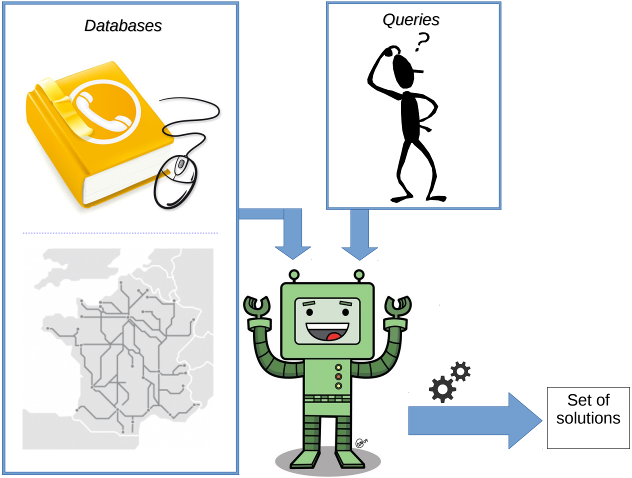
Dynamic Query Evaluation

Alexandre Vigny

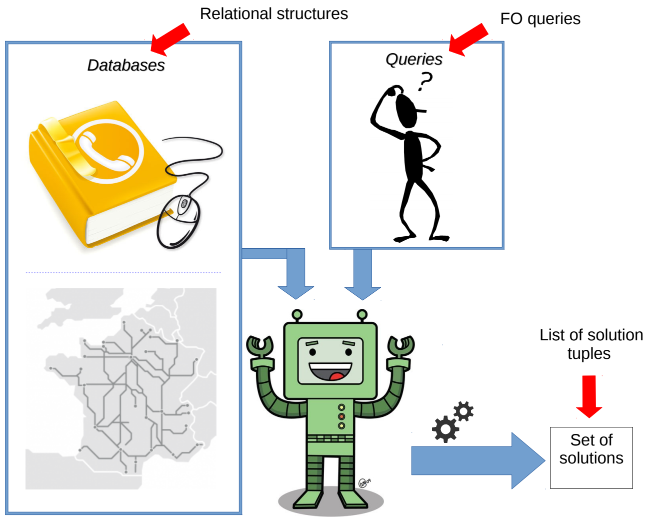
January 18, 2024

Outline

Databases and Queries



Databases and Queries



Formalization: Databases as graphs

P

France
USA
Italy

R

Paris	France
Meudon	France
Houston	USA
Rome	Italy

S

Jack	Houston	Paris

Formalization: Databases as graphs

P

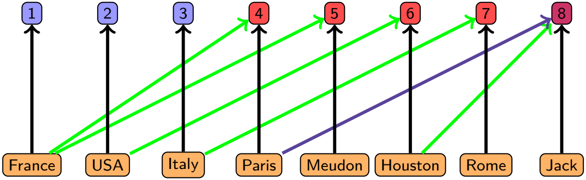
France
USA
Italy

R

Paris	France
Meudon	France
Houston	USA
Rome	Italy

S

Jack	Houston	Paris



Formalization: Databases as graphs

P

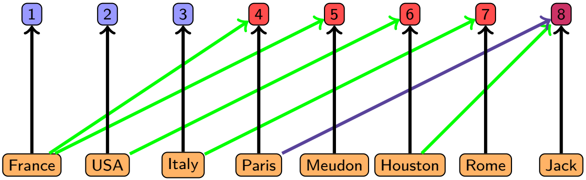
France
USA
Italy

R

Paris	France
Meudon	France
Houston	USA
Rome	Italy

S

Jack	Houston	Paris



$\mathbf{P}(x)$ becomes $\exists w,\ \mathbf{Blue}(w) \wedge \mathbf{B}(x,w)$

$\mathbf{R}(x,y)$ becomes $\exists w,\ \mathbf{Red}(w) \wedge \mathbf{B}(x,w) \wedge \mathbf{G}(y,w)$

$\mathbf{S}(x,y,z)$ becomes $\exists w,\ \mathbf{Purple}(w) \wedge \mathbf{B}(x,w) \wedge \mathbf{G}(y,w) \wedge \mathbf{V}(z,w)$

$\exists x\ \cdots$ becomes $\exists x,\ \mathbf{Orange}(x) \wedge \cdots$

$\forall x\ \cdots$ becomes $\forall x,\ \mathbf{Orange}(x) \Longrightarrow \cdots$

Query evaluation

Input:

→ A database $\mathbf{D}$

→ A query $q(\overline{x})$

Goal:

→ Compute $q(\mathbf{D})$

It is always possible in time $O(|\mathbf{D}|^{|q|})$

Ideally we can do $O(|\mathbf{D}| + |q|) + O(q(\mathbf{D}))$

Computing the whole set of solutions?

