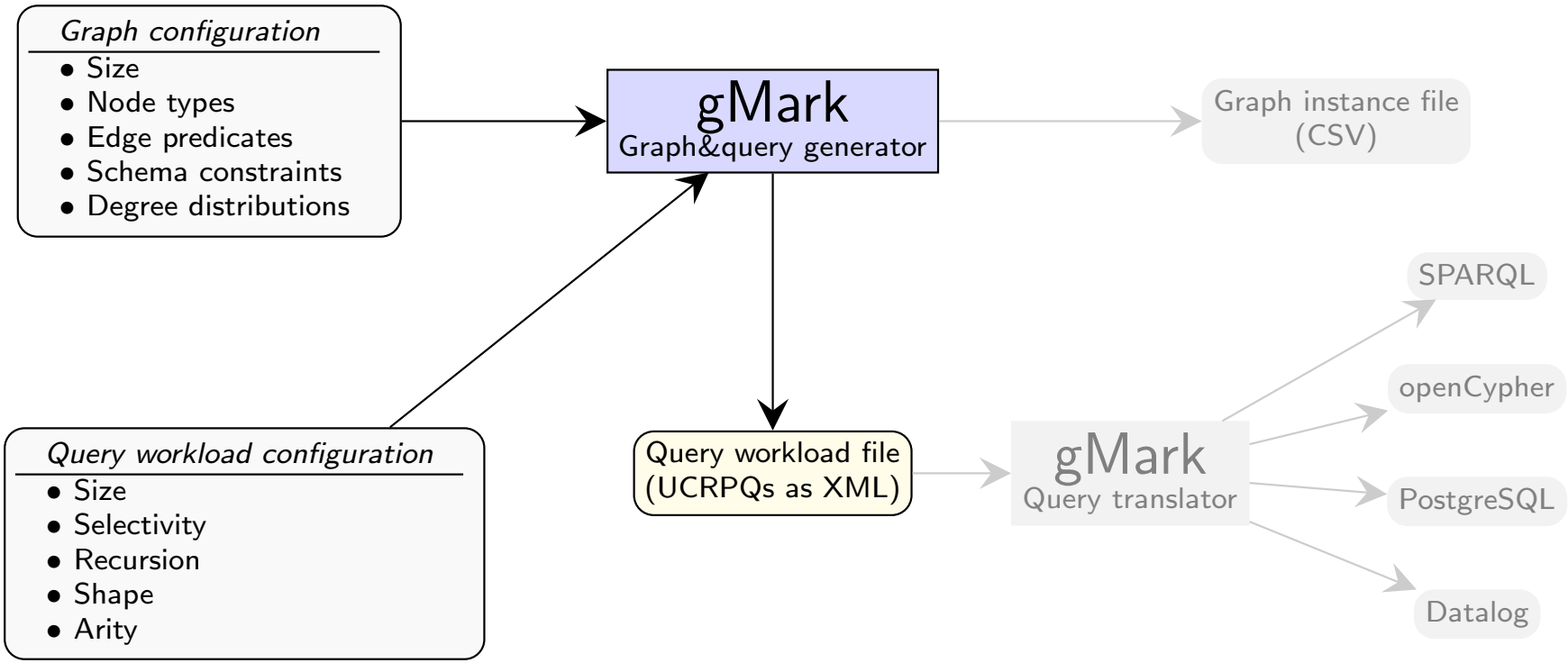
	100K	1M	10M	100M
`Bib`	0m0.057s	0m0.638s	0m8.344s	1m28.725s
`LSN`	0m0.225s	0m1.451s	0m23.018s	3m11.318s
`WD`	0m2.163s	0m25.032s	4m10.988s	113m31.078s
`SP`	0m0.638s	0m7.048s	1m28.831s	15m23.542s

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`)

gMark: Schema-Driven Generation of Graphs and Queries

gMark query generation



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)

