# Synthetic Graph Generation from Finely-Tuned Temporal Constraints

Karim ALAMI
University of Bordeaux

joint work with

Radu CIUCANU and Engelbert MEPHU NGUIFO
(University Clermont Auvergne)

TDLSG workshop
Sep 18, 2017

# Table of contents

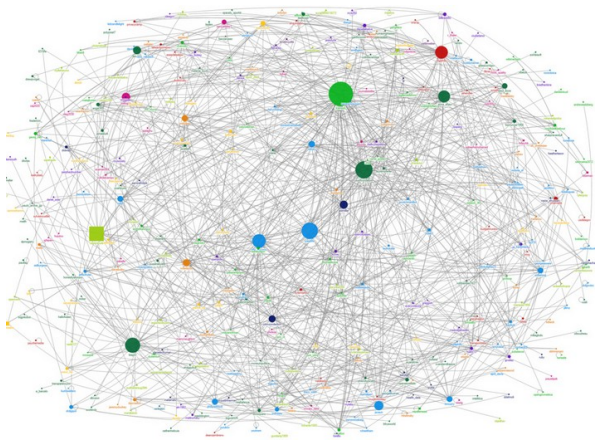
# EGG : Evolving Graph Generator

## Evaluation of EGG

## Conclusion

# Evolving graphs

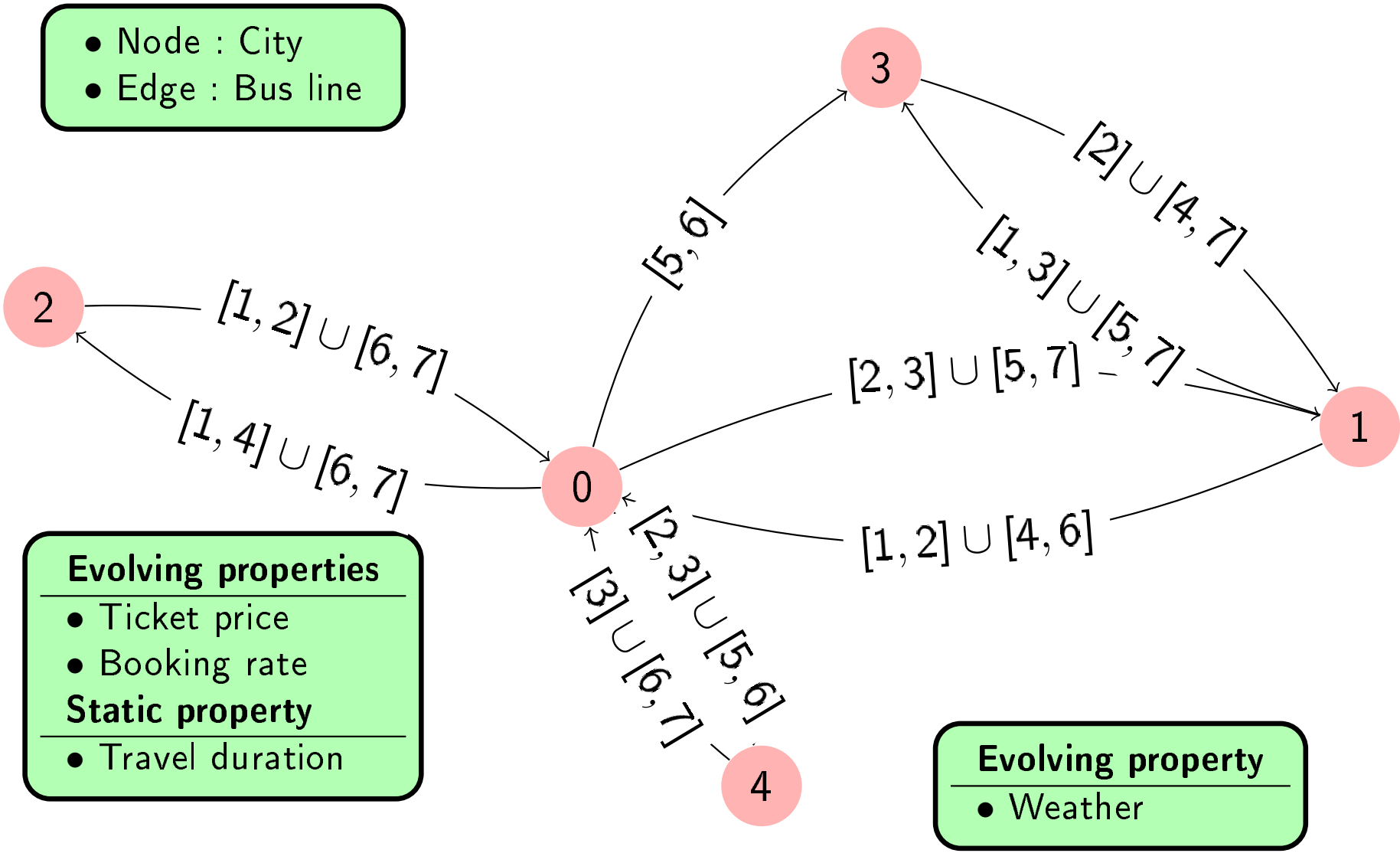
Graphs represent several domains where there is interactions between entities,as social networks, flow networks, biology, geography, and others.



