The complexity of enumeration and counting for acyclic conjunctive queries

Arnaud Durand

Cambridge, march 2012

Counting and enumeration

Counting

- Output the number of solutions
- Example : ♯SAT, ♯PERFECT MATCHING, PERMANENT, problems on graph, knots polynomials

Enumeration

- Generate all solutions one by one
- Output can be large
- Natural algorithmic task

This talk: look at these two tasks for fragments of conjunctive (i.e. $\{\exists, \wedge\}$) queries (mainly **ACQ**)

Outline

- Enumeration
  - A tour on complexity measure for enumeration
  - Results for query problems
  - Complexity characterization for **ACQ**
- Weighted Counting
  - Polynomial representation of query
  - Counting through polynomial evaluation
  - Characterization of the tractability frontier for **ACQ**

Common point : role of free variables in formulas.

Enumeration: measure of complexity

Intractability

- **NP**-completeness of the existence of one solution (hard to start)
- hard after generation of a part of the solution set (hard to continue)

Tractability

- Polynomial total time (succeed but don't know how)
- Incremental Polynomial Time (harder as the number of generated solutions increases)
- Polynomial delay (regular process)

Example

- Generate all models of a propositional formula (hum... intractable)
- Generate all independant sets of maximum size of a graph (intractable)
- Generate all maximal (for inclusion) independant sets
  - polynomial delay for lexicographic ordering
  - intractable for reverse lexicographic ordering
- Generate all models of a 2-CNF propositional formula (polynomial delay)

Complexity of enumeration of satisfiability problems

- Let φ be a formula with variables $x_1, \ldots, x_n$.
- If $\varphi \wedge \neg x_1 \in SAT$: enumerate all solutions of $\varphi \wedge \neg x_1$
- If $\varphi \wedge x_1 \in SAT$: enumerate all solutions of $\varphi \wedge x_1$

Theorem (Creignou, Hebrard'97)

In the Boolean case, there is no other efficient algorithm than the one described above. Consequently, easy (i.e. polynomial delay) cases are Horn, Anti-Horn, 2-CNF and affine.

Schnoor & Schnoor'07 The situation is more complex for CSP on arbitrary finite domain

See also Bulatov, Dalmau, Grohe & Marx'10

Enumeration algorithm: some precision

Algorithms in two steps: $+\begin{cases}\text{Precomputation step}\\ \text{Enumeration step(s)}\end{cases}$

- Computation models (in this talk) : *RAM* with uniform cost (but with bounded values in registers)
- Let $\mathcal{A}$ be an enumeration algorithm for $\textsc{Enum}(R)$ and $x \in I$.
  - $time_i(x)$ denotes the time when the algorithm finishes to write the ith solution if it exists.
  - $delay_i(x) = time_{i+1}(x) - time_i(x)$.

Delay may be : sub-exponential, polynomial, linear, constant (?)
Warning: space is an important factor!

Small classes of enumeration problems

- An enumeration algorithm $\mathcal{A}$ is *constant delay* if there is a constant c such that for any $x \in I$ and i, $delay_i(x) \leq c$.
- $\textsc{Enum}(R)$ is computable with constant delay ($\textsc{Enum}(R) \in \textsc{Constant-Delay}$) if there is a constant delay algorithm $\mathcal{A}$ which computes $\textsc{Enum}(R)$.
- $\textsc{Enum}(R) \in \textsc{Constant-Delay}_{lin}$ if it is reducible in linear time (in the input size) to a problem in $\textsc{Constant-Delay}$

precomputation $\equiv$ reduction

Query problems

Query

- $\mathcal{L}$ = set of formulas
- $\mathcal{S}$ = set of (finite) structures

Enumeration problem

$\textsc{Enum}(\mathcal{L}, \mathcal{S})$

input: $\phi \in \mathcal{L}$ and $\mathcal{A} \in \mathcal{S}$

output: enumerate $\phi(\mathcal{A}) = \{\bar{a} \in D^k \mid (\mathcal{A}, \bar{a}) \models \phi(\bar{x})\}$

Counting problem

$\textsc{Count}(\mathcal{L}, \mathcal{S})$

input: $\phi \in \mathcal{L}$ and $\mathcal{A} \in \mathcal{S}$

output: $|\phi(\mathcal{A})| = |\{\bar{a} \in D^k \mid (\mathcal{A}, \bar{a}) \models \phi(\bar{x})\}|$

Complexity Measures

Framework for complexity analysis of multi-parameterized problems

Parameters

$\mathcal{A}$, φ (inputs)
$\varphi(\mathcal{A})$
But also: number of free variables, of quantified variables, alternation, arity...