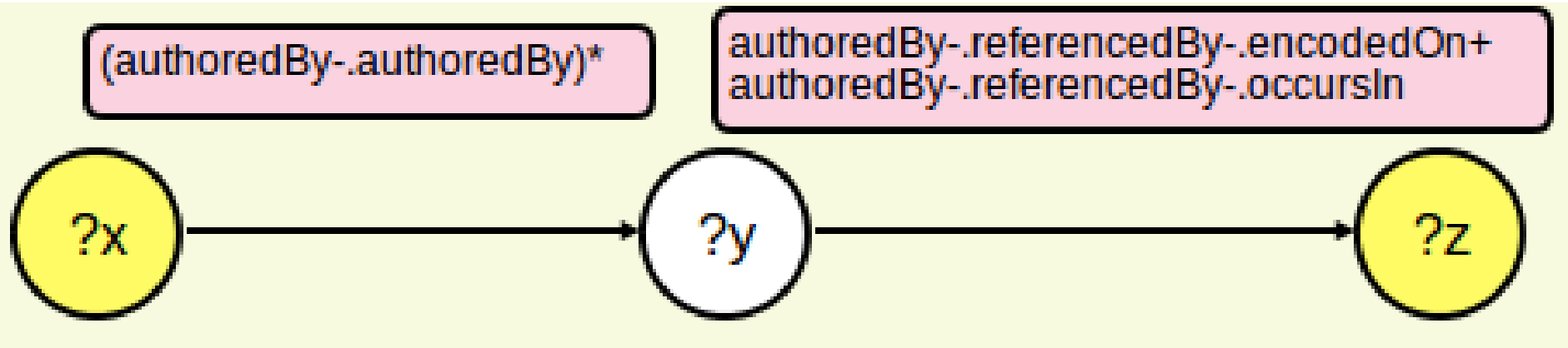
UCRPQ includes **recursive queries** (via the Kleene star *), with applications in social networks, bioinformatics, etc.

gMark generates UCRPQ → the first synthetic workload generator to support recursive queries (and their translation in concrete syntaxes).

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)

#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.
- **Shape**: chain, star, cycle, star-chain.
- **Arity**: arbitrary (including 0 i.e., Boolean).

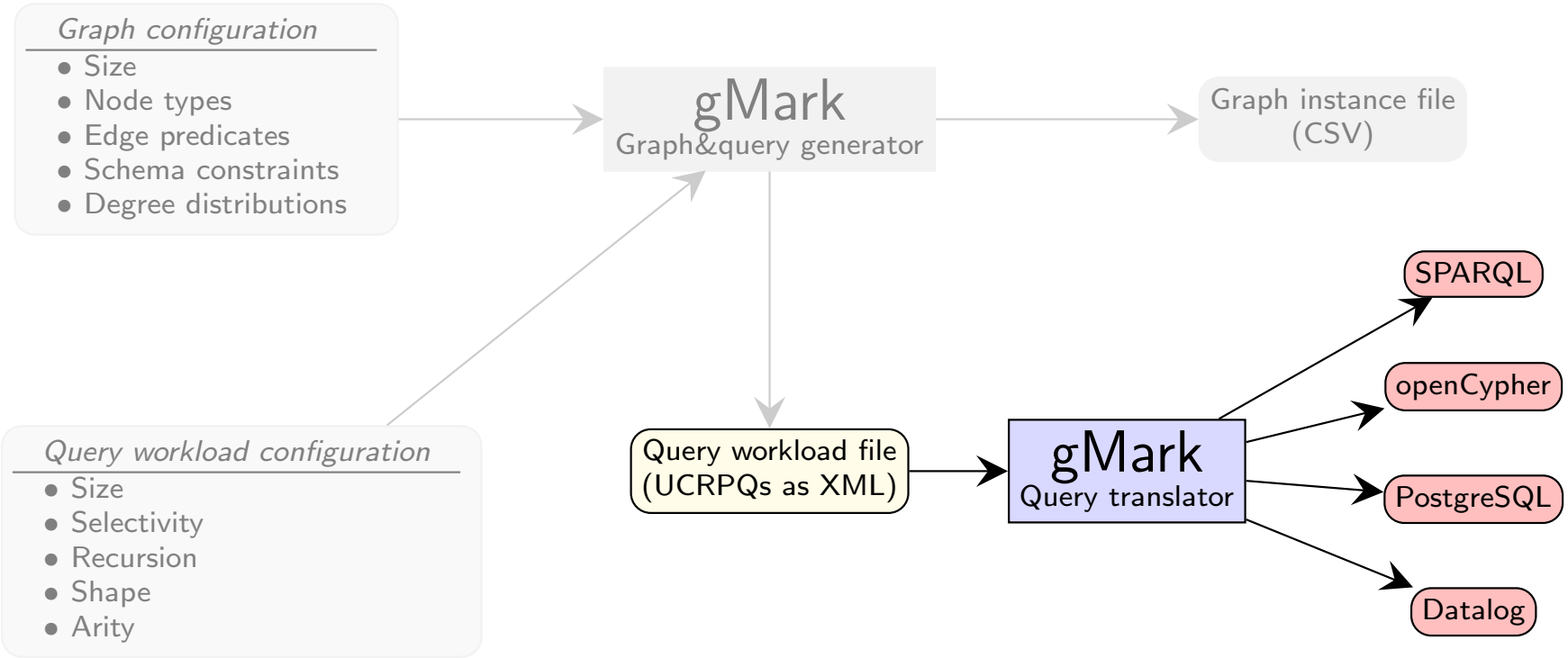
The graph configuration is also input to the query generator.

