Counting Problems over Incomplete Databases

Mikaël Monet

Formal Methods team seminar at LaBRI

Setpember 29th, 2020

About me

[2012–2015] Engineering school in Nancy

[2015–2018] PhD in Paris (*Télécom ParisTech*) with **Pierre Senellart** and **Antoine Amarilli**

→ Database theory, uncertain data management

[2019–August 2020] Postdoctorate in Santiago de Chile (*IMFD*) with **Pablo Barceló**

→ Database theory, uncertain data management, logical aspects of machine learning, complexity of explainability tasks (AI)

[September] Off

[1st October] Research position at Inria Lille, team LINKS

Uncertain data management

- Traditional database research assumes that the data is reliable, complete, clean...
- But real life data is often uncertain, untrustworthy, missing, inconsistent, etc.

Uncertain data management

- Traditional database research assumes that the data is reliable, complete, clean...
- But real life data is often uncertain, untrustworthy, missing, inconsistent, etc.
  - → imperfect sensor precision, error-prone automatic information extraction processes, data integration from multiple sources, missing information
- We could simply clean the data and remove every uncertain data item

Uncertain data management

- Traditional database research assumes that the data is reliable, complete, clean...
- But real life data is often uncertain, untrustworthy, missing, inconsistent, etc.
  - → imperfect sensor precision, error-prone automatic information extraction processes, data integration from multiple sources, missing information
- We could simply clean the data and remove every uncertain data item
- But what if we **actually need/want to acknowledge this uncertainty**? (e.g, if querying the data without taking the uncertainty into account could lead to incorrect answers)

Uncertain data management

- Traditional database research assumes that the data is reliable, complete, clean...
- But real life data is often uncertain, untrustworthy, missing, inconsistent, etc.
  - → imperfect sensor precision, error-prone automatic information extraction processes, data integration from multiple sources, missing information
- We could simply clean the data and remove every uncertain data item
- But what if we **actually need/want to acknowledge this uncertainty**? (e.g, if querying the data without taking the uncertainty into account could lead to incorrect answers)

→ Need to develop theories, tools, etc. to be able to **represent** and **query** such uncertain data

- → This is uncertain data management!

Frameworks for uncertain data management

Lots of existing frameworks to represent and query uncertain data:

- Bayesian networks
- Markov random fields
- Graphical models
- Possibility theory, fuzzy logic, etc.

In this talk, focus on frameworks for relational databases:

- **Probabilistic databases**
- **Incomplete databases**

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

$D =$

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

$D' =$

Likes		π
		0.5
Alice	John	1
		0.2
John	Bob	0.7

Probabilistic databases: example

- **Probabilistic databases**: to quantitatively represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

$D' =$

Likes		π
		0.5
Alice	John	1
		0.2
John	Bob	0.7

$\Pr(D') = (1 - 0.5) \times 1 \times (1 - 0.2) \times 0.7$

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty
  - $\rightarrow$ simplest formalism: tuple-independent database

D =

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

q = "there are two people who like the same person"

Probabilistic databases: example

- **Probabilistic databases**: to quantitatively represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

$D =$

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

q = "there are two people who like the same person"

$\exists x, y, z : L(x,z) \land L(y,z) \land x \neq y$

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

D =

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

q = "there are two people who like the same person"
$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

$\Pr((D, \pi) \models q) = \sum_{\substack{D' \subseteq D \\ D \models q}} \Pr(D')$

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

$D =$

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

q = "there are two people who like the same person"
$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

$\Pr((D, \pi) \models q) = \sum_{\substack{D' \subseteq D \\ D \models q}} \Pr(D')$ (**not efficient**)

Probabilistic databases: example

- **Probabilistic databases**: to quantitatively represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

D =

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

q = "there are two people who like the same person"

$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

$$\Pr((D, \pi) \models q) = 0.5 \times [1 - (1 - 0.2)(1 - 0.7)]$$

Probabilistic databases: example

- **Probabilistic databases**: to **quantitatively** represent and reason about data uncertainty
  - → simplest formalism: tuple-independent database

$D =$

Likes		π
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

q = "there are two people who like the same person"
$\exists x, y, z : L(x,z) \wedge L(y,z) \wedge x \neq y$

$$\Pr((D,\pi) \models q) = 0.5 \times [1 - (1-0.2)(1-0.7)]$$
$$+ (1-0.5) \times [0.2 \times 0.7]$$

Incomplete databases: example

- Probabilistic databases: nice, but this is not what is used in practice most of the time...

ProductId	ProductName	Price	Color	Localisation
439	Printer	\$100	NULL	Paris center
782	Mouse	\$10	red	NULL
398	Mouse	\$30	red	Miami center
...	...	...	...	...

CustomerId	Name	Phone number	Gender	Address
6	Bob	NULL	male	36 main street
76	Mary	551780726	NULL	NULL
...	...	...	...	...

Incomplete databases: example

- Probabilistic databases: nice, but this is not what is used in practice most of the time...

ProductId	ProductName	Price	Color	Localisation
439	Printer	\$100	NULL	Paris center
782	Mouse	\$10	red	NULL
398	Mouse	\$30	red	Miami center
...	...	...	...	...

CustomerId	Name	Phone number	Gender	Address
6	Bob	NULL	male	36 main street
76	Mary	551780726	NULL	NULL
...	...	...	...	...

