Finding paths in large data graphs

Bruno Guillon Mikaela Keller Charles Paperman

UCA

LIMOS – UCA, Clermont-Ferrand

June 4, 2020

2012-2016 PhD in Paris (with C.Choffrut) and Milan (G.Pighizzini):

- nondeterministic two-way transducers and weighted automata
- rational transductions and beyond
- unary transducers
- descriptional complexity of automata

2016-2017 Postdoc in Warsaw (with M.Bojańczyk):

- extension of Monadic Second Order Logic on words
- regular transducers with origin semantics

2017-2018 Postdoc in Milan (with G.Pighizzini):

- reversibility in automata and transducers
- descriptional complexity of automata

2012-2016 PhD in Paris (with C.Choffrut) and Milan (G.Pighizzini):

- nondeterministic two-way transducers and weighted automata
- rational transductions and beyond
- unary transducers
- descriptional complexity of automata

2016-2017 Postdoc in Warsaw (with M.Bojańczyk):

- extension of Monadic Second Order Logic on words
- regular transducers with origin semantics

2017-2018 Postdoc in Milan (with G.Pighizzini):

- reversibility in automata and transducers
- descriptional complexity of automata

2012-2016 PhD in Paris (with C.Choffrut) and Milan (G.Pighizzini):

- nondeterministic two-way transducers and weighted automata
- rational transductions and beyond
- unary transducers
- descriptional complexity of automata

2016-2017 Postdoc in Warsaw (with M.Bojańczyk):

- extension of Monadic Second Order Logic on words
- regular transducers with origin semantics

2017-2018 Postdoc in Milan (with G.Pighizzini):

- reversibility in automata and transducers
- descriptional complexity of automata

2012-2016 PhD in Paris (with C.Choffrut) and Milan (G.Pighizzini):

- nondeterministic two-way transducers and weighted automata
- rational transductions and beyond
- unary transducers
- descriptional complexity of automata

2016-2017 Postdoc in Warsaw (with M.Bojańczyk):

- extension of Monadic Second Order Logic on words
- regular transducers with origin semantics

2017-2018 Postdoc in Milan (with G.Pighizzini):

- reversibility in automata and transducers
- descriptional complexity of automata

2012-2016 PhD in Paris (with C.Choffrut) and Milan (G.Pighizzini):

- nondeterministic two-way transducers and weighted automata
- rational transductions and beyond
- unary transducers
- descriptional complexity of automata

2016-2017 Postdoc in Warsaw (with M.Bojańczyk):

- extension of Monadic Second Order Logic on words
- regular transducers with origin semantics

2017-2018 Postdoc in Milan (with G.Pighizzini):

- reversibility in automata and transducers
- descriptional complexity of automata

November 2018-* Postdoc at INRIA Lille (with C.Paperman):

- algorithms for Regular Path Queries on large graphs

– work in progress

Graph Databases

- Graph databases
- Renewal with the noSQL trend, *e.g.*, neo4j (SPARQL),
  back-end for the rdf and semantic web
  - more flexible than relational databases
  - easier to distribute than relational databases
    – notion of data locality, *e.g.*, TitanDB, JanusGraph (gremlin)

