Query enumeration and nowhere dense graphs

Alexandre Vigny

September 27, 2018

Outline

Outline

Databases and Queries

Here is the menu of a creperie:

Ingredients	Plain	Ham	Cheese	Egg	Mushroom	Tomato	⋯
Price (in €)	1	+1.5	+1.5	+1.5	+2	+1	⋯

Query evaluation problem:

Input:

- A database **D** (here the menu)
- A query q

Output:

$q(\mathbf{D})$

Which combinations cost less than 5€?

Plain (it includes butter)
Ham | Chesse | Egg | ⋯
Ham + Cheese | Ham + Egg | Cheese + Egg | Ham + Tomato ⋯
Ham + Cheese + Tomato | Cheese + Egg + Tomato ⋯

Databases and Queries

Here is the menu of a creperie:

Ingredients	Plain	Ham	Cheese	Egg	Mushroom	Tomato	$\cdots$
Price (in €)	1	$+1.5$	$+1.5$	$+1.5$	$+2$	$+1$	$\cdots$

Query evaluation problem:

Input:

- A database $\mathbf{D}$ (here the menu)
- A query q

Output:

$q(\mathbf{D})$

Special case: boolean query i.e. with yes / no answers

Is there a combination with 3 toppings for less than 4€?

NO

Formalization part 1: Databases as relational structures

Schema : $\sigma := \left\{P_{(1)},\ R_{(2)},\ S_{(3)}\right\}$

A *relational structure* $\boldsymbol{D}$:

P

France
USA
Germany
Italy

R

Paris	France
Bourg-la-Reine	France
Houston	USA
Meudon	France
Rome	Italy

S

Alexandre	Bourg-la-Reine	Paris
Sophie	Bourg-la-Reine	Meudon
Tim	Bourg-la-Reine	Bourg-la-Reine
Jack	Houston	Houston
Julie	Paris	Paris

Formalization part 2: Queries written in first-order logic

What are all of the countries?

$q(x) := P(x)$

Is there someone who works and lives in the same city?

$q() := \exists x \exists y \; S(x,y,y)$

What are the pairs of cities that are in the same country?

$q(x,y) := \exists z \; R(x,z) \wedge R(y,z)$

Who are the people who do not work where they live?

$q(x) := \exists y \exists z \; S(x,y,z) \wedge y \neq z$

Which cities satisfy: everybody who lives there works there too?

$q(x) := \forall y \forall z \; S(y,x,z) \implies z = x$

Formalization part 3: Databases as graphs

P

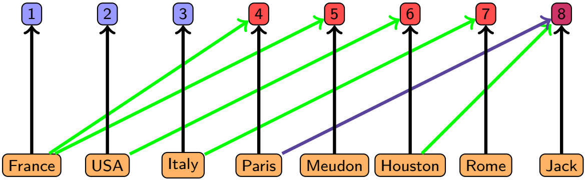
France
USA
Italy

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

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 $\overline{a} \in q(\mathbf{D})$.	$O(1) \circ O(\lVert \mathbf{D} \rVert)$

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 $\left\lvert q(\mathbf{D}) \right\rvert$	$O(\lVert \mathbf{D} \rVert)$

Enumeration : Enumerate the solutions

Other problems

Model-Checking : Is this true ?

Testing : Is this tuple a solution ?

Counting : How many solutions ?

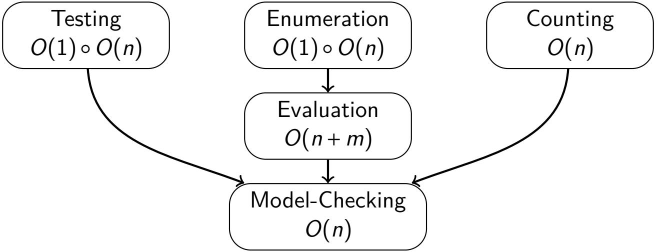
Enumeration : Enumerate the solutions

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

Comparing the problems

For FO queries over a class $\mathscr{C}$ of databases.

			Ideal running time
Model-Checking	:	Is this true ?	$O(n)$
Enumeration	:	Enumerate the solutions	$O(1) \circ O(n)$
Evaluation	:	Compute the entire set	$O(n+m)$
Counting	:	How many solutions ?	$O(n)$
Testing	:	Is this tuple a solution ?	$O(1) \circ O(n)$