→ Incomplete databases: relational databases with missing values

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

Example (from now on, nulls are *named* and represented with $\perp$):

D =

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

Example (from now on, nulls are *named* and represented with $\perp$):

$D =$

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

$\nu : \perp_1 \mapsto c, \perp_2 \mapsto a$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

Example (from now on, nulls are *named* and represented with $\perp$):

$\nu(D) =$

R	
a	b
b	c

S	
c	b
b	a

$\nu : \perp_1 \mapsto c, \perp_2 \mapsto \mathsf{a}$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

Example (from now on, nulls are *named* and represented with $\perp$):

D =

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database
  - → a tuple $\bar{a}$ is a certain answer of $q(\bar{x})$ over the incomplete database D if for every valuation ν of the nulls of D, we have $\bar{a} \in q(\nu(D))$

Example (from now on, nulls are *named* and represented with $\perp$):

$D =$

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database
  - → a tuple $\bar{a}$ is a certain answer of $q(\bar{x})$ over the incomplete database D if for every valuation ν of the nulls of D, we have $\bar{a} \in q(\nu(D))$

Example (from now on, nulls are *named* and represented with $\perp$):

D =

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

$$q(x) = \exists y, z : R(x, y) \land S(y, z)$$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database
  - → a tuple $\bar{a}$ is a certain answer of $q(\bar{x})$ over the incomplete database D if for every valuation ν of the nulls of D, we have $\bar{a} \in q(\nu(D))$

Example (from now on, nulls are *named* and represented with $\perp$):

D =

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

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

Certain answers: (a) and (b)

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

  → a tuple $\bar{a}$ is a certain answer of $q(\bar{x})$ over the incomplete database D if for every valuation ν of the nulls of D, we have $\bar{a} \in q(\nu(D))$

Example (from now on, nulls are *named* and represented with $\perp$):

$D =$

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

$q'(x) = R(x, x)$

How do we query incomplete databases?

- **Default approach** of database theorists for querying incomplete data: certain answers
  - for a valuation ν of the nulls of D into constants, let us write $\nu(D)$ the corresponding complete database

  → a tuple $\bar{a}$ is a certain answer of $q(\bar{x})$ over the incomplete database D if for every valuation ν of the nulls of D, we have $\bar{a} \in q(\nu(D))$

Example (from now on, nulls are *named* and represented with $\perp$):

$D =$

R	
a	b
b	$\perp_1$

S	
$\perp_1$	b
b	$\perp_2$

$q'(x) = R(x, x)$

No certain answer :(

Problem: what if there are no certain answers?

Problem: what if there are no certain answers?

→ We could return possible answers... Not very informative

Problem: what if there are no certain answers?

→ We could return possible answers... Not very informative

→ Recently, Libkin [PODS'18] proposes the notion of better answers

- a tuple $\bar{a}$ is a *better answer* than another tuple $\bar{b}$ if $\{\nu \mid \bar{b} \in q(D)\} \subseteq \{\nu \mid \bar{a} \in q(D)\}$

Problem: what if there are no certain answers?

→ We could return possible answers... Not very informative

→ Recently, Libkin [PODS'18] proposes the notion of better answers

- a tuple $\bar{a}$ is a *better answer* than another tuple $\bar{b}$ if $\{\nu \mid \bar{b} \in q(D)\} \subseteq \{\nu \mid \bar{a} \in q(D)\}$
  - → induces a notion of **best answer**
  - → also, **we can compare (some) tuples**

Another approach: counting

To compare all the tuples, why not study the associated counting problems?

Another approach: counting

To compare all the tuples, why not study the associated counting problems?

- → "How many valuations ν are such that $\bar{a} \in q(\nu(D))$?"
- → "How many distinct databases of the form $\nu(D)$ are such that $\bar{a} \in q(\nu(D))$?"

Another approach: counting

To compare all the tuples, why not study the associated counting problems?

- → "How many valuations ν are such that $\bar{a} \in q(\nu(D))$?"
- → "How many distinct databases of the form $\nu(D)$ are such that $\bar{a} \in q(\nu(D))$?"
  - → **we can compare *all* tuples**
  - → we can answer queries quantitatively (similar to probabilistic databases)

Another approach: counting

To compare all the tuples, why not study the associated counting problems?

→ "How many valuations ν are such that $\bar{a} \in q(\nu(D))$?"

→ "How many distinct databases of the form $\nu(D)$ are such that $\bar{a} \in q(\nu(D))$?"

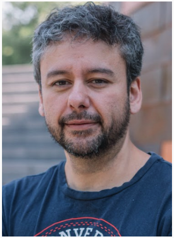
  → **we can compare *all* tuples**

  → we can answer queries quantitatively (similar to probabilistic databases)

→ **This is what we'll do in this talk!**

My co-authors

Rest of the talk is based on paper "Counting Problems over Incomplete Databases" [PODS'20] with Marcelo Arenas and Pablo Barceló



Setting

- Incomplete databases with named (marked) nulls
- Each null $\perp$ comes with its own finite domain $\mathrm{dom}(\perp)$; all valuations ν are such that $\nu(\perp) \in \mathrm{dom}(\perp)$
- $\nu(D)$: the (complete) database obtained from D by substituting every null $\perp$ by $\nu(\perp)$, *and then removing duplicate tuples*. We call such a database a completion of D

