Query enumeration & Nowhere-dense graphs

Alexandre Vigny

January 06, 2017

Outline

1. Introduction
2. What is query enumeration ?
3. Constant-Delay
4. Structural restrictions
5. Bounded expansion and local bounded expansion
6. Our results
7. Conclusion

Outline

Introduction, Query evaluation

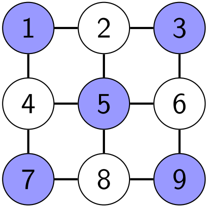
- Query q *small*
- Database D *huge*
- Compute $q(D)$ *gigantic*

Examples :

query q

$$q(x,y) := \exists z(B(x) \wedge E(x,z) \wedge \neg E(y,z))$$

database D



solutions $q(D)$

{(1,2) (1,3) (1,4)
(1,6) (1,7) $\cdots$
(3,1) (3,2) (3,4)
(3,6) (3,7) $\cdots$
$\cdots$ }

Remarks

- Databases = relational structures ≈ coloured graphs
- Query are implicitly **FO** queries
- The arity of q is r. $q(D)$ can blow up to $|D|^r$
  - To big to be computed.
  - The size of the out put does not reflect the difficulty of the task.

Outline

Enumeration

- What is query enumeration ?

  Input : Database : **D** & Query : q

  Goal : output solutions one by one

  $$\overline{a}_1 \longrightarrow \overline{a}_2 \longrightarrow \overline{a}_3 \longrightarrow \ldots \longrightarrow \overline{a}_m$$

- Why ?
  - Not all results at the same time
  - Only few solutions are needed
  - Applications for other task ? (Streaming, ...)

The delay

Definition

The delay is the maximal time between to consecutive output.

The delay can depends on :

1) The size of the query
2) The size of the database
3) The number of solutions that have already been outputed

We will focus on **Constant delay** :

The delay only depends on the size of the query.

In data complexity $O(f(|q|)) = O(1)$

Outline

Linear preprocessing and constant delay

Definition

The preprocessing is the time needed to compute the first solution.

- STEP 1: Preprocessing

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

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

- STEP 2 : Enumeration

  Enumerate the solutions : Index $I \longrightarrow \overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \cdots$

  Delay : $O(f(|q|)) \rightsquigarrow O(1)$

Constant delay enumeration after linear preprocessing $(CD \circ Lin)$

Properties

- The first solution is computed in time $O(\|D\|)$
- The last solution is computed in time $O(\|D\| + |q(D)|)$
- Some possible improvements :
  - Lexicographical enumeration
  - Start the enumeration at a given tuple

Example 1

Input :

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

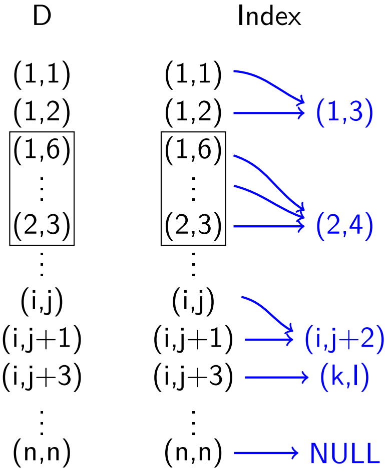
D

(1,1)
(1,2)
(1,6)
⋮
(2,3)
⋮
(i,j)
(i,j+1)
(i,j+3)
⋮
(n,n)

Example 1

Input :

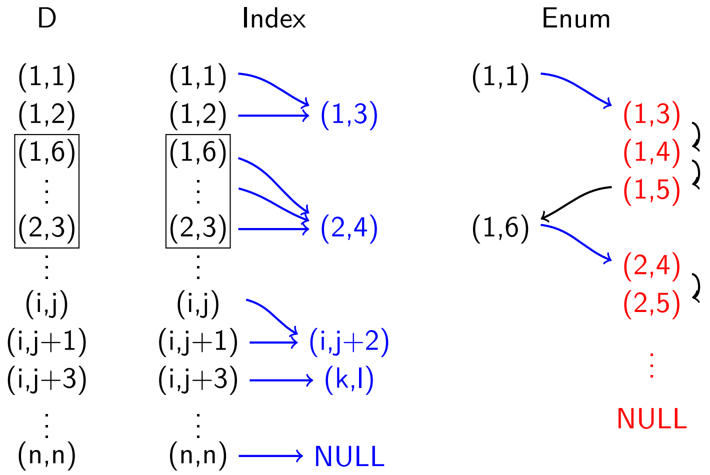
- Database $D := \langle \{1, \cdots, n\}; E \rangle$ $\qquad \|D\| = |E| \quad (E \subseteq D \times D)$
- Query $q(x, y) := \neg E(x, y)$