- $|\mathcal{A}|$, $|\varphi|$ and $|\varphi(\mathcal{A})|$: **Combined Complexity**.
- $|\mathcal{A}|$ and $|\varphi(\mathcal{A})|$: **Data Complexity**
- $|\varphi|$ and $|\varphi(\mathcal{A})|$: **Expression Complexity**

Parameterized complexity: express the complexity in terms of $|\mathcal{A}|$, $|\varphi|$ and $|\varphi(\mathcal{A})|$ but consider $|\varphi|$ as a parameter.

Some tractable cases for enumeration

Data complexity

- DEG_d^τ: the class of τ-structures of maximal degree $\leq d$.
  $\textsc{Enum}(\mathrm{FO}, \mathrm{DEG}_d^\tau) \in \mathrm{CONSTANT\text{-}DELAY}_{lin}$. (D. Grandjean'07, Lindell'08)
  Delay : triply exponential in the size of formula (Kazana, Segoufin'10)
- TW_k^τ: the class of τ-structures of treewidth at most k.
  $\textsc{Enum}(\mathrm{MSO}, \mathrm{TW}_k^\tau) \in \mathrm{LINEAR\text{-}DELAY}_{lin}$. (Bagan'06, Courcelle'07, Frick, ..)

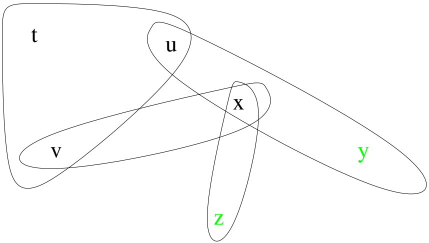
Conjunctive Queries

Conjunctive Queries (**CQ**)

- $\mathcal{L}$: $\{\exists, \wedge\}$-fragment of first-order logic i.e. formulas $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where ψ is a conjunction of atoms.
- Combined complexity of model checking: **NP**-complete. No hope for enumeration.
- Tractable fragments for decision by restricting the formula (acyclicity, hypergraph decomposition methods).

Hypergraph associated to formulas

- The *hypergraph* of a formula φ is the hypergraph $H = (V, E)$ such that
  - V is the set of variables of φ
  - for each atom $R(\bar{v})$ of φ, we associate a hyperedge $\mathrm{var}(R(\bar{v}))$



Acyclic Hypergraph

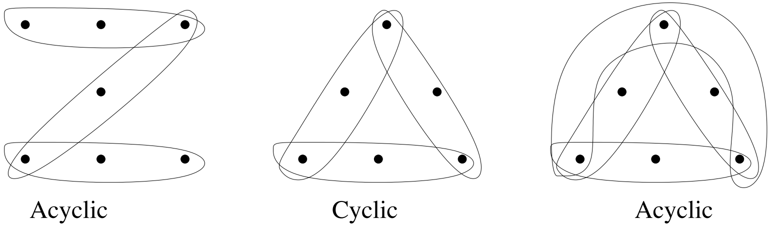
Most general notion: α-acyclicity.

An hypergraph is α-*acyclic* if the application of the following rules as long as possible outputs the empty hypergraph:

1. Remove hyperedges contained in other hyperedges;
2. Remove vertices that appear in at most one hyperedge.

A conjunctive query is acyclic if its associated hypergraph is acyclic.

Acyclic hypergraph/query: examples



Large class of hypergraph/queries.
Base case for more evolved decomposition methods.

Remark: Adding an edge to a cyclic hypergraph may result in an acyclic hypergraph.