Graphs evolve over time, we can cite two types of evolution :

- Validity of subgraphs during an interval of time.
- Evolving properties of nodes and edges.

# Use case


• Node : City
• Edge : Bus line
3
2
1
0
4
[5, 6]
[2] ∪ [4, 7]
[1, 3] ∪ [5, 7]
[1, 2] ∪ [6, 7]
[1, 4] ∪ [6, 7]
[2, 3] ∪ [5, 7]
[1, 2] ∪ [4, 6]
[2, 3] ∪ [5, 6]
[3] ∪ [6, 7]
Evolving properties
• Ticket price
• Booking rate
Static property
• Travel duration
Evolving property
• Weather

# Processing evolving graphs

New approaches for processing evolving graphs.

For eg. the best path from 0 to 3



- The shortest path : 0 -> 3
- The fastest path : 0 -> 3, 1 hour
- The earliest arrival path : 0 -> 1 -> 3, arrival before 4.

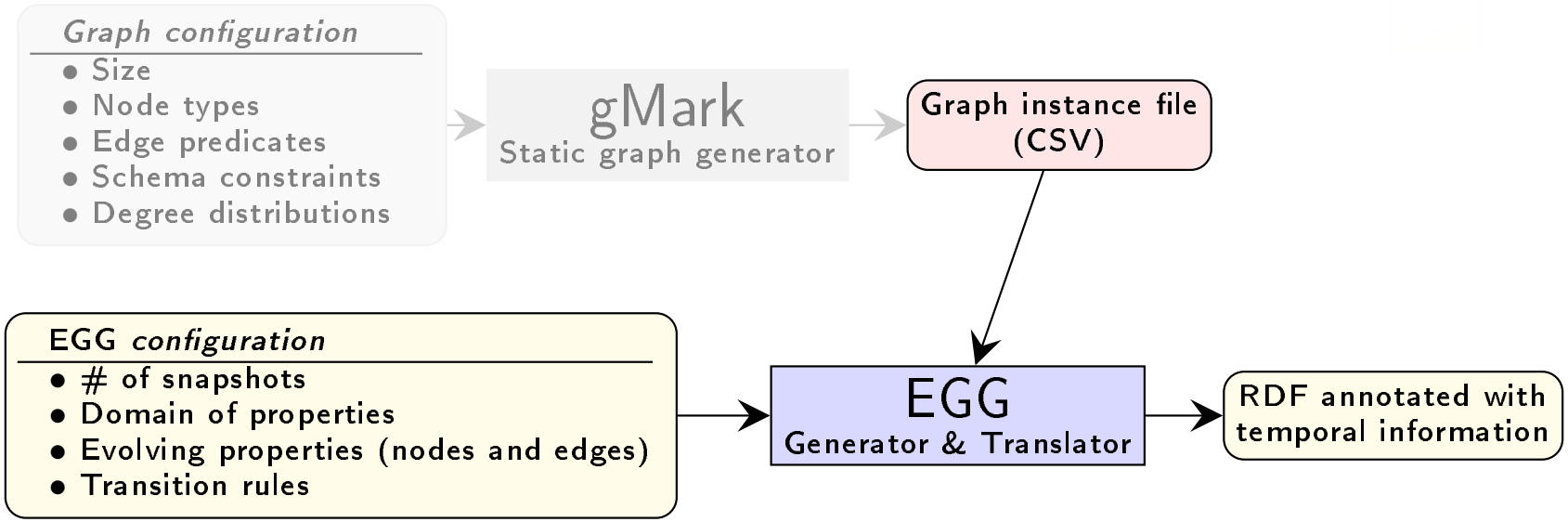
# Graph generation

- gMark [1] : schema-driven static graph generator.
- EvoGen [2] : evolving LUBM [3] generator

1. Bagan, Bonifati, Ciucanu, Fletcher, Lemay, Advokaat. *gMark : Schema-Driven Generation of Graphs and Queries*. TKDE'17.
2. M. Meimaris. Evogen : a generator for synthetic versioned RDF. In EDBT/ICDT, 2016.
3. Y. Guo and et al. LUBM : A benchmark for OWL knowledge base systems. J. Web Sem., 3(2-3) :158-182, 2005.

# Architecture

## EGG : Evolving Graph Generator




we take the following gMark configuration : n=50,
$\Theta = (city, hotel)$, $\Sigma = (train, contains)$,
$T(city) = 0.1$, $T(hotel) = 0.9$,
$\eta(city, train, city) = (n, u), \eta(city, contains, hotel) = (z, u)$.

# EGG configuration

We define $\Delta = (S, I, \sigma, map, \Phi)$ as the evolving graph generator configuration :

- **Interval I** : # of periods we want to generate.
- $\sigma$ : finite set of dynamic properties, is a union of 4 sets of properties type :
