gMark: Schema-driven data and workload generation for graph databases

George Fletcher
TU Eindhoven

Joint work with
G. Bagan (Lyon), A. Bonifati (Lyon), R. Ciucanu (Oxford),
A. Lemay (Lille), and N. Advokaat (Eindhoven)

8th LDBC TUC Meeting
Redwood Shores, California
23 June 2016
We present gMark, an open-source framework for generation of synthetic graphs and workloads.

gMark has been designed to tailor diverse graph data management scenarios, which are often driven by query workloads.
We present **gMark**, an open-source framework for generation of synthetic graphs and workloads.

gMark has been designed to tailor diverse graph data management scenarios, which are often driven by query workloads.

For example

- multi-query optimization,
- mapping discovery and query rewriting in data integration systems,
- workload-driven graph database physical design,

and, in general, flexible specification and generation of diverse workloads addressing particular experimental studies.

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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, gMark is schema-driven,

- allowing finely tailored graph instances for specific application domains; and,

- allowing tightly controlled generation of query workloads.

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Why gMark?

We adopt successful aspects of the state of the art

For example, like the Waterloo Diversity Benchmark, 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, gMark supports focused stress-testing of query optimization choke-points, through fine control of query parameters such as selectivities.

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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.

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Overview of the gMark workflow

**Graph configuration**
- Size
- Node types
- Edge predicates
- Schema constraints
- Degree distributions

**Query workload configuration**
- Size
- Selectivity
- Recursion
- Shape
- Arity

---

**gMark**
Graph&query generator

**Graph instance file (CSV)**

---

**gMark**
Query translator

**Query workload file (UCRPQs as XML)**

---

- SPARQL
- openCypher
- PostgreSQL
- Datalog

---

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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 authoredBy author is Gaussian with parameters $\mu = 3, \sigma = 1$

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Graph configurations: Uniprot schema

<table>
<thead>
<tr>
<th>Node type</th>
<th>Constr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene</td>
<td>35%</td>
</tr>
<tr>
<td>protein</td>
<td>31%</td>
</tr>
<tr>
<td>author</td>
<td>20%</td>
</tr>
<tr>
<td>citation</td>
<td>10%</td>
</tr>
<tr>
<td>organism</td>
<td>1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge predicate</th>
<th>Constr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>authoredBy</td>
<td>64%</td>
</tr>
<tr>
<td>encodedOn</td>
<td>6%</td>
</tr>
<tr>
<td>referencedBy</td>
<td>3%</td>
</tr>
<tr>
<td>occursIn</td>
<td>2%</td>
</tr>
</tbody>
</table>

Node types

Edge predicates

<table>
<thead>
<tr>
<th>source type</th>
<th>predicate</th>
<th>target type</th>
<th>In-distr.</th>
<th>Out-distr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>encodedOn</td>
<td>gene</td>
<td>Zipfian</td>
<td>Gaussian</td>
</tr>
<tr>
<td>protein</td>
<td>occursIn</td>
<td>organism</td>
<td>Zipfian</td>
<td>Uniform</td>
</tr>
<tr>
<td>protein</td>
<td>referencedBy</td>
<td>citation</td>
<td>Zipfian</td>
<td>Gaussian</td>
</tr>
<tr>
<td>citation</td>
<td>authoredBy</td>
<td>author</td>
<td>Zipfian</td>
<td>Gaussian</td>
</tr>
</tbody>
</table>

In- and out-degree distributions

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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.*
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, i.e., it attempts to achieve the exact values of the input parameters and relaxes them whenever this is not possible.
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

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

_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

**Graph configuration**
- Size
- Node types
- Edge predicates
- Schema constraints
- Degree distributions

**Query workload configuration**
- Size
- Selectivity
- Recursion
- Shape
- Arity

---

**gMark**
Graph&query generator

**Query workload file**
(UCRPQs as XML)

**gMark**
Query translator

---

Graph instance file (CSV)

- SPARQL
- openCypher
- PostgreSQL
- Datalog

---

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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 Neo4j’s openCypher
– Well understood theoretical properties (e.g., polynomial data complexity)

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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 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 schema-driven tool to support recursive queries and their generation in concrete syntaxes.