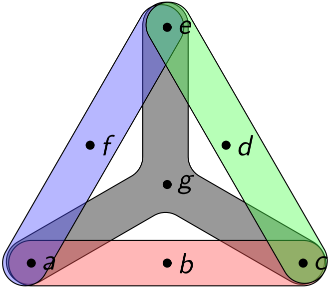
Acyclic Hypergraph: alternative definition

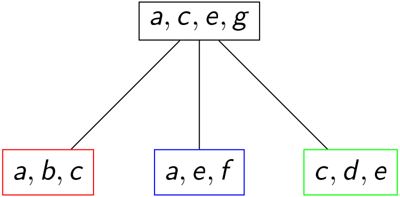
A join tree of a hypergraph $\mathcal{H} = (V, E)$ is a pair $(\mathcal{T}, \lambda)$ where $\mathcal{T} = (V_T, T)$ is a tree and λ is a function from V_T to E such that:

- For each $e \in E$, there is a $t \in V_T$ such that $\lambda(t) = e$ (each vertex of $\mathcal{T}$ is a bag containing one edge)
- For each $v \in V$, the set $\{t \in V_T : \ v \in \lambda(t)\}$ is a connected subtree of T.

Equivalent characterization : $\mathcal{H}$ is acyclic iff it has a join tree.

Acyclic Hypergraph: alternative definition





Complexity of acyclic conjunctive queries

ACQ Acyclic Conjunctive Queries.

ACQ$^{\neq}$ Acyclic Conjunctive Queries with inequalities

Example:

$\varphi(x, y) \equiv \exists z \exists t \exists u (Rxyz \wedge Ryzt \wedge Rztu \wedge Syt \wedge x \neq u)$

Known results

ACQ solvable in $O(|\varphi| \times |\mathcal{A}| \times |\varphi(\mathcal{A})|)$ times (Yannakakis'81)

ACQ$^{\neq}$ solvable in $O(2^{O(|\varphi|)} \times |\mathcal{A}| \times |\varphi(\mathcal{A})| \times \log^2(|\mathcal{A}|))$ times (Papadimitriou & Yannakakis-99) or in $O((|\varphi|!) \times |\mathcal{A}| \times |\varphi(\mathcal{A})|)$ (Bagan-D.-Grandjean-07)

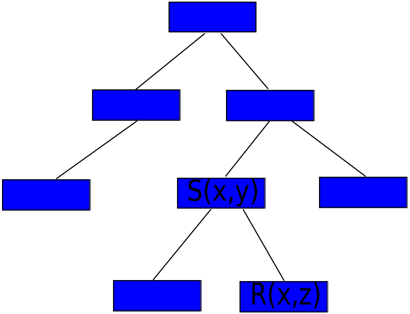
Yannakakis algorithm

Input: $(\mathcal{A}, \varphi)$, a tree decomposition T

- Take a leaf $t \in V_T$, let $R(\bar{x}, \bar{z})$ the associated atom.
- Take its father associated to $S(\bar{x}, \bar{y})$ $(\{\bar{y}\} \cap \{\bar{z}\} = \emptyset)$

Filter relation S in $\mathcal{A}$ by relation R :

$$S := \{(\bar{a}, \bar{b}) \in S \text{ and } \exists \bar{c} (\bar{a}, \bar{c}) \in R\}$$



Yannakakis algorithm

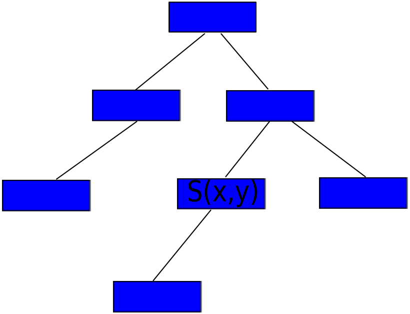
Input: $(\mathcal{A}, \varphi)$, a tree decomposition T

- Take a leaf $t \in V_T$, let $R(\bar{x}, \bar{z})$ the associated atom.
- Take its father associated to $S(\bar{x}, \bar{y})$ $(\{\bar{y}\} \cap \{\bar{z}\} = \emptyset)$

Filter relation S in $\mathcal{A}$ by relation R :

$$S := \{(\bar{a}, \bar{b}) \in S \text{ and } \exists \bar{c} (\bar{a}, \bar{c}) \in R\}$$

Continue bottom up with new data and drop $R(\bar{x}, \bar{z})$ from the list of atoms



Enumeration of **ACQ**

Enumeration

Can we enumerate the results efficiently ?