In general:

Database: $\|D\|$ the size of the database.

Query: k the arity of the query.

Solutions: Up to $\|D\|^k$ solutions!

Practical problem:

A set of 50^{10} solutions is not easy to store / display!

Theoretical problem:

The time needed to compute the answer does not reflect the hardness of the problem.

Can we do anything else instead?

Inspiration from real world

PhD thesis

Inspiration from real world

PhD thesis

around 200.000 results in 0,5 seconds

> Here is a first solution
> Here is a second one
>
>
>

Next!

Other problems

Model-Checking : Is this true ?

Input:	*Goal:*	*Ideally:*
$\mathbf{D}, q$	Yes or NO? $\mathbf{D} \models q$?	$O(\|\mathbf{D}\|)$

Testing : Is this tuple a solution ?

Counting : How many solutions ?

Enumeration : Enumerate the solutions

Other problems

Model-Checking : Is this true ?

Testing : Is this tuple a solution ?

Input:	*Goal:*	*Ideally:*
$\mathbf{D}, q, \overline{a}$	Test whether $\overline{a} \in q(\mathbf{D})$.	$O(1) \circ O(\|\mathbf{D}\|)$

Counting : How many solutions ?

Enumeration : Enumerate the solutions

Other problems

Model-Checking : Is this true ?

Testing : Is this tuple a solution ?

Counting : How many solutions ?

Input:	*Goal:*	*Ideally:*
$\mathbf{D}, q$	Compute $\lvert q(\mathbf{D}) \rvert$	$O(\lVert \mathbf{D} \rVert)$

Enumeration : Enumerate the solutions

Other problems

Model-Checking : Is this true ?

Testing : Is this tuple a solution ?

Counting : How many solutions ?

Enumeration : Enumerate the solutions

Input:	*Goal :*	*Ideally:*
$\mathbf{D}, q$	Compute 1^{st} sol, 2^{nd} sol, ...	$O(1) \circ O(\lVert \mathbf{D} \rVert)$

Query enumeration

Input : $\|\mathbf{D}\| := n$ & $|q| := k$ (computation with RAM)

Goal : output solutions one by one (no repetition)

Query enumeration

Input : $\|\mathbf{D}\| := n$ & $|q| := k$ (computation with RAM)

Goal : output solutions one by one (no repetition)

STEP 1: Preprocessing

Prepare the enumeration : Database $D \longrightarrow$ Index I

Preprocessing time : $f(k) \cdot n \rightsquigarrow O(n)$

Query enumeration

Input : $\|\mathbf{D}\| := n$ & $|q| := k$ (computation with RAM)

Goal : output solutions one by one (no repetition)

STEP 1: Preprocessing

Prepare the enumeration : Database $D \longrightarrow$ Index I

Preprocessing time : $f(k) \cdot n \rightsquigarrow O(n)$

STEP 2 : Enumeration

The enumerate : Index $I \longrightarrow \overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \cdots$

Delay : $O(f(k)) \rightsquigarrow O(1)$

Constant delay enumeration after linear preprocessing

$O(1) \circ O(n)$

Properties of efficient enumeration algorithms

Mandatory:

→ First solution computed in time $O\big(\|\mathbf{D}\|\big)$.

→ Last solution computed in time $O\Big(\|\mathbf{D}\| + \big|q(\mathbf{D})\big|\Big)$.

→ No repetition!

Optional:

→ Enumeration in lexicographical order.

→ Use a constant amount of memory.

Example 1

→ Database $\mathbf{D} := \langle\{1,\cdots,n\};E\rangle$ $\quad\|\mathbf{D}\| = |E|$

→ Query $q_1(x,y) := E(x,y)$

$$\begin{array}{c} E \\ (1,1) \\ (1,2) \\ (1,6) \\ \vdots \\ (4,5) \\ (4,7) \\ (4,8) \\ \vdots \\ (n,n) \end{array}$$

Example 1

→ Database $\mathbf{D} := \langle\{1, \cdots, n\}; E\rangle$ $\|\mathbf{D}\| = |E|$

→ Query $q_1(x, y) := E(x, y)$

E

(1,1)
(1,2)
(1,6)
⋮
(4,5)
(4,7)
(4,8)
⋮
(n,n)

For the enumeration problem

Preprocessing: nothing

Enumeration: read the list.

For the counting problem

Computation: go through the list

Answering: output the result.

For the testing problem

Harder than it looks!