Example 1

Input :

- Database $D := \langle \{1, \cdots, n\}; E \rangle$ $\quad \|D\| = |E| \quad (E \subseteq D \times D)$
- Query $q(x, y) := \neg E(x, y)$



Example 2

Input :

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

Not in $CD \circ Lin$.
Unless the $n \times n$ boolean matrix multiplication is doable in time $O(n^2)$.

$$\Downarrow$$

Restricted databases or/and queries

Example : Graphs with bounded degree.

Example 2a

$\mathcal{C}$ a class of graph with bounded degree d.

Input :

- Graph $G := (V, E)$ a graph of $\mathcal{C}$.
- Query $q(x, y) := \exists z\ E(x, z) \wedge E(z, y)$.

Then the algorithm becomes easy :

- Preprocessing : We can compute the list of all solutions ! Because for each $a \in V$, $|N_2^G(a)| \leq d^2$.

- Enumeration : We just have to read the list.

This can be generalized for every **FO** queries.[1]

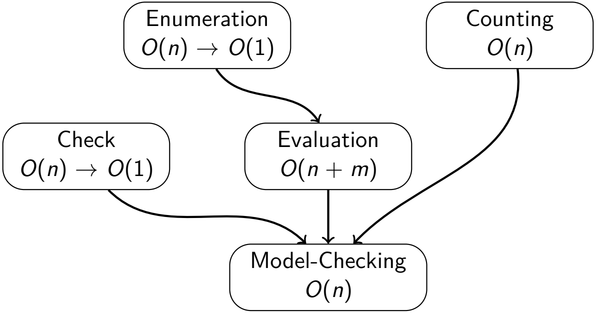
[1] Seese'96, Durand, Grandjean'07, Segoufin, Kazana'11

Outline

Other problems

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

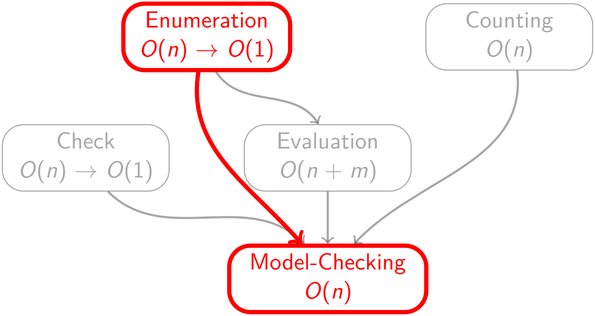
Model-Checking	:	Is this true ?	$O(n)$
Counting	:	How many solutions ?	$O(n)$
Check	:	Is this tuple a solution ?	$O(n) \rightarrow O(1)$
Evaluation	:	Compute the entire set	$O(n+m)$



Other problems

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

Model-Checking	:	Is this true ?	$O(n)$
Counting	:	How many solutions ?	$O(n)$
Check	:	Is this tuple a solution ?	$O(n) \rightarrow O(1)$
Evaluation	:	Compute the entire set	$O(n+m)$



Restrictions are needed

Constant-delay Enumeration $\Longrightarrow$ Linear Model-Checking

Under some complexity hypothesis, the Model-Checking is not doable in polynomial time.[1]