Outline

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

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

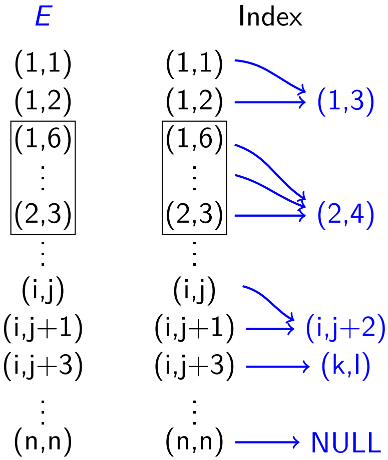
→ Query $q_2(x,y) := \neg E(x,y)$

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

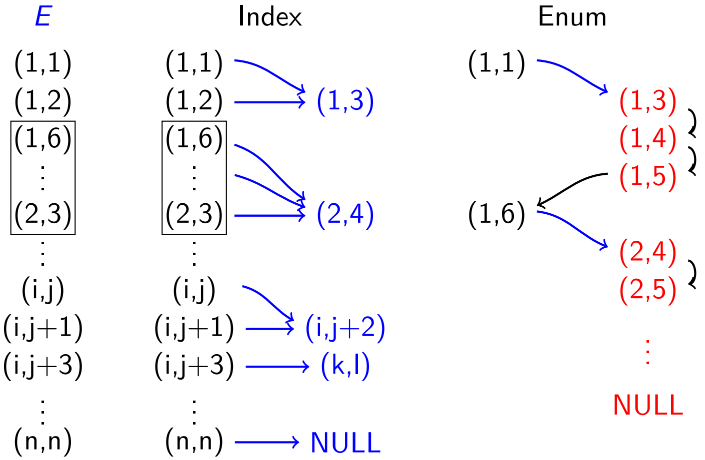
- → Database $D := \langle \{1, \cdots, n\}; E\rangle$ $\qquad \|\mathbf{D}\| = |E|$
- → Query $q_2(x, y) := \neg E(x, y)$



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

Example 3

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

B : Adjacency matrix of E_2

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

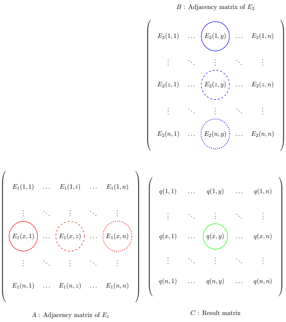
$$\begin{pmatrix} E_1(1,1) & \dots & E_1(1,i) & \dots & E_1(1,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ E_1(x,1) & \dots & E_1(x,z) & \dots & E_1(x,n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ E_1(n,1) & \dots & E_1(n,z) & \dots & E_1(n,n) \end{pmatrix} \begin{pmatrix} q(1,1) & \dots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

A : Adjacency matrix of E_1 $\qquad$ C : Result matrix

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



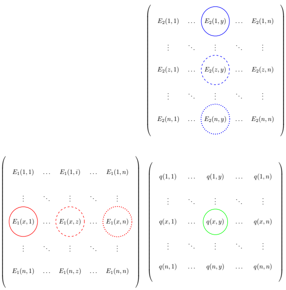
Compute the set of solutions

=

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



B : Adjacency matrix of E_2

A : Adjacency matrix of E_1

C : Result matrix

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

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

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.

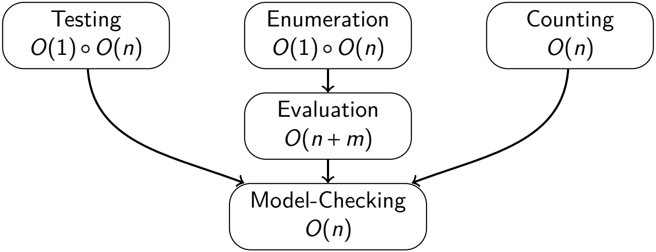
What kind of restrictions?

No restriction on the database part	Highly expressive queries (MSO queries)	FO queries
$\Downarrow$	$\Downarrow$	$\Downarrow$
Only works for a **strict** subset of ACQ	Only works for trees (graphs with bounded tree width)	**This thesis!**
Bagan, Durand, Grandjean	Courcelle, Bagan, Segoufin, Kazana	

Comparing the problems

For FO queries over a class $\mathscr{C}$ of databases.

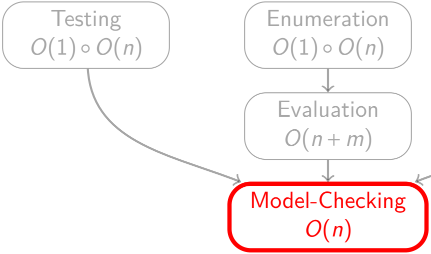
			Ideal running time
Model-Checking	:	Is this true ?	$O(n)$
Enumeration	:	Enumerate the solutions	$O(1) \circ O(n)$
Evaluation	:	Compute the entire set	$O(n+m)$
Counting	:	How many solutions ?	$O(n)$
Testing	:	Is this tuple a solution ?	$O(1) \circ O(n)$



Comparing the problems

For FO queries over a class $\mathscr{C}$ of databases.