Dichotomous research? $O(\log(\|\mathbf{D}\|))$.

Example 2

- → Database $D := \langle\{1, \cdots, n\}; E\rangle$ $\|\mathbf{D}\| = |E|$
- → Query $q_2(x, y) := \neg E(x, y)$

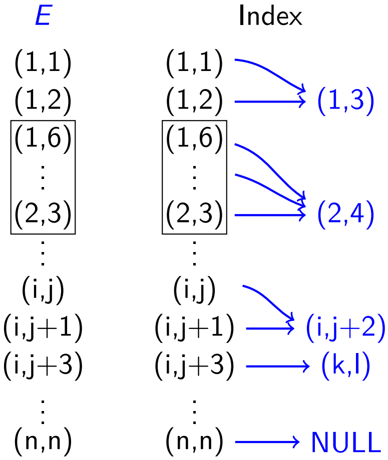
E

(1,1)
(1,2)
$\boxed{\begin{array}{c}(1,6)\\ \vdots \\ (2,3)\end{array}}$
⋮
(i,j)
(i,j+1)
(i,j+3)
⋮
(n,n)

Example 2

→ Database $D := \langle\{1, \cdots, n\}; E\rangle$ $\quad\quad \|\mathbf{D}\| = |E|$

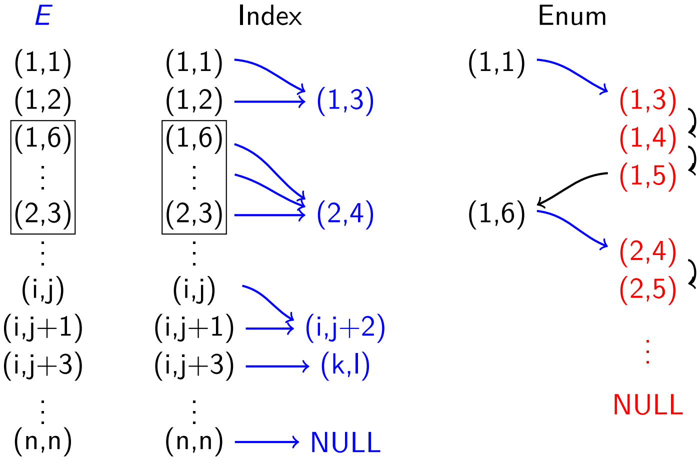
→ Query $q_2(x,y) := \neg E(x,y)$



Example 2

→ Database $D := \langle \{1, \cdots, n\}; E \rangle$ $\|\mathbf{D}\| = |E|$

→ Query $q_2(x, y) := \neg E(x, y)$



Example 3

→ Database $D := \langle \{1, \cdots, n\}; E_1; E_2 \rangle$ $\|D\| = |E_1| + |E_2|$ $(E_i \subseteq D \times D)$

→ Query $q(x,y) := \exists z,\ E_1(x,z) \land E_2(z,y)$

Example 3

→ Database $D := \langle \{1, \cdots, n\}; E_1; E_2 \rangle$ $\quad \|D\| = |E_1| + |E_2| \quad (E_i \subseteq D \times D)$

→ Query $q(x,y) := \exists z,\ E_1(x,z) \wedge E_2(z,y)$

B : Adjacency matrix of E_2

$$
\begin{pmatrix}
E_2(1,1) & \dots & E_2(1,y) & \dots & E_2(1,n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
E_2(z,1) & \dots & E_2(z,y) & \dots & E_2(z,n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
E_2(n,1) & \dots & E_2(n,y) & \dots & E_2(n,n)
\end{pmatrix}
$$

$$
\begin{pmatrix}
E_1(1,1) & \dots & E_1(1,i) & \dots & E_1(1,n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
E_1(x,1) & \dots & E_1(x,z) & \dots & E_1(x,n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
E_1(n,1) & \dots & E_1(n,z) & \dots & E_1(n,n)
\end{pmatrix}
\begin{pmatrix}
q(1,1) & \dots & q(1,y) & \dots & q(1,n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
q(x,1) & \dots & q(x,y) & \dots & q(x,n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
q(n,1) & \dots & q(n,y) & \dots & q(n,n)
\end{pmatrix}
$$

A : Adjacency matrix of E_1

C : Result matrix

Example 3

→ Database $D := \langle\{1,\cdots,n\}; E_1; E_2\rangle$ $\|D\| = |E_1| + |E_2|$ $(E_i \subseteq D \times D)$