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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*

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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
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) ← (?x, (a−·a)*, ?y), (?y, (a−·r−·e + a−·r−·o), ?z)

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

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

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

<table>
<thead>
<tr>
<th>#rules</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#conjuncts</td>
<td>2</td>
</tr>
<tr>
<td>#disjuncts</td>
<td>1, 2</td>
</tr>
<tr>
<td>path length</td>
<td>2, 3, 3</td>
</tr>
</tbody>
</table>

https://github.com/graphMark/gmark
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.
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.
Approach

1. Prepare “schema” and “selectivity” graphs
Approach

1. Prepare “schema” and “selectivity” graphs
2. For each query to be generated:
   2.1 Generate query skeleton
   2.2 Assign selectivity class to each predicate
   2.3 Instantiate each predicate
Schema-driven workload generation

Approach

1. Prepare “schema” and “selectivity” graphs
2. For each query to be generated:
   2.1 Generate query skeleton
   2.2 Assign selectivity class to each predicate
   2.3 Instantiate each predicate

Assigning selectivities required us to develop a non-trivial infrastructure for instance-independent reasoning over query behavior, based on a Selectivity Algebra.

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Selectivity estimation quality of gMark

- Given a binary query $Q$ and a graph $G$, we assume that $|Q(G)| = \mathcal{O}(|\text{nodes}(G)|^\alpha)$.
- $\alpha$ is the selectivity value (0–constant, 1–linear, 2–quadratic).
Selectivity estimation quality of gMark

- Given a binary query $Q$ and a graph $G$, we assume that $|Q(G)| = \mathcal{O}(|\text{nodes}(G)|^\alpha)$.
- $\alpha$ is the selectivity value (0–constant, 1–linear, 2–quadratic).
- Experiments confirmed the assumption and the estimation quality.

<table>
<thead>
<tr>
<th></th>
<th>Constant</th>
<th>Linear</th>
<th>Quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSN-Len</td>
<td>0.200±0.417</td>
<td>1.189±0.261</td>
<td>2.032±0.059</td>
</tr>
<tr>
<td>LSN-Dis</td>
<td>0.182±0.364</td>
<td>1.325±0.318</td>
<td>2.046±0.074</td>
</tr>
<tr>
<td>LSN-Con</td>
<td>0.190±0.391</td>
<td>1.244±0.326</td>
<td>2.017±0.032</td>
</tr>
<tr>
<td>LSN-Rec</td>
<td>0.196±0.409</td>
<td>1.090±0.492</td>
<td>1.564±0.889</td>
</tr>
<tr>
<td>Bib-Len</td>
<td>0.003±0.010</td>
<td>0.921±0.122</td>
<td>1.405±0.337</td>
</tr>
<tr>
<td>Bib-Dis</td>
<td>0.000±0.000</td>
<td>0.995±0.012</td>
<td>1.607±0.261</td>
</tr>
<tr>
<td>Bib-Con</td>
<td>0.023±0.029</td>
<td>0.986±0.112</td>
<td>1.409±0.296</td>
</tr>
<tr>
<td>Bib-Rec</td>
<td>0.100±0.316</td>
<td>0.982±0.073</td>
<td>1.493±0.335</td>
</tr>
<tr>
<td>WD-Len</td>
<td>0.016±0.044</td>
<td>1.427±0.392</td>
<td>2.004±0.022</td>
</tr>
<tr>
<td>WD-Dis</td>
<td>0.009±0.022</td>
<td>1.412±0.380</td>
<td>1.999±0.014</td>
</tr>
<tr>
<td>WD-Con</td>
<td>-0.010±0.026</td>
<td>1.540±0.495</td>
<td>1.750±0.708</td>
</tr>
<tr>
<td>WD-Rec</td>
<td>0.587±0.830</td>
<td>-</td>
<td>1.976±0.012</td>
</tr>
<tr>
<td>SP</td>
<td>0.074±0.130</td>
<td>1.064±0.034</td>
<td>2.034±0.295</td>
</tr>
</tbody>
</table>

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
gMark query translator

Graph configuration
- Size
- Node types
- Edge predicates
- Schema constraints
- Degree distributions

Query workload configuration
- Size
- Selectivity
- Recursion
- Shape
- Arity

---

Graph instance file (CSV)

Query workload file (UCRPQs as XML)

---

gMark
Graph&query generator

---

gMark
Query translator

- SPARQL
- openCypher
- PostgreSQL
- Datalog

---

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Query translation

UCRPQ: (?x, ?z) ← (?x, (a→a)*, ?y), (?y, (a→r→e + a→r→o), ?z)
# Query translation