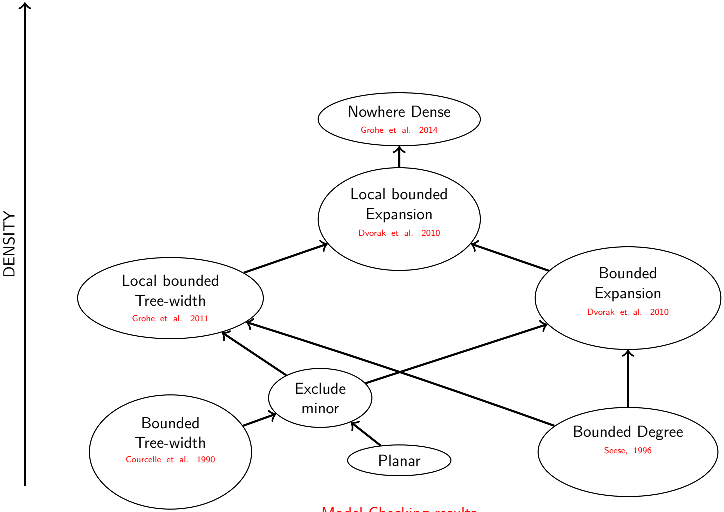
			Ideal running time
Model-Checking	:	Is this true ?	$O(n)$
Enumeration	:	Enumerate the solutions	$O(1) \circ O(n)$
Evaluation	:	Compute the entire set	$O(n+m)$
Counting	:	How many solutions ?	$O(n)$
Testing	:	Is this tuple a solution ?	$O(1) \circ O(n)$



AW[$*$] complete problem!

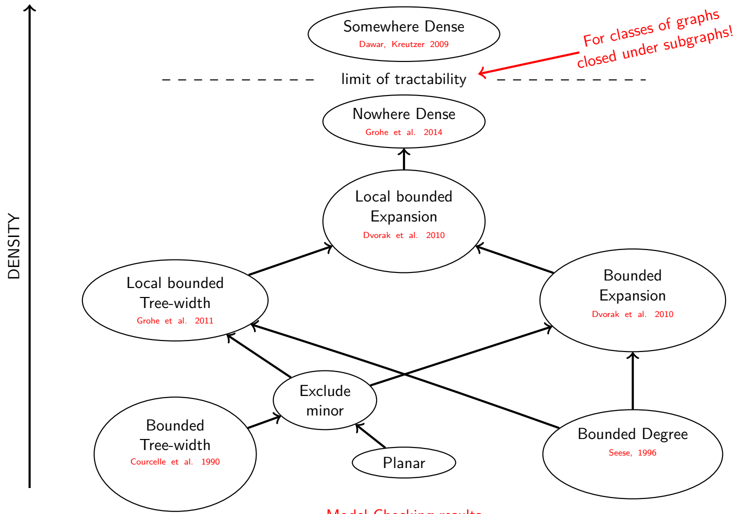
(when no restriction)

Classes of graphs



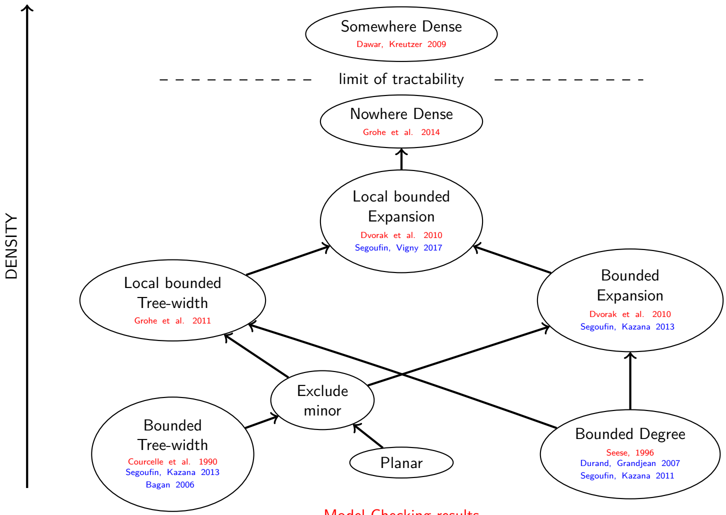
Model-Checking results

Classes of graphs



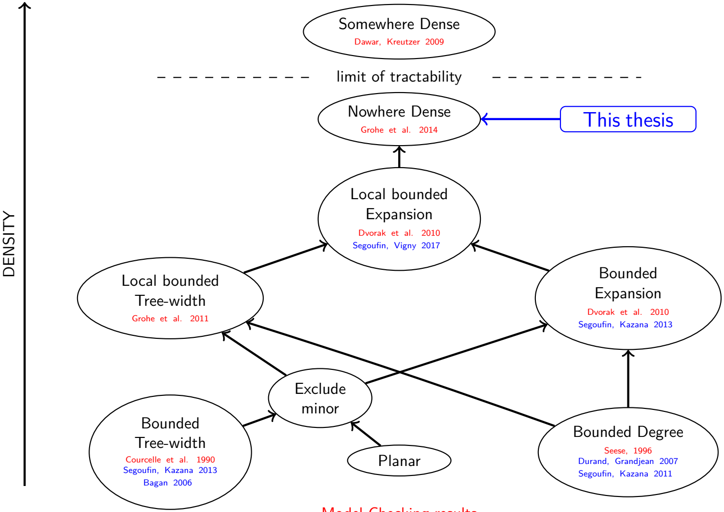
Model-Checking results

Classes of graphs



Model-Checking results
Enumeration results

Classes of graphs



Model-Checking results
Enumeration results

Outline

Nowhere dense graphs

Defined by Nešetřil and Ossona de Mendez.[1]