→ Query $q(x,y) := \exists z,\ E_1(x,z) \wedge E_2(z,y)$

B : Adjacency matrix of E_2

$$\begin{pmatrix} E_2(1,1) & \dots & E_2(1,y) & \dots & E_2(1,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ E_2(z,1) & \dots & E_2(z,y) & \dots & E_2(z,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ E_2(n,1) & \dots & E_2(n,y) & \dots & E_2(n,n) \end{pmatrix}$$

$$\begin{pmatrix} E_1(1,1) & \dots & E_1(1,i) & \dots & E_1(1,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ E_1(x,1) & \dots & E_1(x,z) & \dots & E_1(x,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ E_1(n,1) & \dots & E_1(n,z) & \dots & E_1(n,n) \end{pmatrix} \begin{pmatrix} q(1,1) & \dots & q(1,y) & \dots & q(1,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q(x,1) & \dots & q(x,y) & \dots & q(x,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q(n,1) & \dots & q(n,y) & \dots & q(n,n) \end{pmatrix}$$

A : Adjacency matrix of E_1 C : Result matrix

Compute the set of solutions

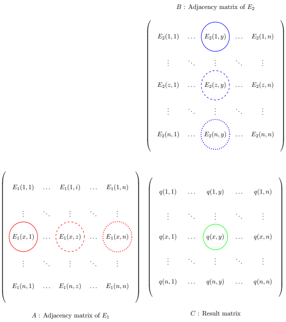
=

Boolean matrix multiplication

Example 3

→ Database $D := \langle \{1, \cdots, n\}; E_1; E_2 \rangle$ $\quad \|D\| = |E_1| + |E_2| \quad (E_i \subseteq D \times D)$

→ Query $q(x,y) := \exists z,\ E_1(x,z) \wedge E_2(z,y)$



→ Linear preprocessing: $O(n^2)$

→ Number of solutions: $O(n^2)$

→ Total time: $O(n^2) + O(1) \times O(n^2)$

→ Boolean matrix multiplication in $O(n^2)$

Conjecture: "There are no algorithm for the boolean matrix multiplication working in time $O(n^2)$."

Example 3

- → Database $D := \langle \{1, \cdots, n\}; E_1; E_2 \rangle$ $\|D\| = |E_1| + |E_2|$ $(E_i \subseteq D \times D)$
- → Query $q(x, y) := \exists z,\ E_1(x, z) \wedge E_2(z, y)$

This query cannot be enumerated with constant delay[1]

We need to put restrictions on queries and/or databases.

[1] Unless there is a breakthrough with the boolean matrix multiplication.

Example 3 bis

- → Database $D := \langle \{1, \cdots, n\}; E_1; E_2 \rangle$ $\|D\| = |E_1| + |E_2|$ $(E_i \subseteq D \times D)$
- → Query $q(x, y) := \exists z,\ E_1(x, z) \wedge E_2(z, y)$

and D is a tree!

Example 3 bis

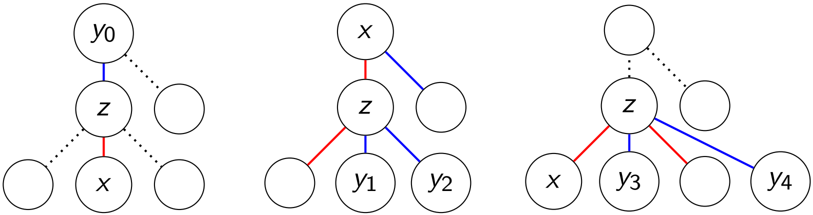
→ Database $D := \langle\{1,\cdots,n\}; E_1; E_2\rangle$ $\quad \|D\| = |E_1| + |E_2| \quad (E_i \subseteq D \times D)$

→ Query $q(x,y) := \exists z,\ E_1(x,z) \wedge E_2(z,y)$

and D is a tree!

Given a node x, every solutions y must be amongst:

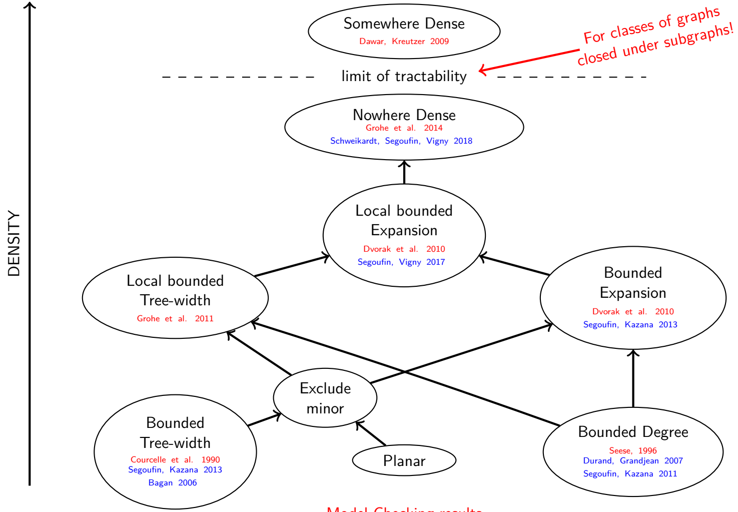
It's "grandparent", "grandchildren", or "siblings"



What kind of restrictions?