Setting

- Incomplete databases with named (marked) nulls
- Each null $\perp$ comes with its own finite domain $\mathrm{dom}(\perp)$; all valuations ν are such that $\nu(\perp) \in \mathrm{dom}(\perp)$
- $\nu(D)$: the (complete) database obtained from D by substituting every null $\perp$ by $\nu(\perp)$, *and then removing duplicate tuples*. We call such a database a completion of D

$D =$

R	
$\perp_1$	$\perp_1$
a	$\perp_2$

$\mathrm{dom}(\perp_1) = \{a, b\}$, $\mathrm{dom}(\perp_2) = \{b, c\}$

Setting

- Incomplete databases with named (marked) nulls
- Each null $\perp$ comes with its own finite domain $\mathrm{dom}(\perp)$; all valuations ν are such that $\nu(\perp) \in \mathrm{dom}(\perp)$
- $\nu(D)$: the (complete) database obtained from D by substituting every null $\perp$ by $\nu(\perp)$, *and then removing duplicate tuples*. We call such a database a completion of D

$D =$

R	
$\perp_1$	$\perp_1$
a	$\perp_2$

$\mathrm{dom}(\perp_1) = \{a, b\}$, $\mathrm{dom}(\perp_2) = \{b, c\}$

$$\nu = \{\perp_1 \mapsto b, \perp_2 \mapsto c\} \quad \to \quad \nu(D) = \{R(b,b), R(a,c)\}$$

Setting

- Incomplete databases with named (marked) nulls
- Each null $\perp$ comes with its own finite domain $\mathrm{dom}(\perp)$; all valuations ν are such that $\nu(\perp) \in \mathrm{dom}(\perp)$
- $\nu(D)$: the (complete) database obtained from D by substituting every null $\perp$ by $\nu(\perp)$, *and then removing duplicate tuples*. We call such a database a completion of D

$D =$

R	
$\perp_1$	$\perp_1$
a	$\perp_2$

$\mathrm{dom}(\perp_1) = \{a, b\}$, $\mathrm{dom}(\perp_2) = \{b, c\}$

$$\nu = \{\perp_1 \mapsto b, \perp_2 \mapsto c\} \quad \rightarrow \quad \nu(D) = \{R(b, b), R(a, c)\}$$

$$\nu = \{\perp_1 \mapsto a, \perp_2 \mapsto a\} \quad \rightarrow \quad \nu(D) = \{R(a, a)\}$$

Problems studied

- Fix a Boolean query q

Definition: problem $\#\mathsf{Val}(q)$

Input: an incomplete database D, together with *finite* domains $\mathrm{dom}(\perp)$ for each null of D

Output: the number of valuations ν such that $\nu(D) \models q$

Problems studied

- Fix a Boolean query q

Definition: problem $\#\mathsf{Val}(q)$

Input: an incomplete database D, together with *finite* domains $\mathrm{dom}(\perp)$ for each null of D

Output: the number of valuations ν such that $\nu(D) \models q$

Definition: problem $\#\mathsf{Comp}(q)$

Input: an incomplete database D, together with *finite* domains $\mathrm{dom}(\perp)$ for each null of D

Output: the number of *completions* $\nu(D)$ such that $\nu(D) \models q$

Example

- **Example:** $D = \{S(a,b), S(\perp_1, a), S(a, \perp_2)\}$,
  $\mathrm{dom}(\perp_1) = \{a,b,c\}, \mathrm{dom}(\perp_2) = \{a,b\}$, $q = \exists x\, S(x,x)$

Example

- **Example:** $D = \{S(a,b), S(\bot_1, a), S(a, \bot_2)\}$, $\mathrm{dom}(\bot_1) = \{a,b,c\}, \mathrm{dom}(\bot_2) = \{a,b\}$, $q = \exists x\, S(x,x)$

$(\nu(\bot_1), \nu(\bot_2))$	(a,a)	(a,b)	(b,a)	(b,b)	(c,a)	(c,b)
$\nu(D)$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S} \\ \hline a & b \\ a & a \\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S} \\ \hline a & b \\ a & a \\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S} \\ \hline a & b \\ b & a \\ a & a \\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S} \\ \hline a & b \\ b & a \\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S} \\ \hline a & b \\ c & a \\ a & a \\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S} \\ \hline a & b \\ c & a \\ \hline\end{array}$
$\nu(D) \vDash Q$?	Yes	Yes	Yes	No	Yes	No

Example

- **Example:** $D = \{S(a,b), S(\perp_1, a), S(a, \perp_2)\}$, $\mathrm{dom}(\perp_1) = \{a,b,c\}, \mathrm{dom}(\perp_2) = \{a,b\}$, $q = \exists x\, S(x,x)$

$(\nu(\perp_1), \nu(\perp_2))$	(a,a)	(a,b)	(b,a)	(b,b)	(c,a)	(c,b)
$\nu(D)$	S: (a, b), (a, a)	S: (a, b), (a, a)	S: (a, b), (b, a), (a, a)	S: (a, b), (b, a)	S: (a, b), (c, a), (a, a)	S: (a, b), (c, a)
$\nu(D) \models Q$?	Yes	Yes	Yes	No	Yes	No