Selectivity estimation quality of gMark

- Given a binary query Q and a graph G, we assume that $|Q(G)| = \mathcal{O}(\beta \times |nodes(G)|^{\alpha})$.
- α is the selectivity value (0–constant, 1–linear, 2–quadratic).
- Assigning selectivities required us to develop a selectivity algebra for instance-independent reasoning over query behavior.
- Experiments confirmed the assumption and the estimation quality.

	Constant	*Linear*	*Quadratic*
LSN	0.200 ± 0.417	1.189 ± 0.261	2.032 ± 0.059
Bib	0.003 ± 0.010	0.921 ± 0.122	1.405 ± 0.337
WD	0.016 ± 0.044	1.427 ± 0.392	2.004 ± 0.022
SP	0.074 ± 0.130	1.064 ± 0.034	2.034 ± 0.295

gMark query translator



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,
.....
```

* openCypher disallows Kleene star above concatenation or inverses.

Scalability of gMark workload generation

On a laptop, gMark 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 requires $\sim 0.1s$.

gMark: Schema-Driven Generation of Graphs and Queries

State-of-the-art graph DBMSs

We studied query evaluation performance of four mainstream graph DBMSs:

- **P**: PostgreSQL (SQL:1999 recursive views)
- **S**: a popular SPARQL query engine (SPARQL 1.1)
- **G**: a native graph database (openCypher)
- **D**: a modern Datalog engine (Datalog)

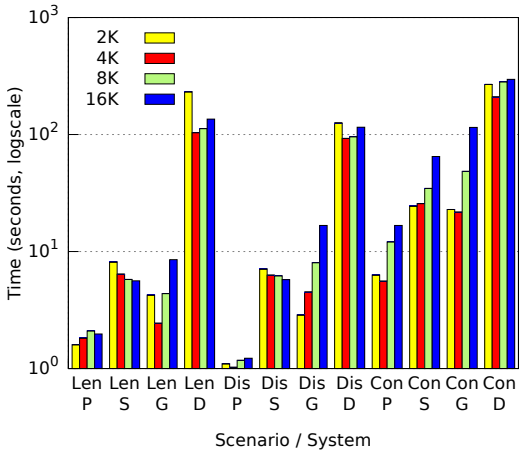
Scalability on non-recursive query workloads

Query execution times for diverse graph sizes and query workloads:

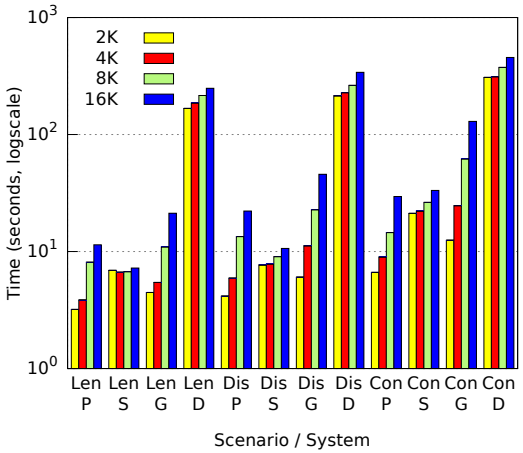
– *Len* (varying path lengths, 1 disjunct, 1 conjunct)

– *Dis* (multiple disjuncts, 1 conjunct)

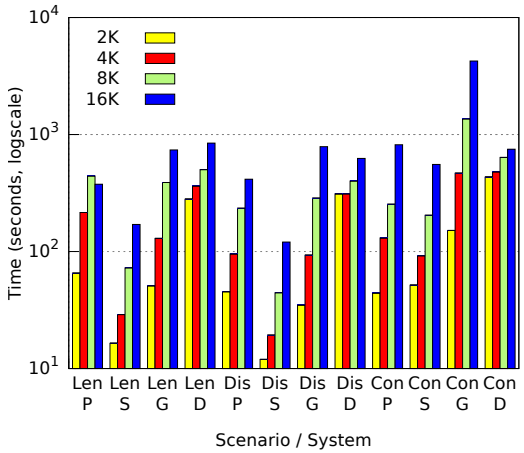
– *Con* (multiple conjuncts and disjuncts)



Constant queries



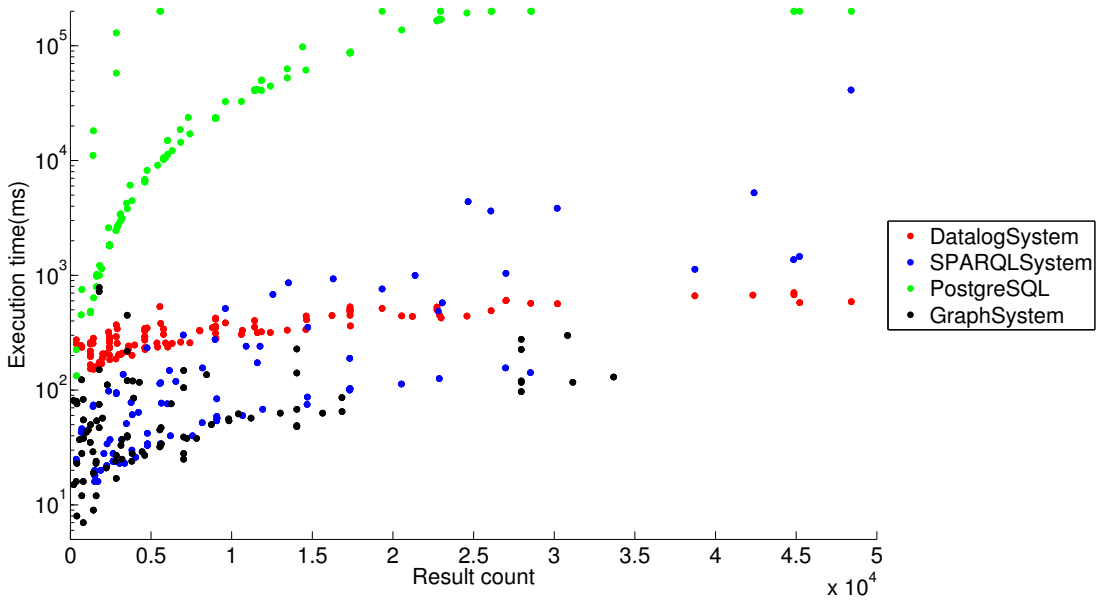
Linear queries



Quadratic queries

Scalability on recursive query workloads

Query execution times for simple recursive queries on various small graph sizes (from 2K to 32K nodes):



gMark: Schema-Driven Generation of Graphs and Queries

Motivation

Graphs are naturally **evolving over time** e.g.,

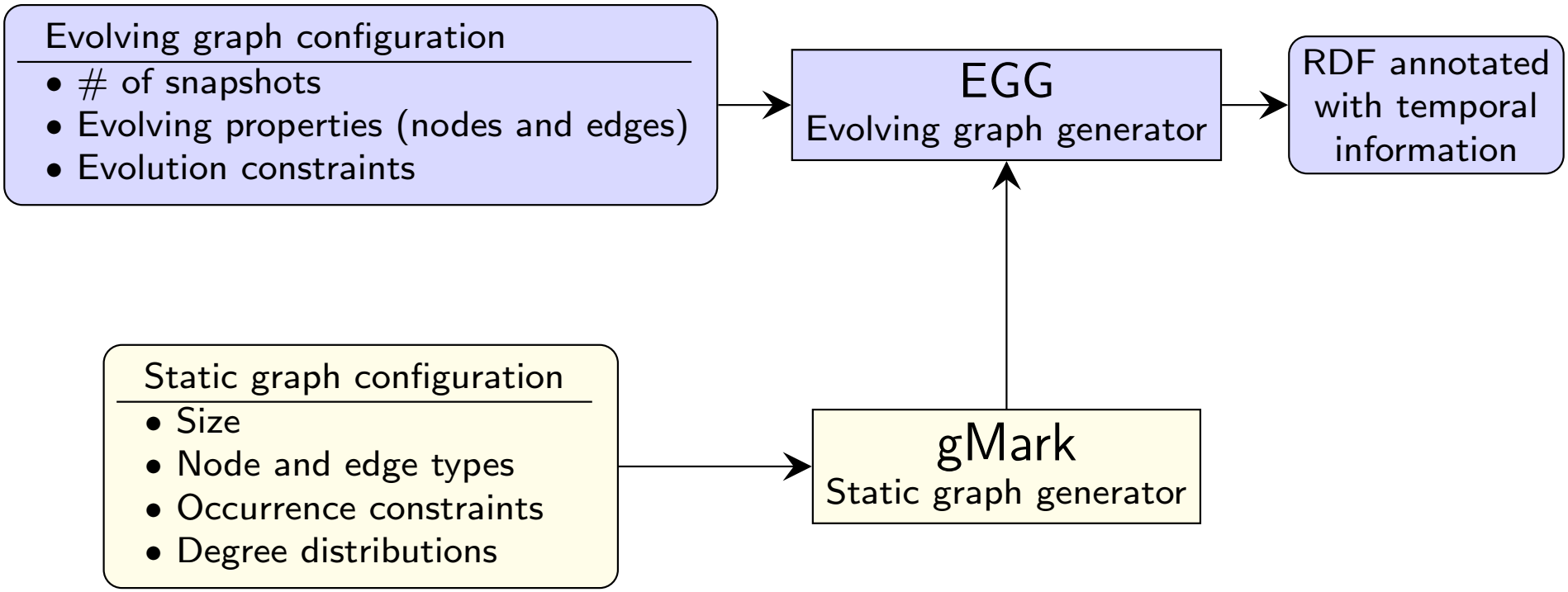