Easy but not that fast!

- let $\varphi'(x_1) \equiv \exists x_2 \dots \exists x_k \exists \bar{y} \varphi(\bar{x}, \bar{y})$
- **For** $a \in \mathrm{ENUM}(\varphi', \mathcal{A})$
  - let $\varphi_a \equiv \varphi(a, x_2, \dots, x_k, \bar{y})$
  - **For** $\bar{b} \in \mathrm{ENUM}(\varphi_a, \mathcal{A})$: output $(a, \bar{b})$

Linear $O(|\varphi| \times |\mathcal{A}|)$ delay.
Similarities with the SAT enumeration in easy cases.

Enumeration of **ACQ**

Easy and fast if all variables are free (i.e. participate to the result).

- Compute a tree decomposition
- Filter each parent with its children so that only tuples that can be extended to a solution remain ("local" consistency)
- run through the tree and output the solutions

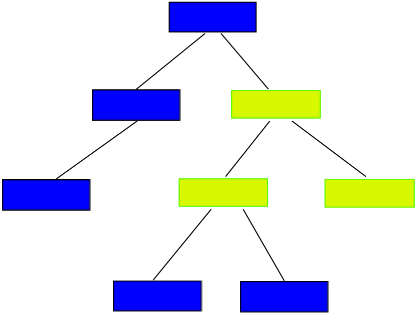
Linear $O(|\varphi| \times |\mathcal{A}|)$ precomputation and constant $O(|\varphi|)$ delay.
Works also with disequalities (but delay exponential in $|\varphi|$)

What's the point with enumeration

Tree decomposition T for φ

- Blue boxes : quantified variables only
- Yellow boxes : free variables only
- In real life : green boxes also...

Good situation

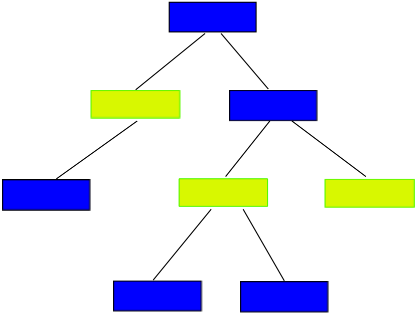


What's the point with enumeration

Tree decomposition T for φ

- Blue boxes : quantified variables only
- Yellow boxes : free variables only
- In real life : green boxes also...

Bad situation



Further question

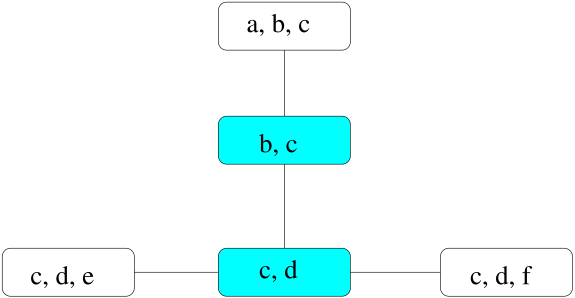
- Can we do better than linear delay (with linear precomputation) ?
- Can we find other fragments with constant delay ?
- Can we fully characterize the enumeration complexity of acyclic conjunctive queries?

Free-connex acyclic formulas

- Let $H = (V, E)$ be an acyclic hypergraph and $S \subseteq V$.
- H is S-connex acyclic if there is an acyclic hypergraph $H' = (V, E')$ and a tree-structure T of H' such that
  - $E \subseteq E'$
  - $\forall e' \in E' \exists e \in E : e' \subseteq e$ (H' inclusive extension of H)
  - there is a connex subset A of vertices of T such that $\bigcup_{t \in A} \lambda(t) = S$
- A formula φ is free-connex acyclic or $\mathbf{CCQ}^{\neq}$ if its hypergraph is free(φ)-connex acyclic.