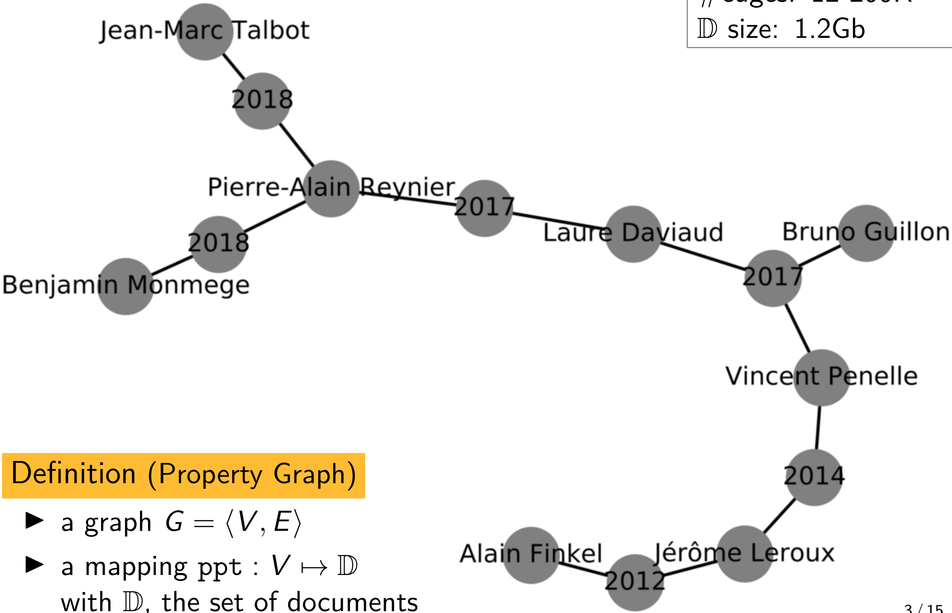
Graph Databases

- Graph databases
- Renewal with the noSQL trend, *e.g.*, neo4j (SPARQL),
  back-end for the rdf and semantic web
  - more flexible than relational databases
  - easier to distribute than relational databases
    – notion of data locality, *e.g.*, TitanDB, JanusGraph (gremlin)

- [Sun *et al.*'15 – SQLGraph: an efficient relational-based property graph]

Running example: DBLP graph

bipartite, undirected
#nodes: 5 750K
#edges: 12 200K
$\mathbb{D}$ size: 1.2Gb



Definition (Property Graph)

- a graph $G = \langle V, E \rangle$
- a mapping ppt : $V \mapsto \mathbb{D}$
  with $\mathbb{D}$, the set of documents

Paths

Definition (Regular Path Queries (RPQ))

Given

- a property graph $G = \langle V, E \rangle$ with $\mathtt{ppt} : V \mapsto \mathbb{D}$;
- $s, t \in V$;
- a regular expression e on $\mathbb{D}$:

find a path $\pi = s \cdot v_1 \cdots v_\ell \cdot t$ such that $\pi \in [|e|]$.

Paths

Definition (Regular Path Queries (RPQ))

Given

- a property graph $G = \langle V, E \rangle$ with $\mathtt{ppt} : V \mapsto \mathbb{D}$;
- $s, t \in V$;
- a regular expression e on $\mathbb{D}$:

find a path $\pi = s \cdot v_1 \cdots v_\ell \cdot t$ such that $\pi \in [|e|]$.

Examples (on DBLP property graph)

find path from given source to given target visiting

- only papers published after 2016
- only journal papers/not arxiv papers
- only French authors
- only paper whose abstract does not contain the word "experimental"

Paths

Definition (Regular Path Queries (RPQ))

Given

- a property graph $G = \langle V, E \rangle$ with $\mathtt{ppt} : V \mapsto \mathbb{D}$;
- $s, t \in V$;
- a regular expression e on $\mathbb{D}$:

find a path $\pi = s \cdot v_1 \cdots v_\ell \cdot t$ such that $\pi \in [|e|]$.

Examples (on DBLP property graph)

find path from given source to given target visiting

- only papers published after 2016
- only journal papers/not arxiv papers
- only French authors
- only paper whose abstract does not contain the word “experimental”
- papers in chronological order along the path

Problem

from given source
to given target

How to find regular paths in large property graphs efficiently?

$G = \langle V, E \rangle$
$\mathrm{ppt} : V \mapsto \mathbb{D}$

Problem

from given source
to given target

How to find regular paths in
large property graphs efficiently?

$G = \langle V, E \rangle$
$\mathrm{ppt} : V \mapsto \mathbb{D}$

stored on disk

Problem

from given source
to given target

How to find regular paths in large property graphs efficiently?

$G = \langle V, E \rangle$
$\mathtt{ppt} : V \mapsto \mathbb{D}$

stored on disk

Problem

from given source
to given target

How to find regular paths in large property graphs efficiently?

$G = \langle V, E \rangle$
$\mathrm{ppt} : V \mapsto \mathbb{D}$

stored on disk

goal: find path from given source to given target by visiting few nodes

Problem

from given source
to given target

How to find regular paths in large property graphs efficiently?