No restriction on the database part	Highly expressive queries (MSO queries)	FO queries
$\Downarrow$	$\Downarrow$	$\Downarrow$
Strict subset of ACQ	**Trees** (graphs with bounded tree width)	Nowhere dense graphs
Bagan, Durand, Grandjean	Courcelle, Bagan, Segoufin, Kazana	next slide

Classes of graphs



Model-Checking results
Enumeration results

Motivations

In general the database is not fixed for ever.

We want to add / remove / update information.

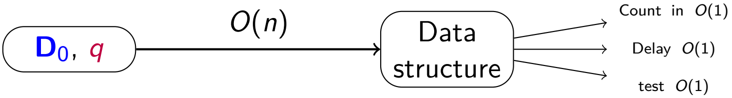
In the graph setting, one is allowed to:

→ create a new vertex / a new edge,

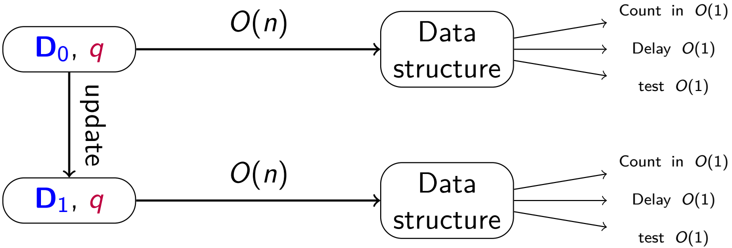
→ delete an edge / an edgeless vertex,

→ change the color of a vertex.

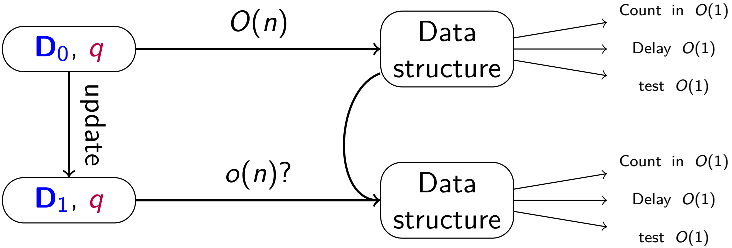
Data structures with updates



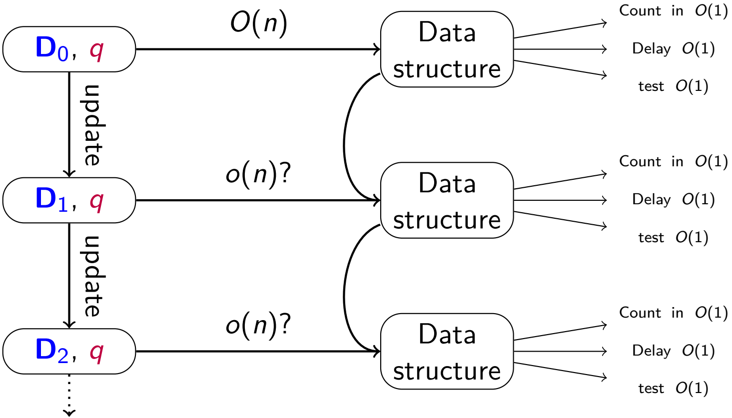
Data structures with updates



Data structures with updates



Data structures with updates



Existing results

No Restriction on databases!

→ Berkholz, Keppeler, Schweikardt PODS '17 & ICDT '18

For MSO queries:

→ Losemann, Martens CSL-LICS '14

→ Niewerth, Segoufin PODS '18

→ Amarilli, Bourhis, Mengel, Niewerth ICDT '18 & PODS '19 & ...

For FO queries:

→ Berkholz, Keppeler, Schweikardt ICDT '17

Recap: → Antoine Amarilli's habilitation. '23

"New" results 1/2

Theorem

Over classes of graphs with *bounded degree*, for every integer ℓ, there is a data structure that:

→ can be **computed** in linear time,

→ can be **updated** in constant time,

→ allows us given any FO query with quantifier-rank + arrity $\leq \ell$:

check whether there is a solution in constant time.
count the number of solution in constant time.
enumerate every solution with constant delay.
test whether a given tuple is a solution in constant time.