Examples:

- → Graphs with bounded degree
- → Graphs with bounded tree-width
- → Planar graphs
- → Graphs that exclude a minor

Can be defined using:

- → An ordering of vertices with good properties
- → A winning strategy for some two player game

⋮

[1] On nowhere dense graphs '11

Definition with a game

Definition : (ℓ, r)-Splitter game[1]

A graph G and two players, Splitter and Connector.
Each turn:

Connector picks a node c

Splitter picks a node s

$G := N_r^G(c) \setminus s$

If in less than ℓ rounds the graph is empty, Splitter wins.

[1] Grohe, Kreutzer, Siebertz STOC '14

Definition with a game

Definition : (ℓ, r)-Splitter game[1]

A graph G and two players, Splitter and Connector.
Each turn:

Connector picks a node c

Splitter picks a node s

$G := N_r^G(c) \setminus s$

If in less than ℓ rounds the graph is empty, Splitter wins.

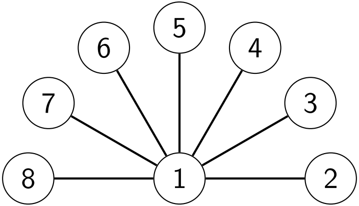
Theorem[1]

$\mathscr{C}$ nowhere dense $\iff \exists f_{\mathscr{C}},\ \ \forall G \in \mathscr{C},\ \ \forall r \in \mathbb{N}$:

Splitter has a wining strategy for the $(f_{\mathscr{C}}(r), r)$-splitter game on G.

[1] Grohe, Kreutzer, Siebertz STOC '14

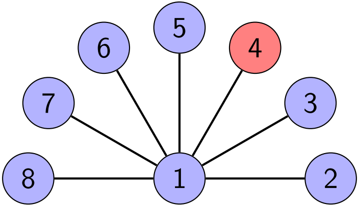
Splitter game on stars



Every edge goes to 1

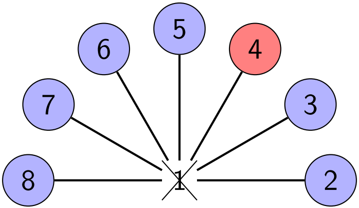
We are playing with $r > 1$

Splitter game on stars



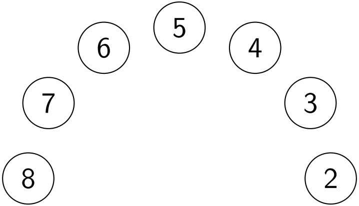
Connector picks 4

Splitter game on stars



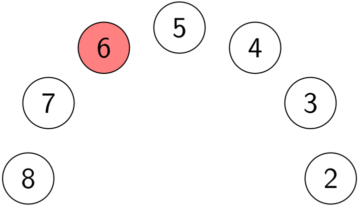
Splitter picks 1

Splitter game on stars



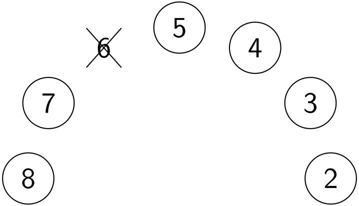
Here is the graph after one round.