Example of S-connex acyclic hypergraph

- Let $H = (V, E)$ with $V = \{a, b, c, d, e, f\}$ and $E = \{\{a, b, c\}, \{c, d, e\}, \{c, d, f\}\}$ and $S = \{b, c, d\}$



Other characterisation of free-connex acyclic formulas

- Let $H = (V, E)$ be a hypergraph and $S \subseteq V$. An *S-path* is a path $(x, y_1, \ldots y_n, z)$ of size at least 2 such that
  - $x \in S, z \in S$
  - for any i, $y_i \notin S$
  - there is no hyperedge $e \in E$ such that $x, z \in e$

Lemma

H is S-connex acyclic iff H is acyclic and doesn't admit any S-path. This property can be checked in polynomial time.

Free-connex acyclicity: "being grouped" up to hypergraph padding

Matrix Multiplication

Matrix Product Enumeration

Given two matrices A and B (with coefficients in some space),
- enumerate all $C_{i,j}$ or
- enumerate the indices (i, j) of $C = A \times B$ such that $C_{i,j} \neq 0$.

Boolean case: solutions of

$$\Phi(x, z) \equiv \exists y \; E(x, y) \wedge E(y, z).$$

Dichotomy result (Bagan-D.-Grandjean'07)

Theorem: Let φ be a simple **ACQ** (resp. $\mathbf{ACQ}^{\neq}$) formula and assume that the boolean matrix product <u>cannot</u> be done in linear time. Then one of the two following cases hold:

- φ is free-connex acyclic
- $\textsc{Enum}(\varphi) \in \textsc{Constant-Delay}_{lin}$
- $\textsc{Eval}(\varphi)$ can be evaluated in time $O(|\mathcal{A}| + |\varphi(\mathcal{A})|)$

or

- φ is not free-connex acyclic
- $\textsc{Enum}(\varphi) \notin \textsc{Constant-Delay}_{lin}$
- $\textsc{Eval}(\varphi)$ cannot be evaluated in time $O(|\mathcal{A}| + |\varphi(\mathcal{A})|)$

Proof of hardness

$\varphi \in \mathbf{ACQ} - \mathbf{CCQ}$ i.e., acyclic but not free(φ)-connex.
Main steps :

- Find a S-path (recall : $S = \text{free}(\varphi)$)
- From two boolean matrices, A and B construct a structure $\mathcal{A}$, in linear time, such that:
  - $\mathcal{A}$ encodes A and B.
  - Data for A and B are on each side of the S-path
  - indices of non zero coefficients of $A \times B$ are (pairs of) vertices related in the path.

Some problems to solve : the path may be arbitrary, there are other "parts" in the query, ...

Skip rest of proof

Proof of hardness

$\varphi \in \mathbf{ACQ} - \mathbf{CCQ}$ i.e., acyclic but not free-connex.
H_ϕ admits a chordless S-path $P = (x, z_1, \ldots, z_k, y)$ with $k \geq 1$, so that

- P is a path: there are $k+1$ hyperedges $e_0, e_1, \ldots, e_{k-1}, e_k \in E$ that contain $e'_0 = \{x, z_1\}, e'_1 = \{z_1, z_2\}, \ldots, e'_{k-1} = \{z_{k-1}, z_k\}, e'_k = \{z_k, y\}$
- P is an S-path: $x, y \in S$ and $z_1, \ldots, z_{k-1} \notin S$
- P is chordless: for each $e \in E, |e \cap P| \leq 1$ or $|e \cap P| = e'_i$ for some $i, (0 \leq i \leq k)$

Skip rest of proof

Proof of hardness

φ simple formula φ of the form:

$$\varphi(x, y, \bar{t}) \equiv \exists z_1 \ldots \exists z_k \exists \bar{u} \psi(x, y, \bar{z}, \bar{t}, \bar{u})$$

with ψ conjunction of atoms $A_e, e \in E$ of the following form ($\bar{v} \subseteq \{\bar{t}, \bar{u}\}$):

1. $R_e(x, z_1, \bar{v})$
2. $R_e(z_k, y, \bar{v})$
3. $R_e(z_i, z_{i+1}, \bar{v})$ where $1 \leq i < k$
4. $R_e(w, \bar{v})$ where $w \in \{x, y, z_1, \ldots, z_k\}$
5. $R_e(\bar{v})$