What's new is that the structure does not depends on the query!

"New" results 2/2

Theorem

Over classes of graphs with *low degree*, for every FO query and every $\epsilon > 0$, there is a data structure that:

→ can be **computed** in time $O(|G|^{1+\epsilon})$, (*pseudo-linear* time)

→ can be **updated** in time $O(|G|^{\epsilon})$, and (*pseudo-constant* time)

→ allows us to:

check whether there is a solution in constant time.
count the number of solution in constant time.
enumerate every solution with constant delay.
test whether a given tuple is a solution in constant time.

Classes of graphs with low degree

Definition

A class $\mathscr{C}$ has low degree if:

$$\forall \epsilon > 0, \quad deg(G) \in O(|G|^{\epsilon})$$

Examples: bounded degree, degree in $O(\log(n))$ or $O(\log^i(n))$

Counter examples: degree in $O(n^{0,0001})$ or $O(\sqrt{n})$

Not a monotone or hereditary notion

Existing results

Model checking results: Grohe. STACS '01

Test/enumeration/counting results: Durand, Schweikardt, Segoufin PODS '13

Tools

Tool~~s~~

Locality of FO

and that's it!

Tool~~s~~

Locality of FO

and that's it!

The fact that a tuple is a solution only depends on it's neighborhood.

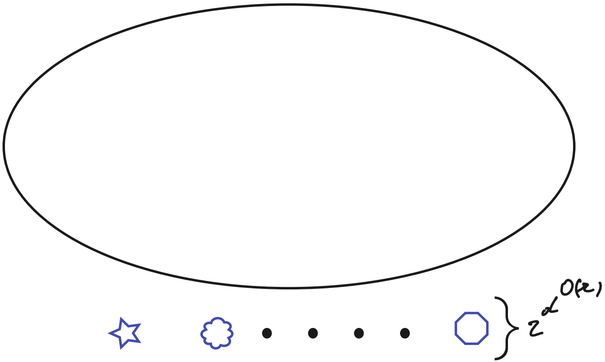
$$G \models q(\bar{a}) \iff N_r^G(\bar{a}) \models q(\bar{a})$$

where $r = O(2^{|q|})$.

Description of the data structure

Given G, d and ℓ: let $r = 2^{\ell}$

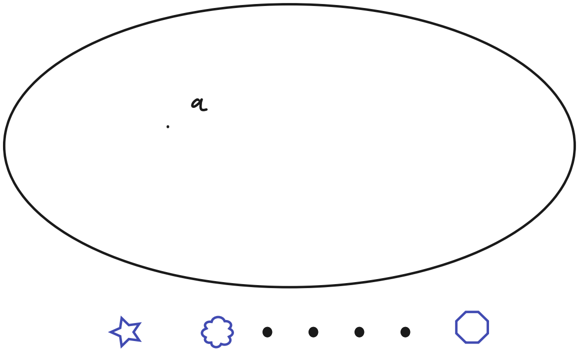
→ The set $\mathscr{H}$ of all graphs H of size d^r and one marked node.
Up to isomorphism. There are only $O\left(2^{(d^r)^2}\right)$ *such graphs.*



Description of the data structure

Given G, d and ℓ: let $r = 2^{\ell}$

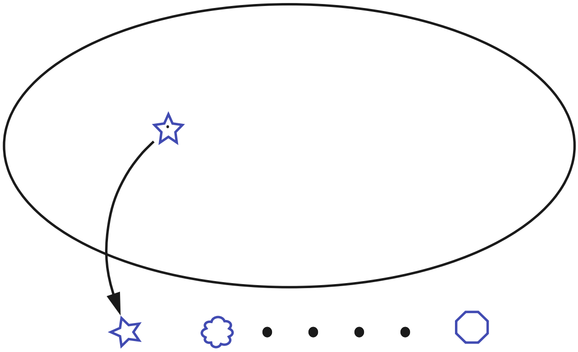
→ A function $\mathscr{X}(\cdot)$ from G to $\mathscr{H}$ such that $N_r^G(a) \simeq \mathscr{X}(a)$.