Splitter game on stars



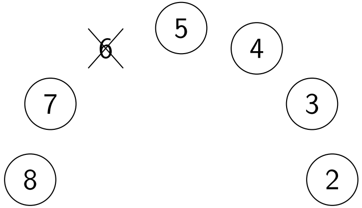
Connector picks 6

Splitter game on stars



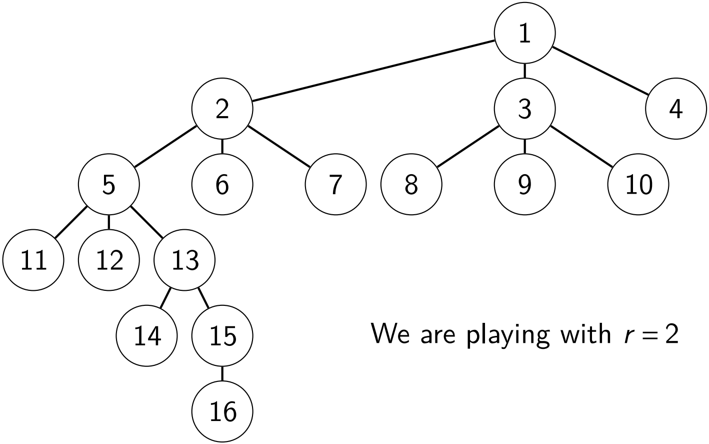
Splitter picks 6

Splitter game on stars



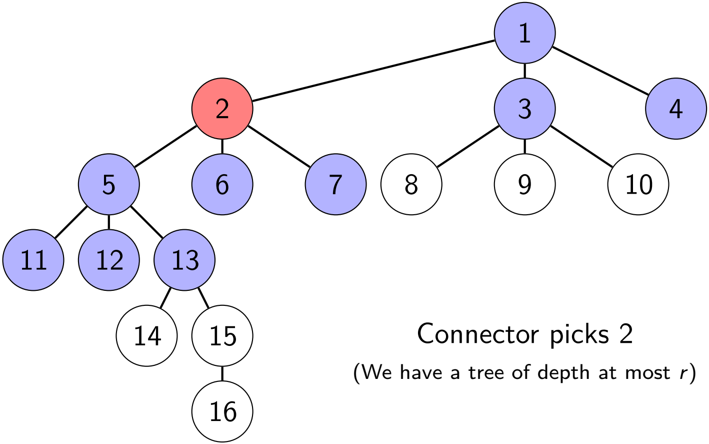
For every $r \in \mathbb{N}$ and every star G
Splitter wins the $(2, r)$-splitter game on G

Splitter game on trees



We are playing with $r = 2$

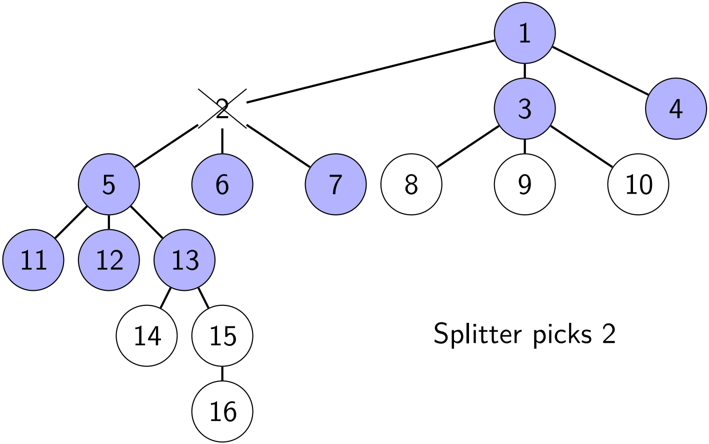
Splitter game on trees



Connector picks 2

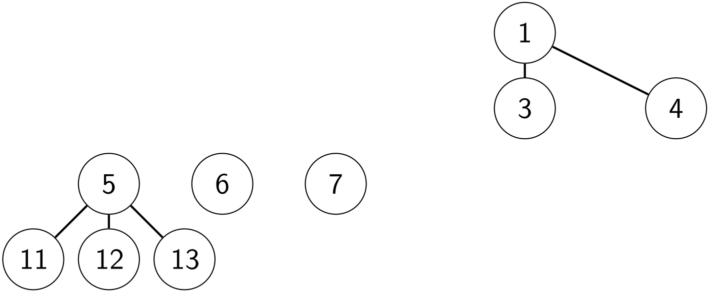
(We have a tree of depth at most r)

Splitter game on trees



Splitter picks 2

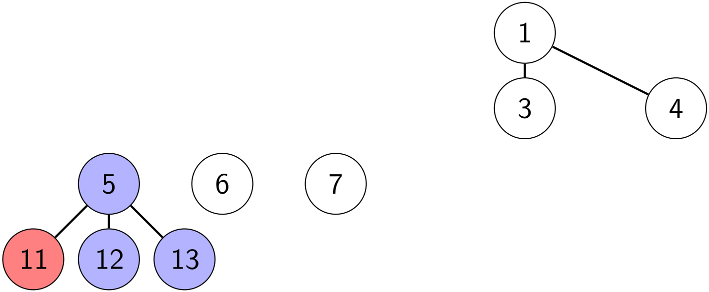
Splitter game on trees



Here is the graph after one round.

(Sevral trees of depth bounded by $r-1$)

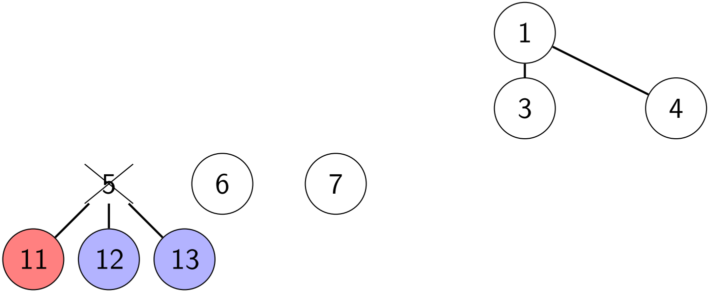
Splitter game on trees



Connector picks 11

(One of the tree of depth $r - 1$)

Splitter game on trees



Splitter picks 5

Splitter game on trees

Here is the graph after two rounds.

(Sevral trees of depth bounded by $r-2$)

Splitter game on trees

For every $r \in \mathbb{N}$ and tree G:
Splitter wins the $(r+1, r)$-splitter game on G.

