Query enumeration and nowhere dense graphs

Alexandre Vigny

February 15, 2019
Outline

Introduction
- Databases and queries
- Beyond query evaluation

Query enumeration
- Definition
- Examples
- Existing results

On nowhere dense graphs
- Definition and examples
- Splitter game

The algorithms
- Results and tools
- Examples

What’s next?

Conclusion
Outline

Introduction

Databases and queries
Beyond query evaluation

Query enumeration
Definition
Examples
Existing results

On nowhere dense graphs
Definition and examples
Splitter game

The algorithms
Results and tools
Examples

What’s next?

Conclusion
Databases and Queries

Input:

- Database \( D \) (contains informations)
- Query \( q \) (asks a question)

Goal:

Compute \( q(D) \)
Formalization part 1: Databases as relational structures

Schema: $\sigma := \{P(1), R(2), S(3)\}$

A relational structure $D$:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$R$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Paris</td>
<td>Alexandre</td>
</tr>
<tr>
<td>Poland</td>
<td>Bourg-la-Reine</td>
<td>Sophie</td>
</tr>
<tr>
<td>Germany</td>
<td>Warsaw</td>
<td>Tim</td>
</tr>
<tr>
<td>Italy</td>
<td>Meudon</td>
<td>Jack</td>
</tr>
<tr>
<td>Rome</td>
<td>Rome</td>
<td>Julie</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>France</th>
<th>Bourg-la-Reine</th>
<th>Poland</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>France</td>
<td>France</td>
<td>Meudon</td>
</tr>
<tr>
<td></td>
<td>Bourg-la-Reine</td>
<td>Bourg-la-Reine</td>
<td>Bourg-la-Reine</td>
</tr>
<tr>
<td></td>
<td>Warsaw</td>
<td>Paris</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td>Meudon</td>
<td>Paris</td>
<td>Paris</td>
</tr>
</tbody>
</table>
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?
$q(x, y) := \exists z \ R(x, z) \land R(y, z)$

Who are the people who do not work where they live?
$q(x) := \exists y \exists z \ S(x, y, z) \land y \neq z$

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
- Poland
- Italy

**R**
- Paris, France
- Meudon, France
- Warsaw, Poland
- Rome, Italy

**S**
- Jack, Warsaw, Paris

\[ P(x) \text{ becomes } \exists w, \text{ Blue}(w) \land B(x, w) \]
\[ R(x, y) \text{ becomes } \exists w, \text{ Red}(w) \land B(x, w) \land G(y, w) \]
\[ S(x, y, z) \text{ becomes } \exists w, \text{ Purple}(w) \land B(x, w) \land G(y, w) \land V(z, w) \]
\[ \exists x \cdots \text{ becomes } \exists x, \text{ Orange}(x) \land \cdots \]
\[ \forall x \cdots \text{ becomes } \forall x, \text{ Orange}(x) \Longrightarrow \cdots \]
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

Flight Warsaw-Paris
Inspiration from real world

Flight Warsaw-Paris

around 200,000 results in 0.5 seconds

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

Model-Checking : Is this true ?

\textbf{Input:} \quad \textbf{Goal:} \quad \textbf{Ideally:}

D, q

Yes or NO? D |= q ?

O(\|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: \[ D, q, \bar{a} \]
Goal: Test whether \( \bar{a} \in q(D) \).
Ideally: \( O(1) \circ O(\|D\|) \)

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?

\textit{Input:} \textit{Goal:} \textit{Ideally:}

\begin{align*}
\mathbf{D}, q & \quad \text{Compute } |q(\mathbf{D})| \quad O(\|\mathbf{D}\|)
\end{align*}

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: $D, q$

Goal: Compute $1^{st}$ sol, $2^{nd}$ sol, ...

Ideally: $O(1) \circ O(\|D\|)$
Comparing the problems

For FO queries over a class \( \mathcal{C} \) of databases.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Ideal running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-Checking</td>
<td>Is this true?</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Enumerate the solutions</td>
<td>( O(1) \circ O(n) )</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Compute the entire set</td>
<td>( O(n + m) )</td>
</tr>
<tr>
<td>Counting</td>
<td>How many solutions?</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Testing</td>
<td>Is this tuple a solution?</td>
<td>( O(1) \circ O(n) )</td>
</tr>
</tbody>
</table>
Outline

Introduction
- Databases and queries
- Beyond query evaluation

Query enumeration
- Definition
- Examples
- Existing results

On nowhere dense graphs
- Definition and examples
- Splitter game

The algorithms
- Results and tools
- Examples

What’s next?