- Nodes and edges have properties whose values change among consecutive snapshots
- Nodes and edges may exist only during specific time intervals

Idea: use gMark to generate schema-driven graphs and enrich them with time-evolving properties

gMark + time-evolving properties = EGG[1]

[1] Open-source: `https://github.com/karimalami7/EGG`

EGG: Evolving Graph Generator



Example

Parameter	**Description**
Size	▷ e.g., 10M
Node types	▷ e.g., `city`, `hotel`
Edge predicates	▷ e.g., `train`, `contains`
Schema constraints	▷ e.g., 10% of all nodes are `cities`
Degree distributions	▷ e.g., the # of hotels in a city follows a Zipfian distribution

Evolving properties:

- `city: weather, qAir`
- `hotel: star, availableRooms, hotelPrice`
- `train: trainPrice`

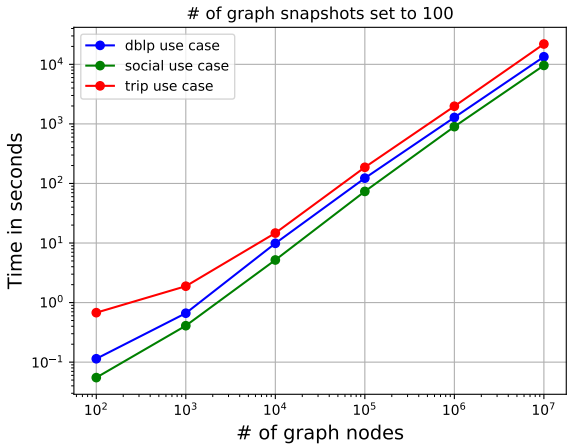
Each graph snapshot corresponds to a day.

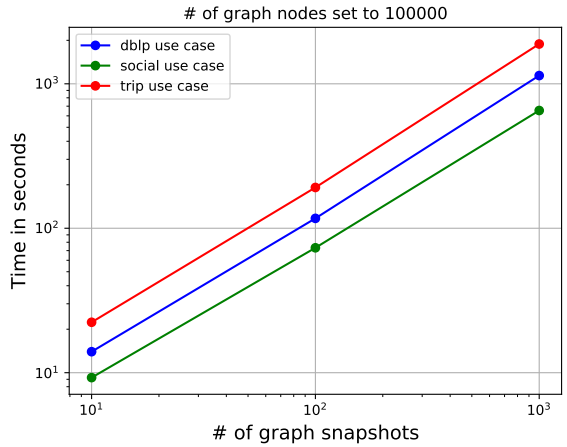
Example

Type	Property	Description
city	weather	unordered qualitative, has three possible values {sunny, cloudy, rainy} successors of sunny: sunny and cloudy.