Description of the data structure

Given G, d and ℓ: let $r = 2^{\ell}$

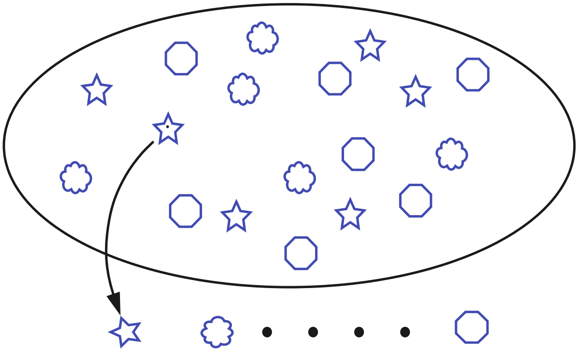
$\rightarrow$ A function $\mathscr{X}(\cdot)$ from G to $\mathscr{H}$ such that $N_r^G(a) \simeq \mathscr{X}(a)$.



Description of the data structure

Given G, d and ℓ: let $r = 2^{\ell}$

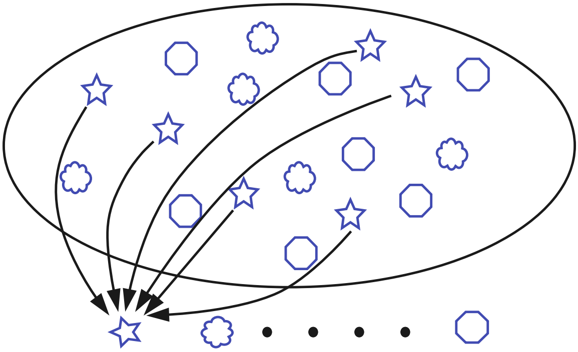
→ A function $\mathscr{X}(\cdot)$ from G to $\mathscr{H}$ such that $N_r^G(a) \simeq \mathscr{X}(a)$.



Description of the data structure

Given G, d and ℓ: let $r = 2^{\ell}$

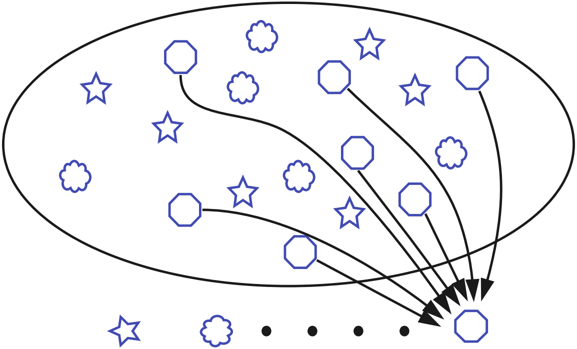
→ A function $\mathscr{X}(\cdot)$ from G to $\mathscr{H}$ such that $N_r^G(a) \simeq \mathscr{X}(a)$.



Description of the data structure

Given G, d and ℓ: let $r = 2^{\ell}$

→ A function $\mathscr{X}(\cdot)$ from G to $\mathscr{H}$ such that $N_r^G(a) \simeq \mathscr{X}(a)$.



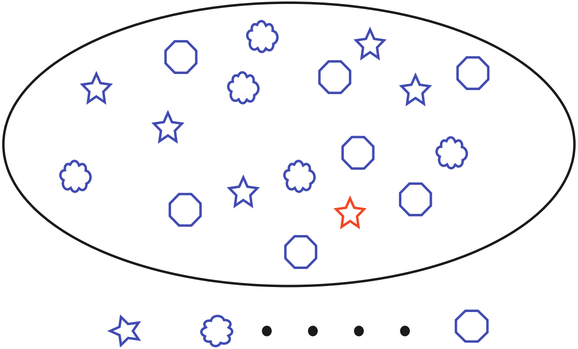
Updates

What happens if there is an update ?

Updates

What happens if there is an update ?

1) If a node is far from the update nothing changes!



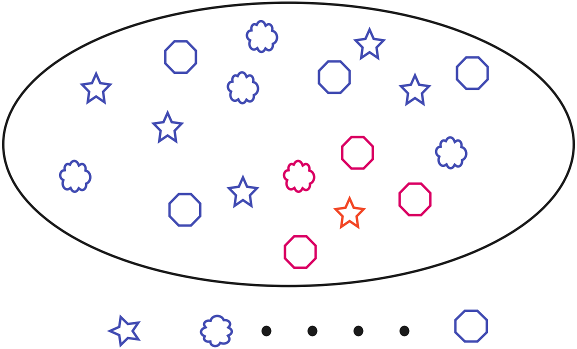
Updates

What happens if there is an update ?

1) If a node is far from the update nothing changes!

2) If a node is close to the update we need to recompute everything

But there are few such nodes!