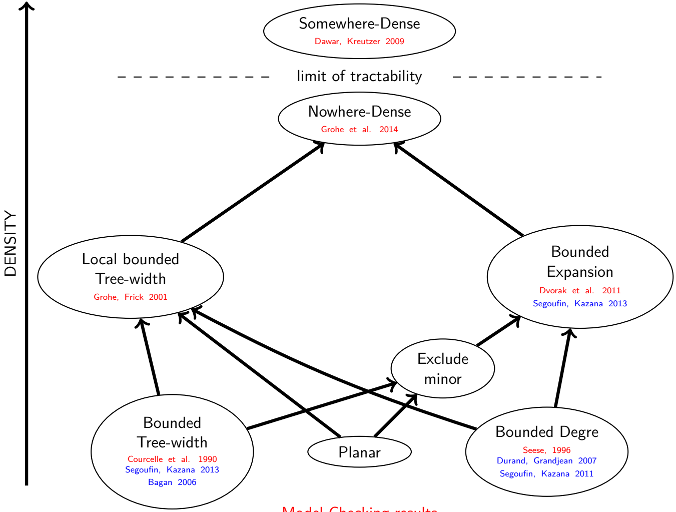
$$\Downarrow$$

Restricted databases or/and queries

Bonded degree, planar $\cdots$	MSO, quantifier free $\cdots$
Nowhere-dense	**First Order**

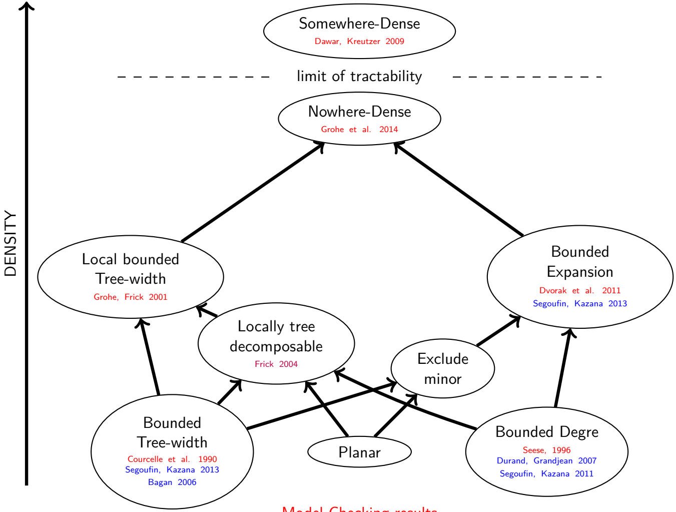
[1] FPT $\neq$ AW[$*$]

Hereditary classes of graphs and FO queries



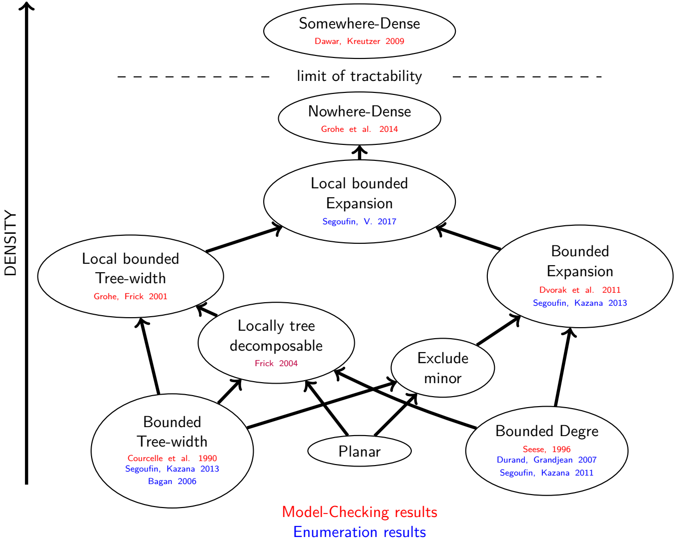
Model-Checking results
Enumeration results

Hereditary classes of graphs and FO queries



Model-Checking results
Enumeration results

Hereditary classes of graphs and FO queries



Outline

Definitions bounded expansion

There are many equivalent definitions:
Winning strategies, asymptotic ratio edges/vertices, good ordering...