  - $\sigma_1$ : set of qualitative properties without order.
    e.g., `weather`
  - $\sigma_2$ : set of qualitative properties with order.
    e.g., `airQuality`, `star`
  - $\sigma_3$ : set of quantitative properties with discrete values.
    e.g., `roomAvailable`
  - $\sigma_4$ : set of quantitative properties with continuous values.
    e.g., `hotelPrice`, `trainPrice`
- **map** : function that maps edge predicates and node types to their properties.

# EGG configuration(suite)

$\Phi$ defines for each property, its domain, its evolution parameters, and its influence on other properties

- $\Gamma$ : frequency of value change
- **value mapping** $v_i : \sigma_i \rightarrow (interval|set\ of\ values)$

  e.g., `weather->"sunny","cloudy","rainy",...`

  e.g., `trainPrice->[20,100]`
- **distribution of values** $\lambda : (\Sigma \cup \Theta, \sigma) \rightarrow distributionlaw$

  e.g., `train, trainPrice->gaussian`
- **dynamicity** : probability that a value change from a valid period to another.
- **succession** :

| | function | values |
|---|---|---|
| $\sigma_1$ | succ | $P(v_1(\sigma_1))$ |
| $\sigma_2 \cup \sigma_3$ | ioffset | $[x, y] \in \mathbb{Z}$ |
| $\sigma_4$ | roffset | $[x, y] \in \mathbb{R}$ |

# EGG configuration(suite)

Correlation rules : there is a need to define two types of rules

- rules that define the property domain

  e.g., `the star of a hotel define its price range`

- rules that define the property evolution

  e.g., `if availableRoom goes down, then the price will grow up`

# Implementation challenges

We had some implementations challenges regarding the processing and memory usage.

- To make the system finish in reasonable time.
- And to consume less memory.