$G = \langle V, E \rangle$
$\text{ppt} : V \mapsto \mathbb{D}$

stored on disk

goal: find path from given source to given target by visiting few nodes

allowed:

- computation time when operating in RAM is "free"

Problem

from given source
to given target

How to find regular paths in large property graphs efficiently?

$G = \langle V, E\rangle$
$\mathrm{ppt} : V \mapsto \mathbb{D}$

stored on disk

goal: find path from given source to given target by visiting few nodes

allowed:

- computation time when operating in RAM is "free"
- precomputed information (on disk): $\mathcal{O}(n \cdot \mathrm{polylog}(n))$ space

Outline

a property graph $G, \mathtt{ppt}$

$\mathcal{O}(n \cdot \mathrm{polylog}(n))$

$\langle G, \mathtt{ppt} \rangle$ + additional information

$\langle G, \mathtt{ppt} \rangle$ + additional information

source $\mathtt{s}$ and target $\mathtt{t}$

regular expression e

fast, using the additional information

a path from $\mathtt{s}$ to $\mathtt{t}$ in G, matching e

Outline

1. Search paths
   - Bilateral Best First Search
   - Tradeoff between short path and efficiency
2. Learning graph embeddings
3. Experimental work
   - What to measure
   - Results
4. Future work

From Regular Path Queries to classical path search

Given an graph exploration algorithm

$$G$$

From Regular Path Queries to classical path search

Given an graph exploration algorithm

we extend it to a graph regular exploration algorithm

G

From Regular Path Queries to classical path search

Given an graph exploration algorithm

we extend it to a graph regular exploration algorithm,

by exploring the direct product of the graph and an automaton

$$G \times A$$

From Regular Path Queries to classical path search

Given an graph exploration algorithm

we extend it to a graph regular exploration algorithm,

by exploring the direct product of the graph and an automaton

$$\underbrace{G \times A}_{\substack{\downarrow \\ \text{not materialized}}}$$

Classical searches

- Breadth-First-Search: very bad (social networks have small diameter)

Classical searches

- Breadth-First-Search: very bad (social networks have small diameter)
- Bilateral BFS: far better

Classical searches

- Breadth-First-Search: very bad (social networks have small diameter)
- Bilateral BFS: far better

- Best First Search algorithms: use heuristic information to guide the search

Classical searches

- Breadth-First-Search: very bad (social networks have small diameter)
- Bilateral BFS: far better

- Best First Search algorithms: use heuristic information to guide the search
  - A^*: if admissible heuristics (shortest path found)

Classical searches

- Breadth-First-Search: very bad (social networks have small diameter)
- Bilateral BFS: far better

- Best First Search algorithms: use heuristic information to guide the search
  - A^*: if admissible heuristics (shortest path found)
  - otherwise, a path may be found, not necessarily a shortest one

Embeddings

Definition:

- $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$
- distance: $||\mu(v) - \mu(u)||$

Embeddings

Definition:

- $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$
- distance: $||\mu(v) - \mu(u)||$

What is a good embedding?

Embeddings

Definition:

- $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$
- distance: $||\mu(v) - \mu(u)||$

What is a good embedding? not clear.

Embeddings

Definition:

- μ : $\{$nodes$\} \mapsto \mathbb{R}^d$
- distance: $||\mu(v) - \mu(u)||$

What is a good embedding? not clear.

Intuitively, we want that being close in the embedding
means being close in the graph topology

Path searching

We assume an embedding $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$.

Path searching

We assume an embedding $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$.

Algorithme : Best-First-Search

input : graph `G`, embedding μ, source `s`, target `t`
output : a path from s to t in G

`Visited` $\leftarrow$ {`s`};
while `t` *not in* `Visited` **do**
  `v` $\leftarrow$ **`select`** *best node* in adherence of `Visited` ;
  add `v` to `Visited` ;
return *a path from* `s` *to* `t` *in the subgraph induced by* `Visited` *;*

Path searching

We assume an embedding $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$.