Splitter game on other classes

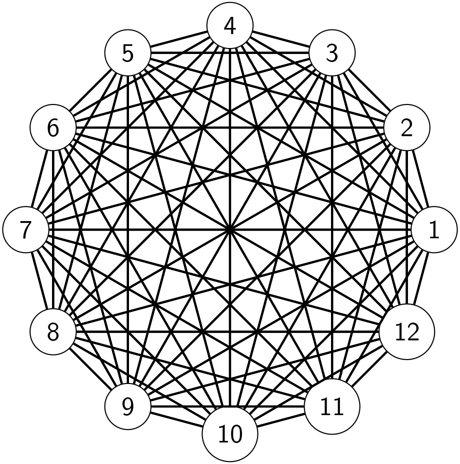
For every $r \in \mathbb{N}$ and every *path* G:

Splitter wins the $(\log(r)+1, r)$-splitter game on G.

For every $r \in \mathbb{N}$, $d \in \mathbb{N}$ and graph G with *degree bounded by d*:

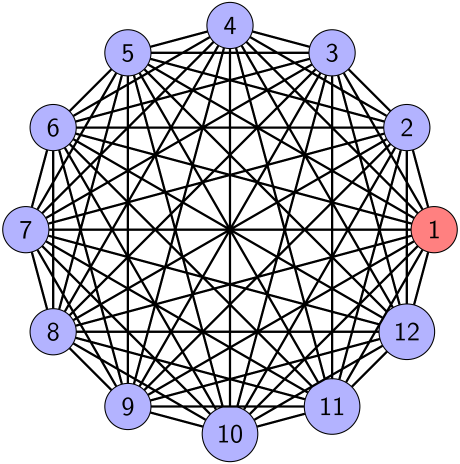
Splitter wins the (d^r+1, r)-splitter game on G.

Splitter game on cliques



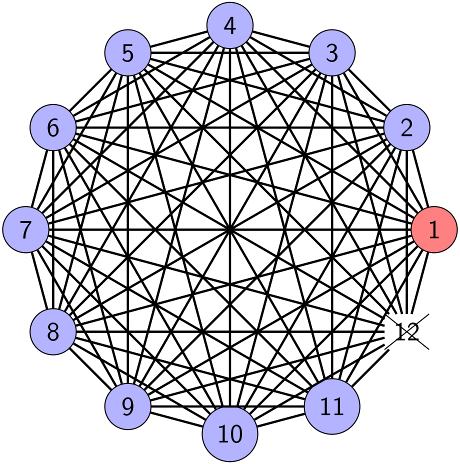
Every pair of nodes is an edge

Splitter game on cliques



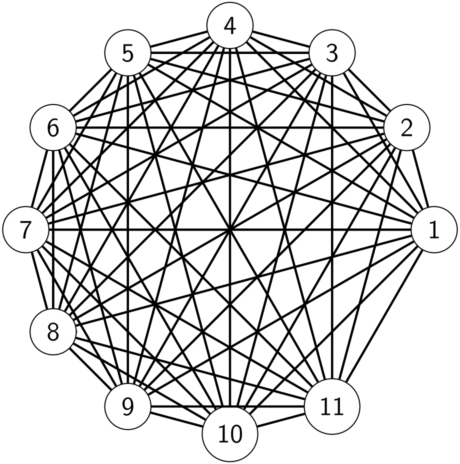
Connector picks 1

Splitter game on cliques



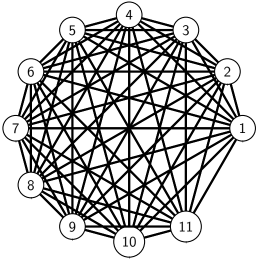
Splitter picks 12

Splitter game on cliques



We have a clique of size $n-1$.

Splitter game on cliques



We have a clique of size $n-1$.

If the number of rounds $<$ size of the clique, Splitter looses.
For $r=1$, $\forall \ell \in \mathbb{N}$ there is a clique G:

<u>Connector</u> wins the $(\ell,1)$-splitter game on G.

Outline

Results

Theorem: Schweikardt, Segoufin, Vigny

Over *nowhere dense* classes of graphs, for every FO query, after a *pseudo-linear* preprocessing, we can:

→ enumerate every solution with constant delay.

→ test whether a given tuple is a solution in constant time.

Theorem: Grohe, Schweikardt *(alternative proof, Vigny)*

Over *nowhere dense* classes of graphs, for every FO query, we can count the number of solutions in *pseudo-linear* time

Pseudo-linear?

Definition

An algorithm is pseudo linear if:

$$\forall \epsilon > 0, \quad \exists N_\epsilon : \begin{cases} \|G\| \leq N_\epsilon \implies \text{Brut force: } O(1) \\ \|G\| > N_\epsilon \implies O(\|G\|^{1+\epsilon}) \end{cases}$$

Examples: $O(n)$, $O(n\log(n))$, $O(n\log^i(n))$

Counter examples: $O(n^{1,0001})$, $O(n\sqrt{n})$

Scheme of proof

We use :

→ A new Hanf normal form for FO queries.[1]
To shape every query into local queries.