	qAir	ordered qualitative, has ten possible values from 1 to 10; can increment or decrement by 1 between two consecutive snapshots.
hotel	star	ordered qualitative, has values from 1 to 5, it changes every 365 snapshots with 1% probability, by one position at most
	availableRooms	discrete quantitative, has values in [1,100]; the offset is set to [-15,15]
	hotelPrice	continuous quantitative, dependent on star for domain and on availableRooms for evolution ▷ e.g., for node x of type hotel: if star(x)=3, then hotelPrice(x)∈[50,100] if availableRooms(x) ↑, then hotelPrice(x) ↓ if availableRooms(x) ↓, then hotelPrice(x) ↑.

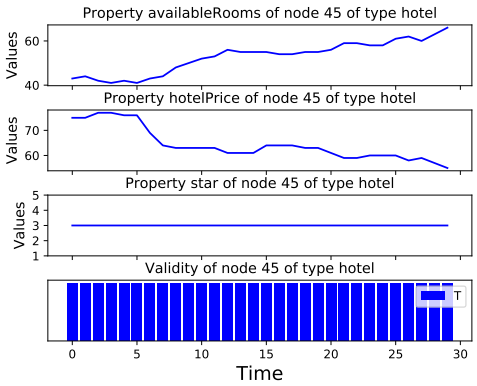
Summary of EGG contributions

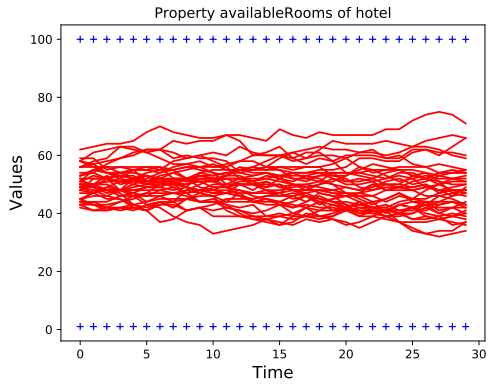
- Linear-time generation algorithm

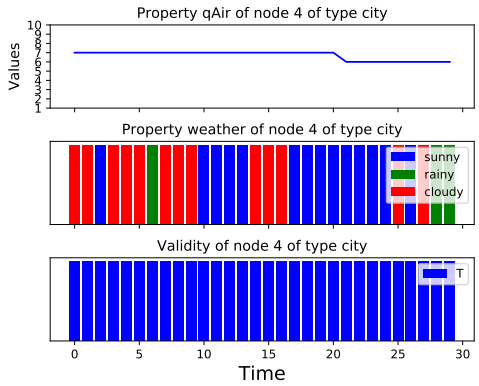




- Visualization module to emphasize the accuracy of EGG





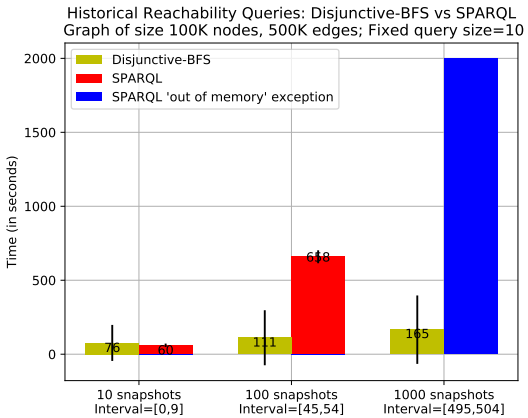


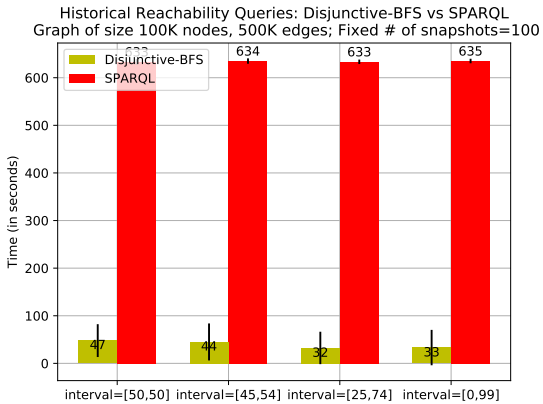
Summary of EGG contributions

- Storage format based on RDF named graphs to decouple static and evolving parts of the graphs e.g.,