4 satisfying valuations, 3 satisfying completions

Example

- **Example:** $D = \{S(a,b), S(\perp_1, a), S(a, \perp_2)\}$,
  $\mathrm{dom}(\perp_1) = \{a,b,c\}, \mathrm{dom}(\perp_2) = \{a,b\}$, $q = \exists x\, S(x,x)$

$(\nu(\perp_1), \nu(\perp_2))$	(a,a)	(a,b)	(b,a)	(b,b)	(c,a)	(c,b)
$\nu(D)$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S}\\ \hline a & b\\ a & a\\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S}\\ \hline a & b\\ a & a\\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S}\\ \hline a & b\\ b & a\\ a & a\\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S}\\ \hline a & b\\ b & a\\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S}\\ \hline a & b\\ c & a\\ a & a\\ \hline\end{array}$	$\begin{array}{cc}\hline \multicolumn{2}{c}{S}\\ \hline a & b\\ c & a\\ \hline\end{array}$
$\nu(D) \models Q$?	Yes	Yes	Yes	No	Yes	No

4 satisfying valuations, 3 satisfying completions

$\rightarrow$ Study the complexity of these problems depending on q (*data complexity*). Obtain **dichotomies**? Can we efficiently **approximate** the number of solutions? Etc.

Problems variants and query language

We also study the settings where:

- all labeled nulls are distinct (***Codd tables***; by contrast to ***naïve tables***)
- all nulls share the same domain (***uniform setting***)

→ In total we consider 8 **different settings** $(\{\#\mathrm{Val}, \#\mathrm{Comp}\} \times \{\text{naïve/Codd}\} \times \{\text{non-uniform/uniform}\})$

Problems variants and query language

We also study the settings where:

- all labeled nulls are distinct (***Codd tables***; by contrast to ***naïve tables***)
- all nulls share the same domain (***uniform setting***)

→ In total we consider 8 **different settings** ($\{\#$Val$, \#$Comp$\} \times \{$naïve/Codd$\} \times \{$non-uniform/uniform$\}$)

- We focus only on **self-join free Boolean conjunctive queries** ($\mathrm{sjfBCQs}$)

Outline

The dichotomies for exact counting

Counting valuations vs. counting completions

Approximations

The dichotomies for exact counting

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)
$q' = R'(u, u, y) \land S(z)$ is a pattern
of $q = R(u, x, u) \land S(y, y) \land T(x, s, z, s)$

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)
$q' = R'(u,u,y) \land S(z)$ is a pattern
of $q = R(u,x,u) \land S(y,y) \land T(x,s,z,s)$

$\rightarrow R(u,x,u) \land S(y,y)$ (delete third atom)

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)

$q' = R'(u, u, y) \wedge S(z)$ is a pattern

of $q = R(u, x, u) \wedge S(y, y) \wedge T(x, s, z, s)$

$\rightarrow R(u, x, u) \wedge S(y, y)$ (delete third atom)

$\rightarrow R(u, x, u) \wedge S(y)$ (delete a variable occurrence)

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)
$q' = R'(u,u,y) \wedge S(z)$ is a pattern
of $q = R(u,x,u) \wedge S(y,y) \wedge T(x,s,z,s)$

- $\rightarrow R(u,x,u) \wedge S(y,y)$ (delete third atom)
- $\rightarrow R(u,x,u) \wedge S(y)$ (delete a variable occurrence)
- $\rightarrow R(u,u,x) \wedge S(y)$ (reorder variables occurrences)

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)
$q' = R'(u,u,y) \wedge S(z)$ is a pattern
of $q = R(u,x,u) \wedge S(y,y) \wedge T(x,s,z,s)$

- $\rightarrow$ $R(u,x,u) \wedge S(y,y)$ (delete third atom)
- $\rightarrow$ $R(u,x,u) \wedge S(y)$ (delete a variable occurrence)
- $\rightarrow$ $R(u,u,x) \wedge S(y)$ (reorder variables occurrences)
- $\rightarrow$ $R'(u,u,x) \wedge S(y)$ (rename R into R')

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)
$q' = R'(u, u, y) \wedge S(z)$ is a pattern
of $q = R(u, x, u) \wedge S(y, y) \wedge T(x, s, z, s)$

- $\rightarrow R(u, x, u) \wedge S(y, y)$ (delete third atom)
- $\rightarrow R(u, x, u) \wedge S(y)$ (delete a variable occurrence)
- $\rightarrow R(u, u, x) \wedge S(y)$ (reorder variables occurrences)
- $\rightarrow R'(u, u, x) \wedge S(y)$ (rename R into R')
- $\rightarrow R'(u, u, y) \wedge S(z)$ (rename x into y and y into z)

Patterns in sjfBCQs

Definition: pattern

A sjfBCQ q' is a pattern of another sjfBCQ q if q' can be obtained from q by deleting atoms or variable occurrences, and then reordering the variables inside the atoms and renaming (injectively) the variables and relation names

Example: (from now on all variables are existentially quantified)
$q' = R'(u,u,y) \wedge S(z)$ is a pattern
of $q = R(u,x,u) \wedge S(y,y) \wedge T(x,s,z,s)$

$\rightarrow R(u,x,u) \wedge S(y,y)$	(delete third atom)
$\rightarrow R(u,x,u) \wedge S(y)$	(delete a variable occurrence)
$\rightarrow R(u,u,x) \wedge S(y)$	(reorder variables occurrences)
$\rightarrow R'(u,u,x) \wedge S(y)$	(rename R into R')
$\rightarrow R'(u,u,y) \wedge S(z)$	(rename x into y and y into z)

Note: reordering and injective renaming are not important, it is just so that we can formally say things like:

- $R(x, y)$ is a pattern of $R(y, x)$; or
- $R(x)$ is a pattern of $S(y)$
- etc.