Conclusion
Query enumeration

Input:  \( \|D\| := n \) &  \( |q| := k \)  
(computation with RAM)

Goal:  output solutions one by one  
(no repetition)

STEP 1: Preprocessing

Prepare the enumeration:  Database \( D \rightarrow Index \ I \)

Preprocessing time:  \( f(k) \cdot n \sim O(n) \)

STEP 2: Enumeration

The enumerate:  \( Index \ I \rightarrow \bar{x_1}, \bar{x_2}, \bar{x_3}, \bar{x_4}, \cdots \)

Delay:  \( O(f(k)) \sim O(1) \)

Constant delay enumeration after linear preprocessing  
\( O(1) \circ O(n) \)
Properties of efficient enumeration algorithms

Mandatory:

→ First solution computed in time $O(\|D\|)$.

→ Last solution computed in time $O(\|D\| + |q(D)|)$.

→ No repetition!

Optional:

→ Enumeration in lexicographical order.

→ Use a constant amount of memory.
Example 1

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

→ Query \( q_1(x, y) := E(x, y) \)

\[
E \\
(1,1) \\
(1,2) \\
(1,6) \\
\vdots \\
(4,5) \\
(4,7) \\
(4,8) \\
\vdots \\
(n,n)
\]
Example 1

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

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

→ Query $q_2(x, y) := \neg E(x, y)$

$E$

$(1,1)$
$(1,2)$
$(1,6)$
$\vdots$
$(2,3)$
$\vdots$
$(i,j)$
$(i,j+1)$
$(i,j+3)$
$\vdots$
$(n,n)$
Example 2

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

→ Query $q_2(x, y) := \neg E(x, y)$

$E$

\begin{align*}
(1,1) \\
(1,2) \\
(1,6) \\
\vdots \\
(2,3) \\
\vdots \\
(i,j) \\
(i,j+1) \\
(i,j+3) \\
\vdots \\
(n,n)
\end{align*}

For counting problem

Computation: Do the same algorithm!
Answering: $|q_2(D)| = n^2 - |q_1(D)|$

For the testing problem

Same difficulty!
\[ \bar{a} \in q_2(D) \iff \bar{a} \not\in q_1(D) \]

For the enumeration problem

We need something else!
Example 2

→ Database \( D := \langle \{1, \ldots, n\}; E \rangle \) \( \|D\| = |E| \)

→ Query \( q_2(x, y) := \neg E(x, y) \)

\[
E
\begin{array}{c|cccc|}
(1,1) & & & & \\
(1,2) & & & & \\
(1,6) & \vdots & & & \\
(2,3) & & & & \\
\vdots & & & & \\
(i,j) & & & & \\
(i,j+1) & & & & \\
(i,j+3) & & & & \\
\vdots & & & & \\
(n,n) & & & & \\
\end{array}
\]
Example 2

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

→ Query $q_2(x, y) := \neg E(x, y)$

<table>
<thead>
<tr>
<th>$E$</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1,1)$</td>
<td>$(1,1)$</td>
</tr>
<tr>
<td>$(1,2)$</td>
<td>$(1,2)$</td>
</tr>
<tr>
<td>$(1,6)$</td>
<td>$(1,6)$</td>
</tr>
<tr>
<td>$(2,3)$</td>
<td>$(2,3)$</td>
</tr>
<tr>
<td>$(i,j)$</td>
<td>$(i,j)$</td>
</tr>
<tr>
<td>$(i,j+1)$</td>
<td>$(i,j+1)$</td>
</tr>
<tr>
<td>$(i,j+3)$</td>
<td>$(i,j+3)$</td>
</tr>
<tr>
<td>$(n,n)$</td>
<td>$(n,n)$</td>
</tr>
</tbody>
</table>
Example 2