Algorithme : Best-First-Search

input : graph `G`, embedding μ, source `s`, target `t`
output : a path from s to t in G

`Visited` $\leftarrow \{\mathtt{s}\}$;
while `t` *not in* `Visited` **do**
  `v` $\leftarrow$ **`select`** *best node* in adherence of `Visited` ;
  minimizing $||\mu(\mathtt{t}) - \mu(\mathtt{v})||$
  add `v` to `Visited` ;
return *a path from* `s` *to* `t` *in the subgraph induced by* `Visited` *;*

Path searching

We assume an embedding $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$.

Algorithme : Best-First-Search

input : graph G, embedding μ, source s, target t, parameter α
output : a path from s to t in G

Visited $\leftarrow \{s\}$;
while t *not in* Visited **do**
  v $\leftarrow$ **select** *best node* in adherence of Visited ;
    minimizing $||\mu(t) - \mu(v)|| \cdot \mathrm{depth}(v)^{\alpha}$
  add v to Visited ;
return *a path from* s *to* t *in the subgraph induced by* Visited ;

- $\mathrm{depth}(v)$: the shortest path length from s to v in the explored subgraph

Path searching

We assume an embedding $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$.

Algorithme : Best-First-Search

input : graph $\mathtt{G}$, embedding μ, source $\mathtt{s}$, target $\mathtt{t}$, parameter α
output : a path from s to t in G

$\mathtt{Visited} \leftarrow \{\mathtt{s}\}$;
while $\mathtt{t}$ *not in* $\mathtt{Visited}$ **do**
  $\mathtt{v} \leftarrow$ **select** *best node* in adherence of $\mathtt{Visited}$;
  minimizing $||\mu(\mathtt{t}) - \mu(\mathtt{v})|| \cdot \mathtt{depth}(v)^{\alpha}$
  add $\mathtt{v}$ to $\mathtt{Visited}$;
return *a path from* $\mathtt{s}$ *to* $\mathtt{t}$ *in the subgraph induced by* $\mathtt{Visited}$;

- $\mathtt{depth}(\mathtt{v})$: the shortest path length from $\mathtt{s}$ to $\mathtt{v}$ in the explored subgraph
- when $\alpha = 0$: deep search – when $\alpha \to \infty$: breadth first search

Path searching

We assume an embedding $\mu : \{\text{nodes}\} \mapsto \mathbb{R}^d$.

Algorithme : Best-First-Search

input : graph $\mathtt{G}$, embedding μ, source $\mathtt{s}$, target $\mathtt{t}$, parameter α

output : a path from s to t in G

$\mathtt{Visited} \leftarrow \{\mathtt{s}\}$;

while $\mathtt{t}$ *not in* $\mathtt{Visited}$ **do**

  $\mathtt{v} \leftarrow$ **select** *best node* in adherence of $\mathtt{Visited}$;

  minimizing $||\mu(\mathtt{t}) - \mu(\mathtt{v})|| \cdot \mathtt{depth}(v)^{\alpha}$

  maintain a layered DAG structure ($\mathtt{depth}$);

  add $\mathtt{v}$ to $\mathtt{Visited}$;

return *a path from* $\mathtt{s}$ *to* $\mathtt{t}$ *extracted from the layered DAG;*

- $\mathtt{depth}(\mathtt{v})$: the shortest path length from $\mathtt{s}$ to $\mathtt{v}$ in the explored subgraph
- when $\alpha = 0$: deep search – when $\alpha \to \infty$: breadth first search

Outline

1. Search paths
   - Bilateral Best First Search
   - Tradeoff between short path and efficiency
2. Learning graph embeddings
3. Experimental work
   - What to measure
   - Results
4. Future work

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks: use learning techniques of natural language processing

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks:
  use learning techniques of natural language processing
  - Uniform random walks: deepWalk (2011)

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks: use learning techniques of natural language processing
  - Uniform random walks: deepWalk (2011)
  - Biased random walks: node2vec (2017)

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks: use learning techniques of natural language processing
  - Uniform random walks: deepWalk (2011)
  - Biased random walks: node2vec (2017)
- Landmarks select a small set of nodes, called *landmarks* compute distances from any node to any landmarks

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks: use learning techniques of natural language processing
  - Uniform random walks: deepWalk (2011)
  - Biased random walks: node2vec (2017)
- Landmarks select a small set of nodes, called *landmarks* compute distances from any node to any landmarks
  - already used to guide path searching on road networks (2007)