Proof strategy

Lemma

Let q, q' be $\mathrm{sjfBCQs}$ such that q' is a pattern of q. Then we have $\#\mathsf{Val}(q') \leq^{\mathrm{p}} \#\mathsf{Val}(q)$

Where $\leq^{\mathrm{p}}$ denote polynomial-time parsimonious reductions (and the same results holds for counting completions, and also if we restrict to Codd tables and/or to the uniform setting)

Proof strategy

Lemma

Let q, q' be $\mathrm{sjfBCQs}$ such that q' is a pattern of q. Then we have $\#\mathsf{Val}(q') \leq^{\mathrm{p}} \#\mathsf{Val}(q)$

Where $\leq^{\mathrm{p}}$ denote polynomial-time parsimonious reductions (and the same results holds for counting completions, and also if we restrict to Codd tables and/or to the uniform setting)

→ for each of the 8 variants of the problem, find a set of patterns that are hard and such that if a sjfBCQ does not have any of these patterns then the problem is in PTIME

Example 1: $\#$Val, naïve, non-uniform

Consider counting valuations, naïve setting (named nulls that can appear in multiple places), non-uniform (each null $\perp$ comes with its own domain $\mathrm{dom}(\perp)$)

- $q_1 = R(x, x)$ is a hard pattern: easy reduction from counting 3-colorings of a graph ($\#\mathrm{P}$-complete)

Example 1: #Val, naïve, non-uniform

Consider counting valuations, naïve setting (named nulls that can appear in multiple places), non-uniform (each null $\perp$ comes with its own domain $\mathrm{dom}(\perp)$)

- $q_1 = R(x, x)$ is a hard pattern: easy reduction from counting 3-colorings of a graph (#P-complete)
  - → on input undirected graph $G = (V, E)$, construct database D_G containing facts $R(\perp_u, \perp_v)$ and $R(\perp_v, \perp_u)$ for every edge $\{u, v\} \in E$. The domain of every null $\perp$ is $\mathrm{dom}(\perp) = \{\bullet, \bullet, \bullet\}$.

Example 1: #Val, naïve, non-uniform

Consider counting valuations, naïve setting (named nulls that can appear in multiple places), non-uniform (each null $\bot$ comes with its own domain $\mathrm{dom}(\bot)$)

- $q_1 = R(x, x)$ is a hard pattern: easy reduction from counting 3-colorings of a graph (#P-complete)
  - → on input undirected graph $G = (V, E)$, construct database D_G containing facts $R(\bot_u, \bot_v)$ and $R(\bot_v, \bot_u)$ for every edge $\{u, v\} \in E$. The domain of every null $\bot$ is $\mathrm{dom}(\bot) = \{\bullet, \bullet, \bullet\}$. Then $\#3\mathrm{Cols}(G) = 3^{|V|} - \#\mathrm{Val}(q_1)(D_G)$

Example 1: #Val, naïve, non-uniform

Consider counting valuations, naïve setting (named nulls that can appear in multiple places), non-uniform (each null $\perp$ comes with its own domain $\mathrm{dom}(\perp)$)

- $q_1 = R(x, x)$ is a hard pattern: easy reduction from counting 3-colorings of a graph ($\#\mathrm{P}$-complete)
  - → on input undirected graph $G = (V, E)$, construct database D_G containing facts $R(\perp_u, \perp_v)$ and $R(\perp_v, \perp_u)$ for every edge $\{u, v\} \in E$. The domain of every null $\perp$ is $\mathrm{dom}(\perp) = \{\bullet, \bullet, \bullet\}$. Then $\#\mathrm{3Cols}(G) = 3^{|V|} - \#\mathrm{Val}(q_1)(D_G)$
- $q_2 = R(x) \wedge S(x)$ is also a hard pattern (trust me)

Example 1: #Val, naïve, non-uniform

Consider counting valuations, naïve setting (named nulls that can appear in multiple places), non-uniform (each null $\bot$ comes with its own domain $\mathrm{dom}(\bot)$)

- $q_1 = R(x, x)$ is a hard pattern: easy reduction from counting 3-colorings of a graph (#P-complete)
  - → on input undirected graph $G = (V, E)$, construct database D_G containing facts $R(\bot_u, \bot_v)$ and $R(\bot_v, \bot_u)$ for every edge $\{u, v\} \in E$. The domain of every null $\bot$ is $\mathrm{dom}(\bot) = \{\bullet, \bullet, \bullet\}$. Then $\#\mathrm{3Cols}(G) = 3^{|V|} - \#\mathrm{Val}(q_1)(D_G)$
- $q_2 = R(x) \wedge S(x)$ is also a hard pattern (trust me)
- If a sjfBCQ q does not have q_1 or q_2 as a pattern then $\#\mathrm{Val}(q)$ is PTIME. Why?