```
ns1:G31 { <hotel:27> ns2:hasProperty <Property:availableRooms> }
ns1:snapshot9 { ns1:G31 ns3:value "57" }
```

- Evaluation of historical reachability queries[1] on top of EGG:
  - A baseline implementation in SPARQL on top of Apache Jena
  - Disjunctive-BFS: dynamic programming approach[1]





[1] K. Semertzidis, K. Lillis, E. Pitoura. *TimeReach: Reachability Queries on Evolving Graphs*. EDBT'15.

Conclusions

Conclusions

- gMark[1]
  - schema-driven graph and query-workload generator
  - finely controlled query workload-centered approach, featuring instance-independent selectivity estimation
  - translation to SPARQL, openCypher, SQL, Datalog
  - discovery of the poor performance of existing graph DBMS on evaluating a basic class of graph queries i.e., regular path queries

- EGG[2]
  - evolving graph generator extending the gMark graphs with properties that evolve over time
  - storage format using RDF named graphs to reduce redundancy
  - easy to use to empirically evaluate evolving graph processing systems

[1] `https://github.com/graphMark/gmark`
[2] `https://github.com/karimalami7/EGG`

gMark & EGG papers

- Bagan, Bonifati, Ciucanu, Fletcher, Lemay, Advokaat
  gMark: Schema-Driven Generation of Graphs and Queries
  TKDE'17 full paper
  ICDE'17 extended abstract

- Bagan, Bonifati, Ciucanu, Fletcher, Lemay, Advokaat
  Generating Flexible Workloads for Graph Databases
  VLDB'16 demo

- Alami, Ciucanu, Mephu Nguifo
  EGG: A Framework for Generating Evolving RDF Graphs
  ISWC'17 demo

- Alami, Ciucanu, Mephu Nguifo
  Synthetic Graph Generation from Finely-Tuned Temporal Constraints
  TD-LSG @ PKDD/ECML'17