Some steps

- Check that dependence graph of the properties is acyclic
- Lighten some constraints
- Minimize the storage redundancy

# Approach

1. Generate a static graph SG through gMark.
2. Sort topologically properties dependencies.
3. Generate the first valid graph $G_0$.(Subgraph of SG)
4. Generate evolving properties for $G_0$.
5. For each snapshot in (1,n)
   - 5.1 Generate the valid graph $G_i$.
   - 5.2 Generate the evolving properties for $G_i$.

# Serialization

A storage that minimizes redundancy of data.

- RDF with temporal predicates [4] : we store the static information once in a named graph and the evolving data in named graphs that have temporal information.

  e.g., `ns1:G31 {<hotel:27> ns2:hasProperty <Property:availableRooms>}`

- Version graph [5] : we store only an edge with its valid interval.

  e.g., `657 567 2,8 10,20`

4. J. Tappolet and et al. Applied temporal RDF : efficient temporal querying of RDF data with SPARQL. In ESWC, pages 308-322, 2009.

5. K. Semertzidis and et al. TimeReach : Historical reachability queries on evolving graphs. In EDBT, pages 121-132, 2015.

# Expressiveness

With the finely-tuned constraints of EGG configuration, we express several real world use cases

dblp : stores nodes of type author and edges of type co-author.

social : stores persons as nodes, and friendship relation as edges.

univ : LUBM [6] like use case.

shop : WatDiv [7] like use case.

---

6. Y. Guo and et al. LUBM : A benchmark for OWL knowledge base systems. J. Web Sem., 3(2-3) :158-182, 2005.

7. G. Aluc and et al. Diversified stress testing of RDF data management systems. In ISWC, pages 197-212, 2014. Data Management Systems

EGG : Evolving Graph Generator

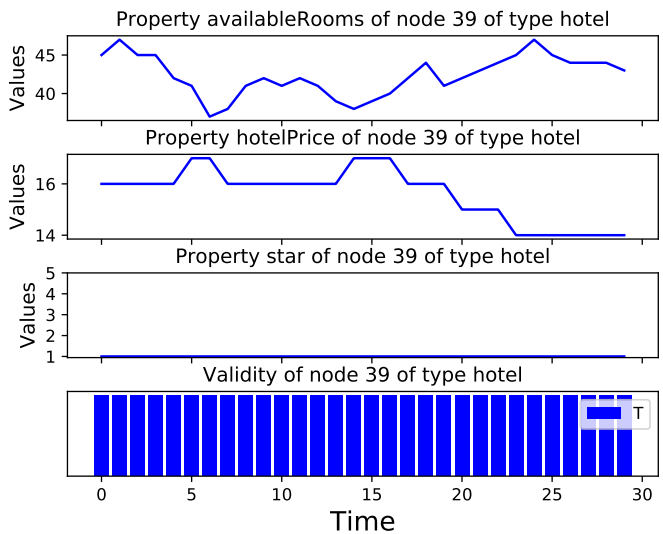
Evaluation of EGG

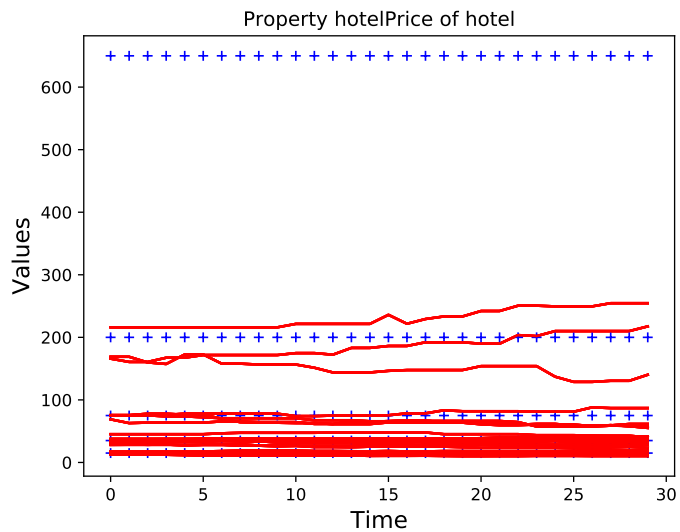
Conclusion

# Accuracy

We developed a visualization module to illustrate the accuracy of EGG generation with respect to the configuration.