Example 1: #Val, naïve, non-uniform

Consider counting valuations, naïve setting (named nulls that can appear in multiple places), non-uniform (each null $\perp$ comes with its own domain $\mathrm{dom}(\perp)$)

- $q_1 = R(x, x)$ is a hard pattern: easy reduction from counting 3-colorings of a graph ($\#\mathrm{P}$-complete)
  - → on input undirected graph $G = (V, E)$, construct database D_G containing facts $R(\perp_u, \perp_v)$ and $R(\perp_v, \perp_u)$ for every edge $\{u, v\} \in E$. The domain of every null $\perp$ is $\mathrm{dom}(\perp) = \{{\color{red}\bullet}, {\color{blue}\bullet}, {\color{green}\bullet}\}$. Then $\#\mathrm{3Cols}(G) = 3^{|V|} - \#\mathrm{Val}(q_1)(D_G)$
- $q_2 = R(x) \wedge S(x)$ is also a hard pattern (trust me)
- If a sjfBCQ q does not have q_1 or q_2 as a pattern then $\#\mathrm{Val}(q)$ is PTIME. Why?
  - → All variable occurrences are distinct, so every valuation is satisfying

Example 2: completions, naïve, Codd

Now consider counting completions for Codd databases (all nulls are distinct), non-uniform

Example 2: completions, naïve, Codd

Now consider counting completions for Codd databases (all nulls are distinct), non-uniform

- $q = R(x)$ is a hard pattern! Reduction from counting the number of vertex covers of a graph

Example 2: completions, naïve, Codd

Now consider counting completions for Codd databases (all nulls are distinct), non-uniform

- $q = R(x)$ is a hard pattern! Reduction from counting the number of vertex covers of a graph
  - $\rightarrow$ on input graph $G = (V, E)$, construct database D_G having:
    - one null $\perp_e$ and fact $R(\perp_e)$ for every edge $e = \{u, v\}$ of G with domain $\mathrm{dom}(\perp_e) = \{u, v\}$

Example 2: completions, naïve, Codd

Now consider counting completions for Codd databases (all nulls are distinct), non-uniform

- $q = R(x)$ is a hard pattern! Reduction from counting the number of vertex covers of a graph
  - $\rightarrow$ on input graph $G = (V, E)$, construct database D_G having:
    - one null $\perp_e$ and fact $R(\perp_e)$ for every edge $e = \{u, v\}$ of G with domain $\mathrm{dom}(\perp_e) = \{u, v\}$
    - one fact $R(\bullet)$ where "$\bullet$" is a special symbol
    - one null $\perp_u$ and fact $R(\perp_u)$ for every node u of G with domain $\mathrm{dom}(\perp_u) = \{u, \bullet\}$

Example 2: completions, naïve, Codd

Now consider counting completions for Codd databases (all nulls are distinct), non-uniform

- $q = R(x)$ is a hard pattern! Reduction from counting the number of vertex covers of a graph
  - → on input graph $G = (V, E)$, construct database D_G having:
    - one null $\bot_e$ and fact $R(\bot_e)$ for every edge $e = \{u, v\}$ of G with domain $\mathrm{dom}(\bot_e) = \{u, v\}$
    - one fact $R(\bullet)$ where "$\bullet$" is a special symbol
    - one null $\bot_u$ and fact $R(\bot_u)$ for every node u of G with domain $\mathrm{dom}(\bot_u) = \{u, \bullet\}$
  - → We have that $\#\mathrm{VC}(G) = \#\mathrm{Comp}(q)(D_G)$

Example 2: completions, naïve, Codd

Now consider counting completions for Codd databases (all nulls are distinct), non-uniform

- $q = R(x)$ is a hard pattern! Reduction from counting the number of vertex covers of a graph
  - → on input graph $G = (V, E)$, construct database D_G having:
    - one null $\perp_e$ and fact $R(\perp_e)$ for every edge $e = \{u, v\}$ of G with domain $\mathrm{dom}(\perp_e) = \{u, v\}$
    - one fact $R(\bullet)$ where "$\bullet$" is a special symbol
    - one null $\perp_u$ and fact $R(\perp_u)$ for every node u of G with domain $\mathrm{dom}(\perp_u) = \{u, \bullet\}$
  - → We have that $\#\mathrm{VC}(G) = \#\mathrm{Comp}(q)(D_G)$
- In other words, here every sjfBCQ is hard...

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	Uniform	Non-uniform	Uniform
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	Uniform	Non-uniform	Uniform
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

→ Valuations, non-uniform, Codd: each variable occurs in at most one atom

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	Uniform	Non-uniform	Uniform
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

→ Valuations, non-uniform, Codd: each variable occurs in at most one atom

→ Completions, uniform (naïve or Codd): all the atoms are unary

The hard patterns

	Counting valuations		Counting completions	
	Non-uniform	Uniform	Non-uniform	Uniform
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

→ Valuations, non-uniform, Codd: each variable occurs in at most one atom

→ Completions, uniform (naïve or Codd): all the atoms are unary

(So. . . not much is tractable)

Counting valuations vs. counting completions

When are our problems in $\#\mathrm{P}$?

- For a Boolean query q, let $\mathrm{MC}(q)$ denote the model checking problem for q

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Val}(q)$ is in $\#\mathrm{P}$.

When are our problems in $\#\mathrm{P}$?