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

→ Query $q_2(x, y) := \neg E(x, y)$

<table>
<thead>
<tr>
<th>$E$</th>
<th>Index</th>
<th>Enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1)</td>
<td>(1,1)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(1,2)</td>
<td>(1,2)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>(1,6)</td>
<td>(1,6)</td>
<td>(1,6)</td>
</tr>
<tr>
<td>(2,3)</td>
<td>(2,3)</td>
<td>(2,4)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>(i,j)</td>
<td>(i,j)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(i,j+1)</td>
<td>(i,j+1)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>(i,j+3)</td>
<td>(i,j+3)</td>
<td>(1,4)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>(1,5)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>(1,6)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>(2,4)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>(2,5)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>(n,n)</td>
<td>(n,n)</td>
<td>NULL</td>
</tr>
</tbody>
</table>
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 $q(x, y) := \exists z, \ E_1(x, z) \land E_2(z, y)$
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 $q(x,y) := \exists z, E_1(x,z) \land E_2(z,y)$

$B$ : Adjacency matrix of $E_2$

$A$ : Adjacency matrix of $E_1$

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

\[ A : \text{Adjacency matrix of } E_1 \]
\[ B : \text{Adjacency matrix of } E_2 \]
\[ C : \text{Result matrix} \]

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

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

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

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

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

and $D$ is a tree!
Example 3 bis

→ Database $D := \langle \{1, \cdots, n\}; \mathcal{E}_1; \mathcal{E}_2 \rangle$ \quad $\|D\| = |\mathcal{E}_1| + |\mathcal{E}_2|$ \quad ($\mathcal{E}_i \subseteq D \times D$)

→ Query $q(x, y) := \exists z, \mathcal{E}_1(x, z) \land \mathcal{E}_2(z, y)$

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?**

<table>
<thead>
<tr>
<th>No restriction on the database part</th>
<th>Highly expressive queries (MSO queries)</th>
<th>FO queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only works for a strict subset of ACQ</td>
<td>Only works for trees (graphs with bounded tree width)</td>
<td><strong>This talk!</strong></td>
</tr>
</tbody>
</table>

Bagan, Durand, Grandjean

Courcelle, Bagan, Segoufin, Kazana
Comparing the problems

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Ideal running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-Checking</td>
<td>Is this true?</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Enumerate the solutions</td>
<td>$O(1) \cdot O(n)$</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Compute the entire set</td>
<td>$O(n + m)$</td>
</tr>
<tr>
<td>Counting</td>
<td>How many solutions?</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Testing</td>
<td>Is this tuple a solution?</td>
<td>$O(1) \cdot O(n)$</td>
</tr>
</tbody>
</table>

$O(n)$ complete problem!

(when no restriction)
Comparing the problems

For **FO queries** over a class \( \mathcal{C} \) of databases.

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 and FO queries

- Nowhere dense
  - Grohe et al. 2014
- Local bounded Expansion
  - Dvorak et al. 2010
- Local bounded Tree-width
  - Grohe et al. 2011
- Excluded minor
- Bounded Expansion
  - Dvorak et al. 2010
- Bounded Tree-width
  - Courcelle et al. 1990
- Planar
- Bounded Degree
  - Seese, 1996

Model-Checking results

For classes of graphs closed under subgraphs.

With: Nicole Schweikardt, Luc Segoufin
Classes of graphs and FO queries

- Somewhere Dense
  - Dawar, Kreutzer 2009

- Nowhere Dense
  - Grohe et al. 2014

- Local bounded Tree-width
  - Grohe et al. 2011

- Local bounded Expansion
  - Dvorak et al. 2010

- Bounded Expansion
  - Dvorak et al. 2010

- Bounded Degree
  - Seese, 1996

- Excludeminor

- Planar

For classes of graphs closed under subgraphs! Limit of tractability

DENSITY

Model-Checking results
Classes of graphs and FO queries

- Somewhere Dense
  - Dawar, Kreutzer 2009
- Local bounded Expansion
  - Dvorak et al. 2010
  - Segoufin, Kazana 2013
  - Guro, Kazana 2017
- Bounded Expansion
  - Dvorak et al. 2010
  - Segoufin, Kazana 2013
- Bounded Degree
  - Seese, 1996
  - Durand, Grandjean 2007
  - Segoufin, Kazana 2011

Model-Checking results
- Enumeration results

- Nowhere Dense
  - Grohe et al. 2014

- Bounded Degree
  - Courcelle et al. 1990
  - Segoufin, Kazana 2013
  - Bagan 2006