→ The algorithm for the model checking.[2]
For the base case of the induction.

→ Game characterization of nowhere dense classes.[2]
Gives us an inductive parameter.

→ Neighbourhood cover.[2]

→ Short-cut pointers dedicated to the enumeration.[3]

---

[1] Grohe, Schweikardt '18
[2] Grohe, Kreutzer, Siebertz '14
[3] Segoufin, Vigny '17

Neighborhood cover

A neighborhood cover is a set of "representative" neighborhoods.

$\mathscr{X} := X_1, \ldots, X_n$ is a *r-neighborhood cover* if it has the following properties:

- $\forall a \in G, \quad \exists X \in \mathscr{X}, \quad N_r(a) \subseteq X$
- $\forall X \in \mathscr{X}, \quad \exists a \in G, \quad X \subseteq N_{2r}(a)$
- the degree of the cover is: $\max_{a \in G} |\{i \mid a \in X_i\}|$.

Theorem: Grohe, Kreutzer, Siebertz '14

Over *nowhere dense* classes, for every r and ϵ, an r-neighborhood cover of degree $|G|^{\epsilon}$ can be computed in time $O(|G|^{1+\epsilon})$.

The examples queries

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

(The distance two query)

→ $q_2(x,y) := \neg q_1(x,y)$

(Nodes that are far apart)

For these queries we do not need the normal form

How to use the game 1/2

G is now fixed

Goal : Given a node a we want to enumerate all b such that $q_1(a, b)$.
(Here $r = 4$)

→ Base case: If Splitter wins the $(1, r)$-Splitter game on G.

Then G is edgeless and there is no solution!

→ By induction: assume that there is an algorithm for every G' such that Splitter wins the (ℓ, r)-Splitter game on G'.

How to use the game 2/2

Here, Splitter wins the $(\ell+1, r)$-game on G.

Idea :

- → Compute some new graph on which Splitter wins the (ℓ, r) game.
- → Alter the query and apply the algorithm given by induction.
  The solutions for the old query on the old graph and for the new query on the new graph must be the same.
- → Enumerate those solutions.

How to use the game 2/2

Here, Splitter wins the $(\ell + 1, r)$-game on G.

Idea :

- → Compute some new graph on which Splitter wins the (ℓ, r) game.
- → Alter the query and apply the algorithm given by induction.
  The solutions for the old query on the old graph and for the new query on the new graph must be the same.
- → Enumerate those solutions.

The new graph is a bag of the neighborhood cover.

For every $(a, b) \in G^2$ we have:

$$G \models q_1(a,b) \iff \bigvee_{X \in \mathscr{X}} X \models q_1(a,b) \iff \mathscr{X}(a) \models q_1(a,b)$$

How to use the game 2/2

Here, Splitter wins the $(\ell+1, r)$-game on G.

Idea :

- → Compute some new graph on which Splitter wins the (ℓ, r) game.
- → Alter the query and apply the algorithm given by induction.
  The solutions for the old query on the old graph and for the new query on the new graph must be the same.
- → Enumerate those solutions.

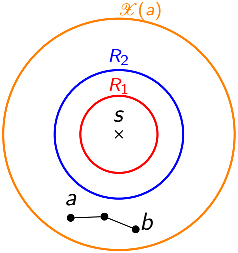
The new graph is a bag of the neighborhood cover.

For every $(a,b) \in G^2$ we have:

$$G \models q_1(a,b) \iff \bigvee_{X \in \mathscr{X}} X \models q_1(a,b) \iff \mathscr{X}(a) \models q_1(a,b)$$

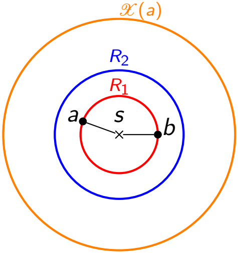
The new graph is $\mathscr{X}(a)$
Then, Splitter deletes a node!

The new queries



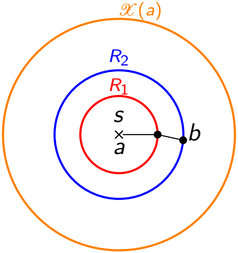
when there is still a 2-path not using s

the new query is:
$q_1(x, y)$



when s is on the only short path from a to b

the new query is:
$R_1(x) \wedge R_1(y)$



when $a = s$
(similarly for $b = s$)

the new query is:
$R_2(y)$

Running time

Without using the cover

- → To each a, associate $N_r^G(a) \setminus s_a$.
- → For every such graphs, compute the preprocessing given by induction.

Total running time: $\sum_{a \in G} \left(\left| N_r^G(a) \setminus s_a \right| \right) = O(|G|^2)$.

Using the cover

- → To each a, associate an $X \in \mathscr{X}$ such that $N_r^G(a) \subseteq X$.
- → To each X, associate the answer s_X of Splitter.
- → For every $X \setminus s_X$, compute the preprocessing given by induction.