- For a Boolean query q, let $\mathrm{MC}(q)$ denote the model checking problem for q

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Val}(q)$ is in $\#\mathrm{P}$.

- for counting valuations of sjfBCQs, we had dichotomies between PTIME and $\#\mathrm{P}$-completeness

What about counting completions? In general when $\mathrm{MC}(q)$ is PTIME, is $\#\mathsf{Comp}(q)$ in $\#\mathrm{P}$?

When are our problems in $\#\mathrm{P}$?

- For a Boolean query q, let $\mathrm{MC}(q)$ denote the model checking problem for q

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Val}(q)$ is in $\#\mathrm{P}$.

- for counting valuations of sjfBCQs, we had dichotomies between PTIME and $\#\mathrm{P}$-completeness

What about counting completions? In general when $\mathrm{MC}(q)$ is PTIME, is $\#\mathsf{Comp}(q)$ in $\#\mathrm{P}$? **Unlikely:**

Proposition

There exists an sjfBCQ q such that $\#\mathsf{Comp}(q)$ is not in $\#\mathrm{P}$ unless $\mathrm{NP} \subseteq \mathrm{SPP}$

A natural complexity class for counting completions (1/2)

- A counting problem A is in SpanP if there exists a nondeterministic transducer M (= Turing machine with output tape) running in polynomial time such that, on input x, the **number of distinct outputs** for $M(x)$ is equal to $A(x)$

A natural complexity class for counting completions (1/2)

- A counting problem A is in SpanP if there exists a nondeterministic transducer M (= Turing machine with output tape) running in polynomial time such that, on input x, the **number of distinct outputs** for $M(x)$ is equal to $A(x)$
  - → Clearly $\#\mathrm{P} \subseteq \mathrm{SpanP}$, but we have $\#\mathrm{P} = \mathrm{SpanP}$ if and only if $\mathrm{NP} = \mathrm{UP}$ (Köbler et al. [Acta Informatica'89])

A natural complexity class for counting completions (1/2)

- A counting problem A is in SpanP if there exists a nondeterministic transducer M (= Turing machine with output tape) running in polynomial time such that, on input x, the **number of distinct outputs** for $M(x)$ is equal to $A(x)$
  - → Clearly $\#\mathrm{P} \subseteq \mathrm{SpanP}$, but we have $\#\mathrm{P} = \mathrm{SpanP}$ if and only if $\mathrm{NP} = \mathrm{UP}$ (Köbler et al. [Acta Informatica'89])
  - → A complete problem for SpanP: INPUT: a 3-CNF φ and integer k; OUTPUT: the number of assignments of the first k variables that can be extended to a satisfying assignment of φ

A natural complexity class for counting completions (1/2)

- A counting problem A is in SpanP if there exists a nondeterministic transducer M (= Turing machine with output tape) running in polynomial time such that, on input x, the **number of distinct outputs** for $M(x)$ is equal to $A(x)$
  - → Clearly $\#\mathrm{P} \subseteq \mathrm{SpanP}$, but we have $\#\mathrm{P} = \mathrm{SpanP}$ if and only if $\mathrm{NP} = \mathrm{UP}$ (Köbler et al. [Acta Informatica'89])
  - → A complete problem for SpanP: INPUT: a 3-CNF φ and integer k; OUTPUT: the number of assignments of the first k variables that can be extended to a satisfying assignment of φ
  - → (A problem in SpanP but unknown to be complete for it: INPUT: a graph G; OUTPUT: the number of Hamiltonian subgraphs of G)

A natural complexity class for counting completions (2/2)

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Comp}(q)$ is in SpanP.

A natural complexity class for counting completions (2/2)

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Comp}(q)$ is in SpanP.

Proposition

There exists a sjfBCQ q such that $\#\mathsf{Comp}(\neg q)$ is SpanP-complete.

A natural complexity class for counting completions (2/2)

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Comp}(q)$ is in SpanP.

Proposition

There exists a sjfBCQ q such that $\#\mathsf{Comp}(\neg q)$ is SpanP-complete.

[WARNING: hardness for SpanP is defined in terms of parsimonious reductions (while $\#\mathrm{P}$-completeness is usually defined with Turing reductions)]

A natural complexity class for counting completions (2/2)

Fact

If $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Comp}(q)$ is in SpanP.

Proposition

There exists a sjfBCQ q such that $\#\mathsf{Comp}(\neg q)$ is SpanP-complete.

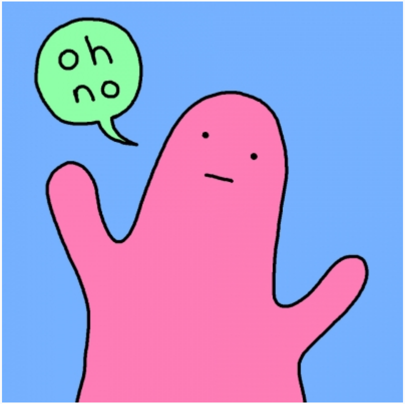
[WARNING: hardness for SpanP is defined in terms of parsimonious reductions (while $\#\mathrm{P}$-completeness is usually defined with Turing reductions)]

For Codd tables we can still show membership in $\#\mathrm{P}$:

Proposition

For **Codd tables**, if $\mathrm{MC}(q)$ is PTIME then $\#\mathsf{Comp}(q)$ is in $\#\mathrm{P}$

Approximations



My counting problem is very much intractable :(

→ Try Fully Polynomial-time Randomized Approximation Scheme!