Updates

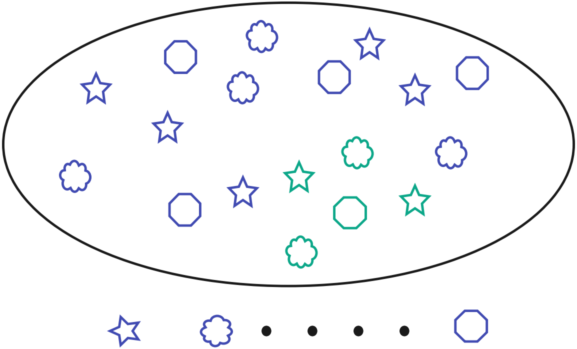
What happens if there is an update ?

1) If a node is far from the update nothing changes!

2) If a node is close to the update we need to recompute everything

But there are few such nodes!

There are only $d(G)^r$ nodes that need something to be done.



The examples queries

→ $q_1() := \exists x, \exists y, \exists z \; E(x,y) \wedge E(y,z) \wedge E(z,x)$

(The triangle query)

→ $q_2(x,y) := \exists z \; E(x,z) \wedge E(z,y)$

(The distance two query)

→ $q_3(x,y) := R(x) \wedge B(y) \wedge \neg q_2(x,y)$

(Red Blue nodes that are far apart)

How to use the structure 1/3

Input: G is now fixed, the data structure is computed:

Goal : test whether there is a triangle.

How to use the structure 1/3

Input: G is now fixed, the data structure is computed:

Goal : test whether there is a triangle.

Solution : Is there an $H \in \mathscr{H}$ such that:

→ There is a triangle in H,

→ $\#L_H > 0$.

Total time: constant time ! (triply exponential in ℓ)

How to use the structure 2/3

Input: G is now fixed, the data structure is computed:

Goal : test/count/enumerate the pairs (a, b) such that $G \models q_2(a, b)$.

$$q_2(x, y) := \exists z \ E(x, z) \wedge E(z, y)$$

How to use the structure 2/3

Input: G is now fixed, the data structure is computed:

Goal : test/count/enumerate the pairs (a, b) such that $G \models q_2(a, b)$.

$$q_2(x, y) := \exists z \; E(x, z) \wedge E(z, y)$$

Solution (count): for all $H \in \mathscr{H}$ of:

→ How many b in H, such that $H \models q_2(a, b)$?
Where a is the marked node.

→ Add $\#L_H \cdot \#q_2(H)$ to the total.

Total time: constant time ! (triply exponential in ℓ)

How to use the structure 3/3

Input: G is now fixed, the data structure is computed:

Goal : test/count/enumerate the pairs (a, b) such that $G \models q_3(a, b)$.

$$q_3(x, y) := R(x) \wedge B(y) \wedge \neg q_2(x, y)$$

How to use the structure 3/3

Input: G is now fixed, the data structure is computed:

Goal : test/count/enumerate the pairs (a, b) such that $G \models q_3(a, b)$.

$$q_3(x, y) := R(x) \wedge B(y) \wedge \neg q_2(x, y)$$

Solution (enumerate):

- → Run through all a such that $G \models \exists y, q_3(a, y)$
  performed by induction
- → Run through all b in B, and don't output those that are close to a.

Constant delay ! (triply exponential in ℓ)

Recap

Theorem:

Over classes of graphs with low or bounded degree, for every FO query, there is a data structure that:

- → can be computed efficiently,
- → can be recomputed efficiently,
- → allows to answer the query in constant time / delay.

Complexity

linear preprocessing: $O\big(f(|q|) \times |G|\big)$
But $f(\cdot)$ is triply exponential, which is optimal.

What remains?

There are still many classes of graphs with statics results but no dynamic ones.

What remains?

There are still many classes of graphs with statics results but no dynamic ones.

What about more powerful updates ?

$$R^G := q(G)$$

New directions

2) Implementable scenarios.

In general the constants factors are a tower of exponentials.

Can they be polynomial? Or even linear?

Existing results for: Document Spanners,[1,2] and for ACQ.[3]

[1] Florenzano, Riveros, Ugarte, Vansummeren, Vrgoc ACM Trans. Database Syst. '20
[2] Amarilli, Bourhis, Mengel, Niewerth ACM Trans. Database Syst. '21
[3] Bagan, Durand, Filiot, Gauwin CSL '10

New directions

2) Implementable scenarios.

In general the constants factors are a tower of exponentials.

Can they be polynomial? Or even linear?

Existing results for: Document Spanners,[1,2] and for ACQ.[3]

Thank you!

[1] Florenzano, Riveros, Ugarte, Vansummeren, Vrgoc ACM Trans. Database Syst. '20
[2] Amarilli, Bourhis, Mengel, Niewerth ACM Trans. Database Syst. '21
[3] Bagan, Durand, Filiot, Gauwin CSL '10