- by visualizing the evolution of properties of a node or edge.
- by visualizing the evolution of a property regarding all the nodes or edges that have this property



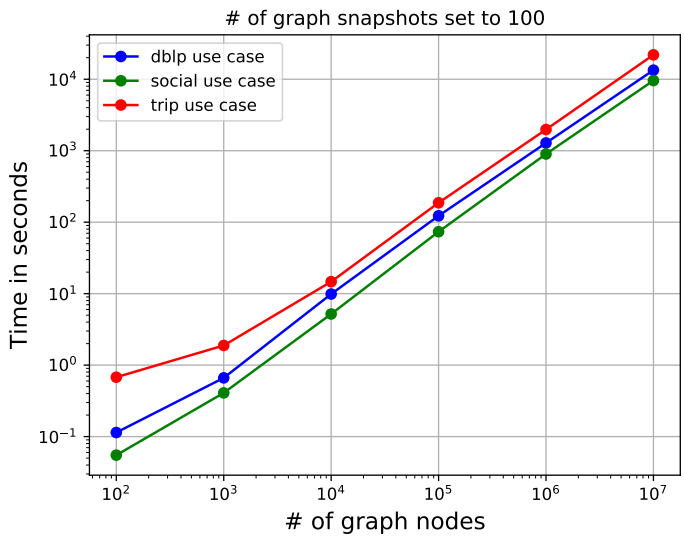



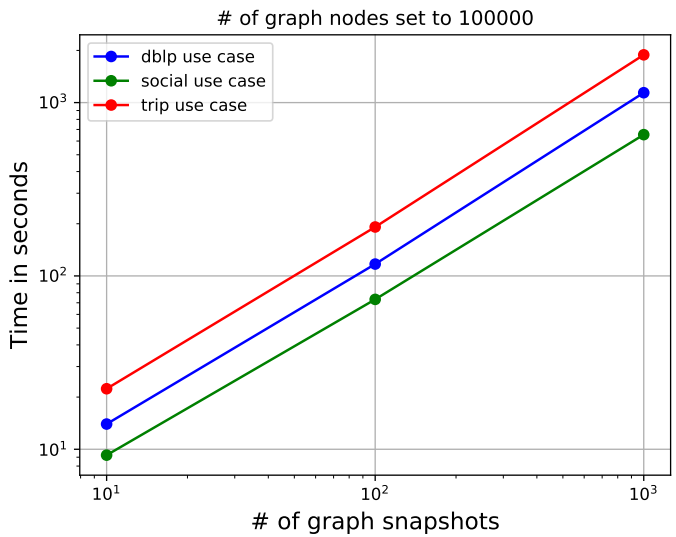
# Scalability

We evaluate the scalability with respect to two parameters :

- size of the static graph in the gMark configuration.
- number of graph snapshots in the EGG configuration.

Machine configuration :16 x 2.4 GHz CPU and 64 GB RAM, reporting averages over 5 runs



# Problem

To illustrate the ease of using EGG in empirical evaluations.

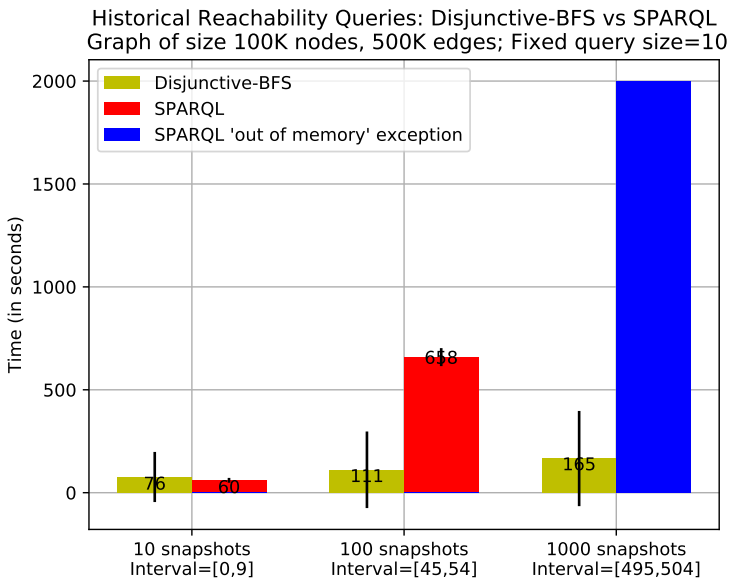
Historical Reachability problem : asks whether there exists a path between two nodes in a specified interval of time.