Fully Polynomial-time Randomized Approximation Scheme!

Definition (FPRAS)

Let Σ be a finite alphabet and $f : \Sigma^* \to \mathbb{N}$ be a counting problem. Then f is said to have an FPRAS if there is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0,1) \to \mathbb{N}$ and a polynomial $p(u,v)$ such that, given $x \in \Sigma^*$ and $\epsilon \in (0,1)$, algorithm $\mathcal{A}$ runs in time $p(|x|, 1/\epsilon)$ and satisfies the following condition:

$$\Pr\left(|f(x) - \mathcal{A}(x,\epsilon)| \leq \epsilon f(x)\right) \geq \frac{3}{4}.$$

Fully Polynomial-time Randomized Approximation Scheme!

Definition (FPRAS)

Let Σ be a finite alphabet and $f : \Sigma^* \to \mathbb{N}$ be a counting problem. Then f is said to have an FPRAS if there is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0,1) \to \mathbb{N}$ and a polynomial $p(u, v)$ such that, given $x \in \Sigma^*$ and $\epsilon \in (0,1)$, algorithm $\mathcal{A}$ runs in time $p(|x|, 1/\epsilon)$ and satisfies the following condition:

$$\Pr\Big(|f(x) - \mathcal{A}(x,\epsilon)| \leq \epsilon f(x)\Big) \quad \geq \quad \frac{3}{4}.$$

Note: the property of having an FPRAS is closed under polynomial-time parsimonious reductions (i.e., if we have an FPRAS for a counting problem A and for counting problem B we have that $B \leq^{\mathrm{p}} A$, then we also have an FPRAS for B).

FPRAS for counting valuations

Proposition

For every Boolean UCQ q, the problem $\#\mathsf{Val}(q)$ has a FPRAS

Proof: via SpanL. SpanL = there exists an NL transducer with write-only output tape such that the result is the number of distinct outputs

FPRAS for counting valuations

Proposition

For every Boolean UCQ q, the problem $\#\text{Val}(q)$ has a FPRAS

Proof: via SpanL. SpanL = there exists an NL transducer with write-only output tape such that the result is the number of distinct outputs

Theorem (Arenas et al. [PODS'19])

Every problem in SpanL has an FPRAS

FPRAS for counting valuations

Proposition

For every Boolean UCQ q, the problem $\#\mathsf{Val}(q)$ has a FPRAS

Proof: via SpanL. SpanL = there exists an NL transducer with write-only output tape such that the result is the number of distinct outputs

Theorem (Arenas et al. [PODS'19])

Every problem in SpanL has an FPRAS

Fact

For every Boolean UCQ q, the problem $\#\mathsf{Val}(q)$ is in SpanL

FPRAS for counting completions?

Theorem (Dyer et al. [SICOMP'2002])

Counting vertex covers has no FPRAS unless $\mathrm{NP} = \mathrm{RP}$

- Our reduction from $\#\mathrm{VC}$ for Codd tables to $\#\mathsf{Comp}(\exists x\, R(x))$ was **parsimonious**
- Our reduction for the notion of pattern is also **parsimonious**

→ Therefore $\#\mathsf{Comp}(q)$ restricted to Codd tables for any sjfBCQ has no FPRAS unless $\mathrm{NP} = \mathrm{RP}$

FPRAS for counting completions?

Theorem (Dyer et al. [SICOMP'2002])

Counting vertex covers has no FPRAS unless $\mathrm{NP} = \mathrm{RP}$

- Our reduction from $\#\mathrm{VC}$ for Codd tables to $\#\mathrm{Comp}(\exists x\, R(x))$ was **parsimonious**
- Our reduction for the notion of pattern is also **parsimonious**

→ Therefore $\#\mathrm{Comp}(q)$ restricted to Codd tables for any sjfBCQ has no FPRAS unless $\mathrm{NP} = \mathrm{RP}$

What about the uniform setting? We prove that for naïve tables, uniform setting, $\#\mathrm{Comp}(q)$ has no FPRAS if q contains a non-unary symbol (otherwise it is PTIME)

- For uniform Codd tables, **we do not know**

Conclusion

To sum up:

- Counting valuations and completions is **hard**, even in very restricted settings (uniform Codd tables)
- But counting valuations has a **FPRAS** for UCQs
- While counting completions does not
- SpanP is the right class to consider for problems of the form $\#\mathrm{Comp}(q)$
- If you liked it, we have a lot of cute reductions in the paper :)

Thanks for your attention!

Marcelo Arenas, Pablo Barceló, and Mikaël Monet.
Counting Problems over Incomplete Databases.
In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 165–177, 2020.

Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros.
Efficient logspace classes for enumeration, counting, and uniform generation.
In *PODS*, pages 59–73, 2019.

Martin Dyer, Alan Frieze, and Mark Jerrum.
On counting independent sets in sparse graphs.
SIAM J. on Computing, 31(5):1527–1541, 2002.

Johannes Köbler, Uwe Schöning, and Jacobo Torán.
On counting and approximation.
Acta Informatica, 26(4):363–379, 1989.

Leonid Libkin.
Certain Answers Meet Zero-One Laws.
In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 195–207, 2018.