DENSITY
Classes of graphs and FO queries

- Somewhere Dense
  - Dawar, Kreutzer 2009

- Nowhere Dense
  - Grohe et al. 2014

  - limit of tractability

- Local bounded Expansion
  - Dvorak et al. 2010
  - Segoufin, Vigny 2017

- Bounded Expansion
  - Dvorak et al. 2010
  - Segoufin, Kazana 2013

- Bounded Tree-width
  - Courcelle et al. 1990
  - Segoufin, Kazana 2013
  - Bagan 2006

- Bounded Expansion
  - Dvorak et al. 2010
  - Segoufin, Kazana 2013

- Local bounded Tree-width
  - Grohe et al. 2011

- Excluded minor

- Exclude minor

- Planar

- Bounded Degree
  - Seese, 1996
  - Durand, Grandjean 2007
  - Segoufin, Kazana 2011

Model-Checking results
Enumeration results

With:
Nicole Schweikardt
Luc Segoufin
Outline

Introduction
- Databases and queries
- Beyond query evaluation

Query enumeration
- Definition
- Examples
- Existing results

On nowhere dense graphs
- Definition and examples
- Splitter game

The algorithms
- Results and tools
- Examples

What’s next?

Conclusion
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

¹On nowhere dense graphs ’11
Definition with a game

Definition: \((\ell, r)\)-Splitter game\(^1\)

A graph \(G\) and two players, Splitter and Connector.

Each turn:

- Connector picks a node \(c\)
- Splitter picks a node \(s\)

\(G := N^G_r(c) \setminus s\)

If in less than \(\ell\) rounds the graph is empty, Splitter wins.

\(^1\)Grohe, Kreutzer, Siebertz STOC ’14
Definition with a game

Definition: \((\ell, r)\)-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 \(\ell\) rounds the graph is empty, Splitter wins.

Theorem\(^1\)

\(\mathcal{C}\) nowhere dense \(\iff\) \(\exists f_{\mathcal{C}}, \forall G \in \mathcal{C}, \forall r \in \mathbb{N}:\)

Splitter has a winning strategy for the \((f_{\mathcal{C}}(r), r)\)-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
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.
(Several 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 \((d' + 1, r)\)-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 $< \text{size of the clique}$, Splitter looses.

For $r = 1$, $\forall \ell \in \mathbb{N}$ there is a clique $G$:

**Connector** wins the $(\ell,1)$-splitter game on $G$. 
Introduction

- Databases and queries
- Beyond query evaluation

Query enumeration
- Definition
- Examples
- Existing results

On nowhere dense graphs
- Definition and examples
- Splitter game

The algorithms
- Results and tools
- Examples

What's next?

Conclusion
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 \textit{(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:

\[
\begin{align*}
\forall \varepsilon > 0, \quad \exists N_\varepsilon : \\
\|G\| \leq N_\varepsilon \Rightarrow & \quad \text{Brut force: } O(1) \\
\|G\| > N_\varepsilon \Rightarrow & \quad O(\|G\|^{1+\varepsilon})
\end{align*}
\]

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.

\( \mathcal{X} := X_1, \ldots, X_n \) is a \( r \)-neighborhood cover if it has the following properties:

\[
\begin{align*}
\rightarrow & \quad \forall a \in G, \ \exists X \in \mathcal{X}, \ N_r(a) \subseteq X \\
\rightarrow & \quad \forall X \in \mathcal{X}, \ \exists a \in G, \ X \subseteq N_{2r}(a)
\end{align*}
\]

\( \rightarrow \) 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 \( \epsilon \), an \( r \)-neighborhood cover of degree \( |G|^\epsilon \) can be computed in time \( O(|G|^{1+\epsilon}) \).
The examples queries

\[ q_1(x, y) := \exists z \ E(x, z) \land E(z, y) \]

(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 $(\ell, r)$-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 \((\ell, r)\) 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 \((\ell, r)\) 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:

\[
G \models q_1(a, b) \iff \bigvee_{X \in \mathcal{X}} X \models q_1(a, b) \iff \mathcal{X}(a) \models q_1(a, b)
\]
How to use the game 2/2

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

Idea:

\(\rightarrow\) Compute some new graph on which Splitter wins the \((\ell, r)\) game.

\(\rightarrow\) 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.*

\(\rightarrow\) Enumerate those solutions.

The new graph is a bag of the neighborhood cover.