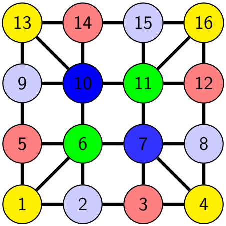
Definition : Bounded expansion

$\mathcal{C}$ has *bounded expansion* if and only if for all $k \in \mathbb{N}$, there is a $N_k \in \mathbb{N}$, such that for all $G \in \mathcal{C}$, G admit a *k-tree width colouring* using N_k colors.

tree-width colouring

Definition : k-tree width colouring with N colors

G is k-tree width coloured if and only if it is coloured (with less than N colors) and for all $H \subseteq G$, if H use less than k colours, then tree-width$(H) \leq k$

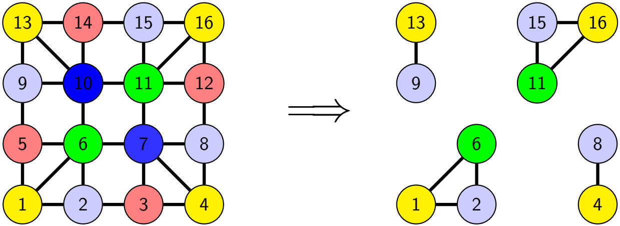


4-tree widh colouring with 5 colors

tree-width colouring

Definition : k-tree width colouring with N colors

G is k-tree width coloured if and only if it is coloured (with less than N colors) and for all $H \subseteq G$, if H use less than k colours, then tree-width$(H) \leq k$

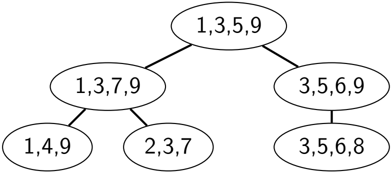


4-tree widh colouring with 5 colors

choosing 3 colours

Example: graph with bounded tree width

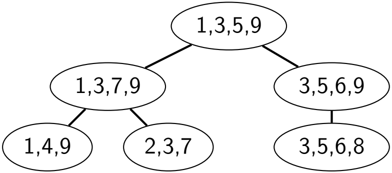
If G has tree width t, then for every k, we chose $N_k := t + 1$.



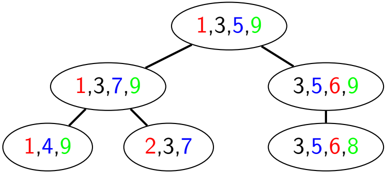
Tree-width $t = 3$

Example: graph with bounded tree width

If G has tree width t, then for every k, we chose $N_k := t + 1$.



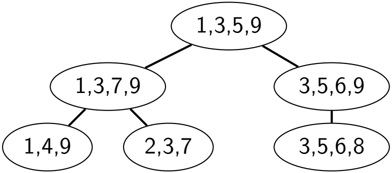
Tree-width $t = 3$



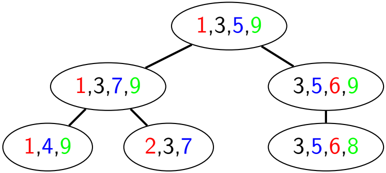
Colouring using $N_k = t + 1 = 4$ colors

Example: graph with bounded tree width

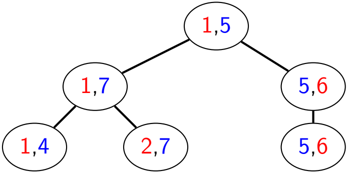
If G has tree width t, then for every k, we chose $N_k := t + 1$.



Tree-width $t = 3$



Colouring using $N_k = t + 1 = 4$ colors



Choosing $k = 2$ colors

Example: graph with bounded degree

If G has degree d, then for every k, we chose $N_k := d^k$.

Algorithm: for all $a \in G$, chose a color **not used** in is k neighbourhood.
We now chose I a subset of k-1 colors. $H = G[I]$.

Example: graph with bounded degree

If G has degree d, then for every k, we chose $N_k := d^k$.

Algorithm: for all $a \in G$, chose a color **not used** in is k neighbourhood.
We now chose I a subset of k-1 colors. $H = G[I]$.

a) There is no path of length k in H.

   proof: If there is such path, then atleast two nodes must have the same color, but they are to close !