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

<table>
<thead>
<tr>
<th>SPARQL</th>
<th>openCypher</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PREFIX : <a href="http://example.org/gmark/">http://example.org/gmark/</a></strong></td>
<td>MATCH (x)&lt;-[[:a]-()-[:a]]-&gt;(y),</td>
</tr>
<tr>
<td><strong>SELECT DISTINCT ?x ?z</strong></td>
<td>(y)&lt;-[[:a]-()-[:r]-()-[:e]]-&gt;(z)</td>
</tr>
<tr>
<td>*<em>WHERE { ?x (^:a/:a)</em> ?y \ . \ ?y ((^:a/^:r/:e)</td>
<td>(:^a/^:r/:o)) ?z \ . }**</td>
</tr>
<tr>
<td></td>
<td>UNION \</td>
</tr>
<tr>
<td></td>
<td>MATCH (x)&lt;-[[:a]-()-[:a]]-&gt;(y),</td>
</tr>
<tr>
<td></td>
<td>(y)&lt;-[[:a]-()-[:r]-()-[:o]]-&gt;(z)</td>
</tr>
<tr>
<td></td>
<td>RETURN DISTINCT x, z;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Datalog</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g0(x,y)&lt;- edge(x1,a,x0), edge(x1,a,x2), \</td>
<td>WITH RECURSIVE c0(src, trg) AS (</td>
</tr>
<tr>
<td>\ \ \ \ \ x=x0, y=x2. \  \  \  \  \  \  \</td>
<td>SELECT edge.src, edge.src FROM edge</td>
</tr>
<tr>
<td>$g0(x,y)&lt;- g0(x,z), g0(z,y)$ \ . \ \ \</td>
<td>UNION \</td>
</tr>
<tr>
<td>$g1(x,y)&lt;- edge(x1,a,x0), edge(x2,r,x1), \</td>
<td>SELECT edge.trg, edge.trg FROM edge \</td>
</tr>
<tr>
<td>\ \ \ \ \ edge(x2,e,x3), x=x0, y=x3. \ .</td>
<td>UNION \</td>
</tr>
<tr>
<td>$g1(x,y)&lt;- edge(x1,a,x0), edge(x2,r,x1), \</td>
<td>SELECT s0.src, s0.trg \</td>
</tr>
<tr>
<td>\ \ \ \ \ edge(x2,o,x3), x=x0, y=x3. \ .</td>
<td>FROM (SELECT trg as src, src as trg,</td>
</tr>
<tr>
<td>$query(x,z)&lt;- g0(x,y), g1(y,z)$ \ . \ \ \</td>
<td>........................................</td>
</tr>
</tbody>
</table>
On my laptop, gMark easily generates workloads of one thousand queries for Bib in ~ 0.3s; LSN and SP in ~ 1.5s; and for the richer WD scenario in ~ 10s.

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

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Wrap Up
Recap

Novel contributions of gMark

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Recap

Novel contributions of gMark

- schema-driven graph and query-workload generation, featuring instance-independent selectivity estimation;

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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!

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
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

George Fletcher, TU Eindhoven
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

Come see us at our VLDB 2016 demo!

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Looking ahead to gMark v2.0

To-do/wishlist.

- richer queries
  - support of constants in queries
  - additional query shapes
  - aggregation for BI workloads
  - extensions of selectivity estimation to higher arity queries,
Looking ahead to gMark v2.0

To-do/wishlist.

- richer queries
  - support of constants in queries
  - additional query shapes
  - aggregation for BI workloads
  - extensions of selectivity estimation to higher arity queries,
- richer schemas
  - configuration parameter completion,
  - schema constructs for correlated structure,
  - class hierarchies

https://github.com/graphMark/gmark

George Fletcher, TU Eindhoven
Looking ahead to gMark v2.0

To-do/wishlist.

- richer queries
  - support of constants in queries
  - additional query shapes
  - aggregation for BI workloads
  - extensions of selectivity estimation to higher arity queries,
- richer schemas
  - configuration parameter completion,
  - schema constructs for correlated structure,
  - class hierarchies
- align our work with LDBC activities?

https://github.com/graphMark/gmark
George Fletcher, TU Eindhoven
gMark: Schema-driven data and workload generation for graph databases

George Fletcher
TU Eindhoven, Netherlands

Joint work with
G. Bagan (Lyon), A. Bonifati (Lyon), R. Ciucanu (Oxford), A. Lemay (Lille), and N. Advokaat (Eindhoven)

https://github.com/graphMark/gmark

Thank you!