Total running time: $\sum_{X \in \mathscr{X}} \left(\left| X \setminus s_X \right| \right) = O(|G|^{1+\epsilon})$

The second query

$q_2(x,y) := \mathrm{dist}(x,y) > 2$

Two kinds of solutions:

$b \in \mathscr{X}(a)$ (similar to the previous example)

$b \notin \mathscr{X}(a)$ We need something else !

The second query

$q_2(x,y) := \mathrm{dist}(x,y) > 2$

Two kinds of solutions:

$b \in \mathscr{X}(a)$	(similar to the previous example)
$b \notin \mathscr{X}(a)$	We need something else !

Goal: given a bag X, enumerate all $b \notin X$

The shortcut pointers

Given X we want to enumerate all b such that $b \not\in X$.

The shortcut pointers

Given X we want to enumerate all b such that $b \notin X$.

$$\underset{b \in X}{NEXT}(b, X) := \min\{b' \in G \mid b' \geq b \ \land \ b' \notin X\}$$

The shortcut pointers

Given X we want to enumerate all b such that $b \notin X$.

$$\underset{b \in X}{NEXT}(b, X) := \min\{b' \in G \mid b' \geq b \ \wedge \ b' \notin X\}$$

For all $X \in \mathscr{X}$ with $b_{max} \in X$, we have $NEXT(b_{max}, X) \ = \ NULL$

The shortcut pointers

Given X we want to enumerate all b such that $b \notin X$.

$$\underset{b \in X}{NEXT}(b, X) := \min\{b' \in G \mid b' \geq b \ \wedge \ b' \notin X\}$$

For all $X \in \mathscr{X}$ with $b_{max} \in X$, we have $NEXT(b_{max}, X) \ = \ NULL$

$$NEXT(b, X) \in \{b+1 \ , \ NEXT(b+1, X)\}$$

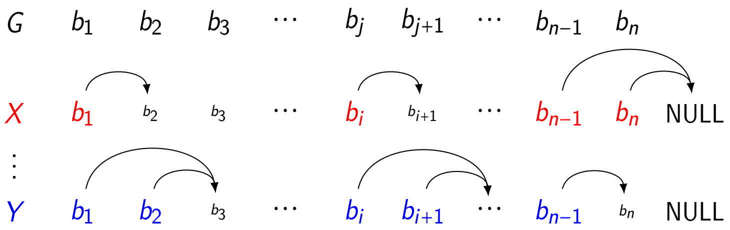
The shortcut pointers

Given X we want to enumerate all b such that $b \notin X$.

$$\underset{b \in X}{NEXT(b,X)} := \min\{b' \in G \mid b' \geq b \ \wedge \ b' \notin X\}$$

For all $X \in \mathscr{X}$ with $b_{max} \in X$, we have $NEXT(b_{max}, X) \ = \ NULL$

$$NEXT(b,X) \in \{b+1 \ , \ NEXT(b+1,X)\}$$



Outline

Recap

Theorem: Schweikardt, Segoufin, Vigny

Over *nowhere dense* classes of graphs, for every FO query, after a *pseudo-linear* preprocessing, we can:

→ enumerate every solution with constant delay.

→ test whether a given tuple is a solution in constant time.

Theorem: Grohe, Schweikardt *(alternative proof, Vigny)*

Over *nowhere dense* classes of graphs, for every FO query, we can count the number of solutions in *pseudo-linear* time.

Complexity

Pseudo-linear preprocessing: $O\big(f(|q|) \times |G|^{1+\epsilon}\big)$
But $f(\cdot)$ is a non-elementary function.

New directions

→ Classes of graphs that are not closed under subgraphs.[1]

→ Enumeration with update:
What happens if a small change occurs after the preprocessing ?

Existing results for: words,[2,3] *graphs with bounded degree*[4] *and* ACQ.[5]

[1] Gajarský, Kreutzer, Nešetřil, Ossona de Mendez, Pilipczuk, Siebertz, Toruńczyk ICALP '18
[2] Losemann, Martens CSL-LICS '14
[3] Niewerth, Segoufin PODS '18
[4] Berkholz, Keppeler, Schweikardt ICDT '17
[5] Berkholz, Keppeler, Schweikardt PODS '17 & ICDT '18

New directions

→ Classes of graphs that are not closed under subgraphs.[1]

→ Enumeration with update:
What happens if a small change occurs after the preprocessing ?

Existing results for: words,[2,3] *graphs with bounded degree*[4] *and* ACQ.[5]

Thank you!

[1] Gajarský, Kreutzer, Nešetřil, Ossona de Mendez, Pilipczuk, Siebertz, Toruńczyk ICALP '18
[2] Losemann, Martens CSL-LICS '14
[3] Niewerth, Segoufin PODS '18
[4] Berkholz, Keppeler, Schweikardt ICDT '17
[5] Berkholz, Keppeler, Schweikardt PODS '17 & ICDT '18