Example: graph with bounded degree

If G has degree d, then for every k, we chose $N_k := d^k$.

Algorithm: for all $a \in G$, chose a color **not used** in is k neighbourhood.
We now chose I a subset of k-1 colors. $H = G[I]$.

a) There is no path of length k in H.

   proof: If there is such path, then atleast two nodes must have the same color, but they are to close !

b) Every connected component in H have less than k elements.

   proof: Let a and b be in the same component.
   By a), $\mathrm{dist}^H(a, b) < k$. Hence $\mathrm{col}(a) \neq \mathrm{col}(b)$. And we have chosen only k-1 colors.

Results

Theorem (Dvorak et al. '11))

*Model-checking for **FO** queries can be answered in time $O(|G|)$*

Theorem (Kazana,Segoufin '13))

*Enumeration for **FO** queries is doable with constant delay after linear preprocessing.*

Stronger result : After a linear preprocessing, given any tuple $\overline{a}$, the smallest solution $\overline{b} \geq \overline{a}$ is computable in constant time.

Theorem (Kazana,Segoufin '13))

*Counting for **FO** queries is doable in time $O(|G|)$*

Definition local bounded expansion

Definition : Class of r-neighbourhoods

Let $\mathcal{C}$ be a class of graphs, $r \in \mathbb{N}$. $\mathcal{C}_r := \{H \mid \exists G \in \mathcal{C}\ \exists a \in G\ H \subseteq N_r^G(a)\}$

Definition : Local bounded expansion

Let $\mathcal{C}$ be a class of graphs, $\mathcal{C}$ has locally bounded expansion if and only if for all integer r, $\mathcal{C}_r$ has bounded expansion.

The expansion of a neighbourhood only depends on it's radius.

Some properties

Let $\mathcal{C}$ be a class of graph with local bounded expansion.

The number of edges is *pseudo-linear*.

$$\forall e > 0,\ \exists N_\epsilon \in \mathbb{N},\ \forall G \in \mathcal{C},\ |G| > N_\epsilon \Longrightarrow \mathrm{edge(G)} \in O(|G|^{1+\epsilon})$$

$$n \ll n\log(n) \ll n\log^i(n) \ll \text{pseudo-linear} \ll n\sqrt{n}$$

$$\text{“Pseudo linear} \approx n\log^i(n)\text{”}$$

$$\text{“Pseudo constant} \approx \log^i(n)\text{”}$$

Other properties

For every integer k every graph G can be k-tree width coloured using a pseudo-constant number of colors.

Theorem (Grohe et al. 2014)

The model-checking for nowhere-dense classes of graphs is pseudo-linear.

We are aiming at pseudo-linear preprocessing but still constant delay.

Outline

Our results

Theorem (Segoufin, V. '17)

*Over classes of graphs with local bounded expansion, every **FO** queries can be enumerate with constant delay after a pseudo-linear preprocessing.*

Theorem (Segoufin, V. '17)

*Over classes of graphs with local bounded expansion, the counting problem for **FO** queries is pseudo-linear.*

Tools

- Gaifman theorem.
- Neighbourhood cover.[1]
- Enumeration for graphs with **Bounded expansion**.
- Short-cut pointers dedicated to the enumeration.

[1] Grohe et al. '14

Sketch of proof (1/4) : Gaifman theorem

Theorem (Gaifman)

Every first order query is a combination of sentences and local queries.

$$q(x, y) := q_1(x) \wedge q_2(y) \wedge \mathrm{dist}(x, y) > 2r$$

Where q_1 and q_2 are r-local queries.

$$G \models q_1(a) \iff N_r^G(a) \models q_1(a)$$

Sketch of proof (2/4) : Neighbourhood cover

Algorithm :

$L := \emptyset$, for all $a \in G$ if $\left(N_r^G(a) \models q_1(a)\right)$ then add a in L

Total time $O(n^2)$

Sketch of proof (2/4) : Neighbourhood cover

Algorithm :

$L := \emptyset$, for all $a \in G$ if $\left(N_r^G(a) \models q_1(a)\right)$ then add a in L

Total time $O(n^2)$

A neighbourhood cover is a set of "representative" neighbourhood.