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

\[
G \models q_1(a, b) \iff \bigvee_{X \in \mathcal{X}} X \models q_1(a, b) \iff \mathcal{X}(a) \models q_1(a, b)
\]

The new graph is \(\mathcal{X}(a)\)

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: $R_1(x) \land R_1(y)$

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

Using the cover

→ To each $a$, associate an $X \in \mathcal{X}$ 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: $\sum_{X \in \mathcal{X}} \left( |X \setminus s_X| \right) = O(|G|^{1+\epsilon})$
The second query

\[ q_2(x, y) := \text{dist}(x, y) > 2 \]

Two kinds of solutions:

\[ b \in \mathcal{X}(a) \quad \text{(similar to the previous example)} \]

\[ b \not\in \mathcal{X}(a) \quad \text{We need something else!} \]
The second query

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

Two kinds of solutions:

- $b \in \mathcal{K}(a)$ (similar to the previous example)
- $b \not\in \mathcal{K}(a)$ We need something else!

Goal: given a bag $X$, enumerate all $b \not\in X$
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 $b \notin X$.

$$\text{NEXT}(b, X) := \min_{b \in X} \{b' \in G \mid b' \geq b \land b' \notin X\}$$
The shortcut pointers

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

$$NEXT(b, X) := \min_{b \in X} \{ b' \in G \mid b' \geq b \land b' \not\in X \}$$

For all $X \in \mathcal{X}$ with $b_{\text{max}} \in X$, we have $NEXT(b_{\text{max}}, X) = \text{NULL}$
The shortcut pointers

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

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

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

$$\text{NEXT}(b, X) \in \{b + 1, \text{NEXT}(b + 1, X)\}$$
The shortcut pointers

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

\[
\text{NEXT}(b, X) := \min\{ b' \in G \mid b' \geq b \land b' \not\in X \}
\]

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

\[
\text{NEXT}(b, X) \in \{ b + 1 \, , \, \text{NEXT}(b + 1, X) \}
\]
Outline

Introduction
- Databases and queries
- Beyond query evaluation

Query enumeration
- Definition
- Examples
- Existing results

On nowhere dense graphs
- Definition and examples
- Splitter game

The algorithms
- Results and tools
- Examples

What’s next?

Conclusion
Dense classes of graphs

Requirers:
Classes of graphs that are not closed under subgraphs.

Idea: Interpretation of classes of graphs
- $G = (V, E) \leadsto G' = (V, E')$ where $E' = \phi(G)$
- dual of graphs
- power of graphs

Results for classes with:
- Bounded degree\(^1\)
- Bounded expansion\(^2\)
- Nowhere dense ?

\(^1\)Gajarský, Hlinený, Obdrzálek, Lokshtanov, Ramanujan LICS ’16
\(^2\)Gajarský, Kreutzer, Nešetřil, Ossona de Mendez, Pilipczuk, Siebertz, Toruńczyk ICALP ’18
Different restrictions

What about other query languages?

- **Beyon FO queries:**
  - FO + Mod\(^1\)
  - FO + Count\(^2\)
  - FO + ?

- **Bellow FO queries:**
  - Dichotomy for CQ?\(^3\)

---

\(^1\)Berkholz, Keppeler, Schweikardt  ICDT '17
\(^2\)Grohe, Schweikardt  PODS '18
\(^3\)Bagan, Durand, Filiot, Gauwin  CSL '10
Updates

- What happens if a small change occurs after the preprocessing?
  
  Solution 1: Start the preprocessing from scratch.
  Solution 2: Be smarter!

- Goal: update in $O(1)$ or $O(\log(n))$.

Existing results for: words,$^1,^2$ graphs with bounded degree $^3$ and ACQ.$^4$

- What remains?
  
  nowhere dense classes of graphs
  classes of graphs with low degree
  more powerful updates

---

$^1$Losemann, Martens  CSL-LICS ’14
$^2$Niewerth, Segoufin  PODS ’18
$^3$Berkholz, Keppeler, Schweikardt  ICDT ’17
$^4$Berkholz, Keppeler, Schweikardt  PODS ’17 & ICDT ’18
Outline

Introduction
  Databases and queries
  Beyond query evaluation

Query enumeration
  Definition
  Examples
  Existing results

On nowhere dense graphs
  Definition and examples
  Splitter game

The algorithms
  Results and tools
  Examples

What’s next?

Conclusion
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: $O(f(|q|) \times |G|^{1+\epsilon})$
But $f(\cdot)$ is a non-elementary function.
The end

Thank you!

Any question?