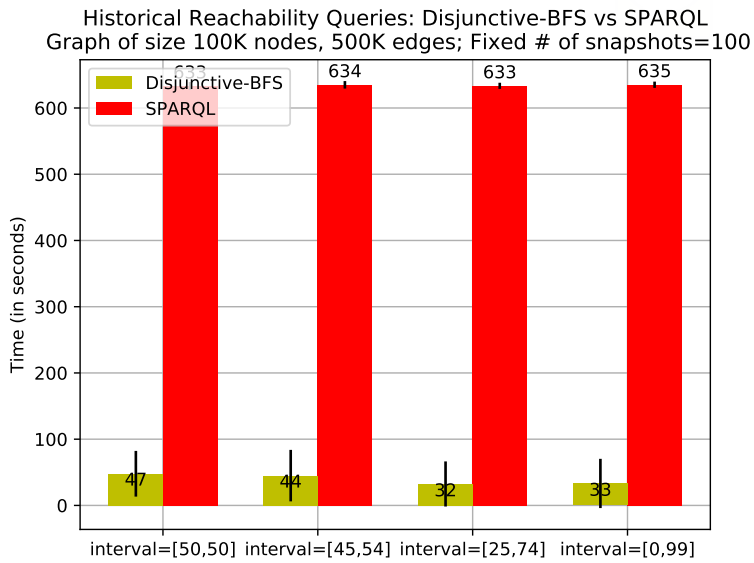
- Disjunctive-BFS [8] presented in the paper, based on dynamic programming takes as input the evolving graph in version graph format.
- SPARQL implementation on top of Apache JENA [9] with evolving graph in RDF format.

8. K. Semertzidis and et al. TimeReach : Historical reachability queries on evolving graphs. In EDBT, pages 121-132, 2015.
9. https ://jena.apache.org/

# Experiments

We have run two experiments that we have found relevant as they express difference between the two approaches :



EGG : Evolving Graph Generator

Evaluation of EGG

Conclusion

# Synthesis

Contribution

- Generation of evolving graphs driven by finely-tuned constraints defined by the user.
- Serialization of the evolving graph in RDF.
- Use in empirical evaluations of evolving graph processing systems.

Future work

- Computational complexity of the generation process.

# Valorization

## Open source

Available on Github repository at
`https://github.com/karimalami7/EGG`

## Publications

Alami, Ciucanu, Mephu Nguifo
EGG : A Framework for Generating Evolving RDF Graphs
ISWC 2017 demo

## Acknowledgement

LEG Project : Large Evolving Graphs
http ://leg.isima.fr/

# Snippets of EGG output

Version graph of social use case (generation of 10 snapshots)
30 0 0,9
40 0 3,9
16 0 0,9
1 1 0,9
16 1 0,9
33 2 1,9
12 2 1,9
10 2 1,9
33 2 0,9
9 2 2,9
32 3 1,9
10 3 0,9
23 4 1,9
30 4 0,9

# Snippets of EGG output

RDF graph of social use case (generation of 10 snapshots)
ns1 :snapshot5 {
<person :1> ns2 :friendOf <person :31> .
<person :11> ns2 :friendOf <person :12> .
<person :20> ns2 :friendOf <person :18>, <person :37> .
<person :27> ns2 :friendOf <person :14>, <person :42>,
<person :49> .
<person :34> ns2 :friendOf <person :0>, <person :16>,
<person :17>, <person :32> .
<person :38> ns2 :friendOf <person :13>, <person :43>,
<person :6> .
...}