$T := U_1, \ldots, U_\omega$ with the following properties:

- $\forall a \in G, \exists U_\lambda \in T,\ N_r(a) \subseteq U_\lambda$
- $\forall U_\lambda \in T, \exists a \in G,\ U_\lambda \subseteq N_{2r}(a)$ *(The bags have bounded expansion !)*
- $\forall a \in G,\ |\{\lambda \leq \omega \mid a \in U_\lambda\}|$ is psendo-constant

Sketch of proof (2/4) : Neighbourhood cover

Algorithm :

$L := \emptyset$, for all $a \in G$ if $\Big(N_r^G(a) \models q_1(a)\Big)$ then add a in L

Total time $O(n^2)$

A neighbourhood cover is a set of "representative" neighbourhood.

$T := U_1, \ldots, U_\omega$ with the following properties:

- $\forall a \in G, \exists U_\lambda \in T,\ N_r(a) \subseteq U_\lambda$
- $\forall U_\lambda \in T, \exists a \in G,\ U_\lambda \subseteq N_{2r}(a)$ *(The bags have bounded expansion !)*
- $\forall a \in G,\ |\{\lambda \leq \omega \mid a \in U_\lambda\}|$ is psendo-constant

Algorithm :

$L := \emptyset$, for all $\lambda \leq \omega$ for all $a \in U_\lambda$ if $\Big(U_\lambda \models q_1(a)\Big)$ then add a in L

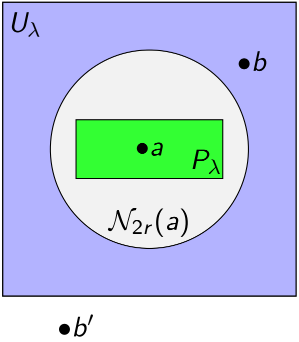
Total time pseudo linear.

Local queries can now be considered as unary predicate.

Only remains the distance !

Sketch of proof (3/4) : Using bounded expansion

$P_\lambda := \{a \in G \mid N_{2r}(a) \subseteq U_\lambda\}$



2 kind of solutions

close enough	far enough
exemple : (a,b)	*exemple : (a,b')*
$x \in P_\lambda \wedge y \in U_\lambda$	$x \in P_\lambda \wedge y \notin U_\lambda$
Studied locally (within U_λ)	No need to check the distance

Sketch of proof (4/4) : The short-cut pointers I

Let $a \in q_1(G)$, let λ such that $a \in P_\lambda$ we want to enumerate :

- $\{b \in q_2(G) \mid b \in U_\lambda\}$ Easy using the enumeration procedure within U_λ.
- $\{b \in q_2(G) \mid b \notin U_\lambda\}$ Here we need something else.

Sketch of proof (4/4) : The short-cut pointers I

Let $a \in q_1(G)$, let λ such that $a \in P_\lambda$ we want to enumerate :

- $\{b \in q_2(G) \mid b \in U_\lambda\}$ Easy using the enumeration procedure within U_λ.
- $\{b \in q_2(G) \mid b \notin U_\lambda\}$ Here we need something else.

$$\underset{b \in U_\lambda}{NEXT}(b, \lambda) := \min\{b' \in q_2(G) \mid b' \geq b \ \wedge \ b' \notin U_\lambda\}$$

Sketch of proof (4/4) : The short-cut pointers I

Let $a \in q_1(G)$, let λ such that $a \in P_\lambda$ we want to enumerate :

- $\{b \in q_2(G) \mid b \in U_\lambda\}$ Easy using the enumeration procedure within U_λ.
- $\{b \in q_2(G) \mid b \notin U_\lambda\}$ Here we need something else.

$$\underset{b \in U_\lambda}{NEXT}(b, \lambda) := \min\{b' \in q_2(G) \mid b' \geq b \ \wedge \ b' \notin U_\lambda\}$$