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks: use learning techniques of natural language processing
  - Uniform random walks: deepWalk (2011)
  - Biased random walks: node2vec (2017)
- Landmarks select a small set of nodes, called *landmarks* compute distances from any node to any landmarks
  - already used to guide path searching on road networks (2007)
  - and to estimate node distance in social networks (2009)

Embeddings

- SVD: find eigenvectors of maximum eigenvalues of the inverse Laplacian matrix
- Random walks + neural networks: use learning techniques of natural language processing
  - Uniform random walks: deepWalk (2011)
  - Biased random walks: node2vec (2017)
- Landmarks select a small set of nodes, called *landmarks* compute distances from any node to any landmarks
  - already used to guide path searching on road networks (2007)
  - and to estimate node distance in social networks (2009)
- many other embeddings to test...

What to measure?

efficiency: how many nodes have been visited with respect to the standard bilateral BFS?

quality: are the found paths much longer than the shortest paths?

Experimental results

method	SVD	node2vec	deepWalk	BFS

Powerlaw cluster graph (#nodes: 2000, #edges: 7979)

mean visited	65.06	76.36	64.13	86.76
score	74.99%	88.01%	73.92%	
mean error	0.03%	0.05%	0.01%	

Regular graph (#nodes: 2000, #edges: 7000)

mean visited	53.55	53.02	56.16	98.43
score	54.41%	53.86%	57.05%	
mean error	0.02%	0.03%	0.0%	

DBLP Graph (#nodes: 5 745K, #edges: 12 205K)

mean visited			1652.14	11632.02
score			14.2%	
mean error			0.26%	
median error			0.22%	

Future work

- improve embeddings

Future work

- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks

Future work

- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks
- from filtered paths to data regular paths

Future work

- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks
- from filtered paths to data regular paths
- subgraph extraction using the embedding – convex closure

Future work

- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks
- from filtered paths to data regular paths
- subgraph extraction using the embedding – convex closure
- index for searching nodes close to a point in the topology
  – quadtree, octrees, k-d-tree (good for small dimensions)
  – space-filling curves

Future work

- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks
- from filtered paths to data regular paths
- subgraph extraction using the embedding – convex closure
- index for searching nodes close to a point in the topology
  – quadtree, octrees, *k*-*d*-tree (good for small dimensions)
  – space-filling curves
- graph clustering

Future work

- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks
- from filtered paths to data regular paths
- subgraph extraction using the embedding – convex closure
- index for searching nodes close to a point in the topology
  – quadtree, octrees, *k*-*d*-tree (good for small dimensions)
  – space-filling curves
- graph clustering

Future work

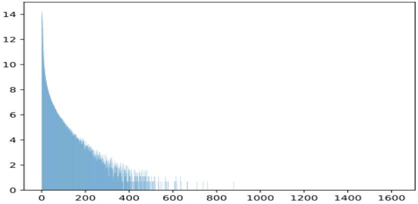
- improve embeddings
- maintain embeddings under updates
  – force algorithms, continuous learning, landmarks
- from filtered paths to data regular paths
- subgraph extraction using the embedding – convex closure
- index for searching nodes close to a point in the topology
  – quadtree, octrees, *k*-*d*-tree (good for small dimensions)
  – space-filling curves
- graph clustering

Thank you for your attention!

More on DBLP graph

- sources: https://dblp.uni-trier.de/, January 16th, 2019 (2.4Gb `xml` file)
- graph: #nodes = **5 746 803** (2 152 092 authors) + `json` for each node, #edges = **12 205 333**, (199Mb for edges (`txt`); 1.2Gb for nodes + `jsons`)
- connected comp.: 57 186 – one large (5 500 304) and others small (< 101)
- degrees: avg=5.24769 . . ., med=3, stdev=10, max=1626,

repartition:
(log scale)



- distances (main cc, evaluated with sample $>$ 110 000 pairs of nodes): avg=10.9, med=11, stdev=2, max=**27**
- alternative encoding "nodes are authors, edges are journals" yields much more edges (19 972 747)