For all λ with $b_{max} \in U_\lambda$, we have $NEXT(b_{max}, \lambda) \ = \ NULL$

Sketch of proof (4/4) : The short-cut pointers I

Let $a \in q_1(G)$, let λ such that $a \in P_\lambda$ we want to enumerate :

- $\{b \in q_2(G) \mid b \in U_\lambda\}$ Easy using the enumeration procedure within U_λ.
- $\{b \in q_2(G) \mid b \notin U_\lambda\}$ Here we need something else.

$$\underset{b \in U_\lambda}{NEXT}(b, \lambda) := \min\{b' \in q_2(G) \mid b' \geq b \ \wedge \ b' \notin U_\lambda\}$$

For all λ with $b_{max} \in U_\lambda$, we have $NEXT(b_{max}, \lambda) \ = \ NULL$

$$NEXT(b_i, \lambda) \in \{b_{i+1} \ , \ NEXT_\lambda(b_{i+1})\}$$

Sketch of proof (4/4) : The short-cut pointers I

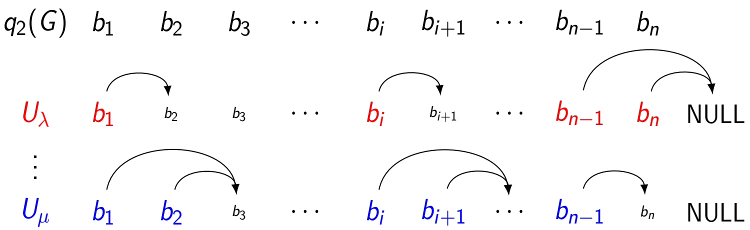
Let $a \in q_1(G)$, let λ such that $a \in P_\lambda$ we want to enumerate :

- $\{b \in q_2(G) \mid b \in U_\lambda\}$ Easy using the enumeration procedure within U_λ.
- $\{b \in q_2(G) \mid b \notin U_\lambda\}$ Here we need something else.

$$\underset{b \in U_\lambda}{NEXT}(b, \lambda) := \min\{b' \in q_2(G) \mid b' \geq b \ \wedge \ b' \notin U_\lambda\}$$

For all λ with $b_{max} \in U_\lambda$, we have $NEXT(b_{max}, \lambda) \ = \ NULL$

$$NEXT(b_i, \lambda) \in \{b_{i+1} \ , \ NEXT_\lambda(b_{i+1})\}$$



Sketch of proof (4/4) : The short-cut pointers II

We are given two nodes (a, b), with $a \in q_1(G)$ and $b \in q_2(G)$. We want to either confirm that (a, b) is a solution and output it, or compute the next solution.

Sketch of proof (4/4) : The short-cut pointers II

We are given two nodes (a, b), with $a \in q_1(G)$ and $b \in q_2(G)$. We want to either confirm that (a, b) is a solution and output it, or compute the next solution.

- First case :$a \in P_\lambda,\ b \notin U_\lambda$ for some λ.
  Therefore $G \models q(a, b)$ and we can output it.

Sketch of proof (4/4) : The short-cut pointers II

We are given two nodes (a, b), with $a \in q_1(G)$ and $b \in q_2(G)$. We want to either confirm that (a, b) is a solution and output it, or compute the next solution.

- First case : $a \in P_\lambda,\ b \notin U_\lambda$ for some λ.
  Therefore $G \models q(a, b)$ and we can output it.

- Second case : $a \in P_\lambda,\ b \in U_\lambda$ for some λ.
  - Let $b_1 = \min\{y \in U_\lambda \mid \mathrm{dist}(a, y) > 2r \wedge y \geq b \wedge y \in q_2(G)\}$.
    From Segoufin-Kazana algorithm, we can compute b_1, that is the smallest solution within U_λ.
  - Let $b_2 = NEXT(b, \lambda)$, that is the next solution outside of U_λ.

Let $b' = \min(b_1, b_2)$. We can output (a, b') that is the next solution.

Outline

Future work

- Generalize the Nowhere-Dense case.

- Enumeration with update.
  What happens if a small change occurs after the preprocessing ?

Thank you !

Any Question ?