Proof of hardness

Transformation r of the structure: to each σ_{AB}-structure $\mathcal{A} = (D, A, B)$ associates the following $\mathcal{A}'$:

- $\mathcal{A}' = (D', (R_e)_{e \in E})$
- $D' = D \cup \{\perp\}$: here $\perp$ is a new special "padding" symbol
- For each $e \in E$ define the relation of arity p, R_e according to the form of its (unique) occurrence in φ
  1. $R_e = A \times \{\perp\}^{p-2}$
  2. $R_e = B \times \{\perp\}^{p-2}$
  3. $R_e = I \times \{\perp\}^{p-2}$ where I is the identity (equality) relation of D ($I = \{(a, a) : a \in D\}$)
  4. $R_e = D \times \{\perp\}^{p-1}$
  5. $R_e = \{\perp\}^{p}$

Proof of hardness

$\Pi(\mathcal{A}) = A \times B$

Fact 1: The map $r : \mathcal{A} \to \mathcal{A}'$ is computable in time $O(|\mathcal{A}|)$
Fact 2: $\varphi'(\mathcal{A}') = \Pi(\mathcal{A}) \times \{\perp\}^m$
Fact 3: The projection $(a, b, \bar{c}) \to (a, b)$ is a one-one mapping from $\varphi(\mathcal{A}')$ onto $\Pi(\mathcal{A})$

Conclusion for enumeration

- Preceeding result shows some dichotomy in the complexity of enumeration (not in the same sense as usual dichotomy results in complexity).
- Dichotomy distinguishes tractable and very tractable cases
- Dichotomy based on positions of free variables in the formula

$\#\mathrm{ACQ}$

Here, situation differs more drastically if quantified variables are allowed.

Theorem (Pichler, Skritek'11)

- *If only quantifier free formulas are authorized as inputs, then* $\#\mathrm{ACQ}$ *is computable in polynomial time.*
- $\#\mathrm{ACQ}$ *is* $\#\mathbf{P}$*-complete even for formulas with only one existentially quantified variable.*

Hardness result (interpretation)

Given bipartite graph G, one construct a structure $\mathcal{A}_G$ and an acyclic conjunctive formula $\varphi_G(\bar{x}) = \exists y \phi(\bar{x}, y)$ such that :
The number of perfect matching in $G = |\varphi_G(\mathcal{A}_G)|$

End of the story for $\#\mathrm{ACQ}$?

- What about weighted counting for **ACQ**?
- Hardness or tractability for counting the disjunction (or conjunction) of, say, two **ACQ**?
- One quantifier is enough to design hard instance but is that all we can say?
- Can we isolate island of tractability for $\#\mathrm{ACQ}$ with quantified variables? or better chart the tractability frontier for quantified $\#\mathrm{ACQ}$?

Weighted counting

- $\mathbb{F}$: ring with operations $+$ and $\times$
- $\mathcal{S}$: finite structure of domain D
- $\mathbb{F}$-weight function for $\mathcal{S}$: mapping $w : D \to \mathbb{F}$
- If $\bar{a} \in D^k$, the weight of $\bar{a}$ is

$$w(\bar{a}) = \prod_{i=1}^{k} w(a_i).$$

$\#_{\mathbb{F}}\mathrm{CQ}$ (resp. $\#_{\mathbb{F}}\mathrm{ACQ}$)

Input: A conjunctive query (resp. acyclic) $\Phi = (\mathcal{S}, \phi)$ and an $\mathbb{F}$-weighted function w

Output: $\sum_{\bar{a} \in \phi(\mathcal{S})} w(\bar{a})$.

When w is the constant function 1, this value is equal to $|\phi(\mathcal{S})|$.

Weighted counting

What is the complexity of $\#_{\mathbb{F}}\mathrm{ACQ}$ for quantifier free formulas?
Either in terms of number of symbolic operation or in number of bits (for spaces where iterated multiplication and addition is polynomial time)

Strategy

- Consider query instances as polynomials
- Construct efficiently arithmetic circuits that "recognize" these polynomials
- Evaluate the polynomial to solve the weighted counting problem

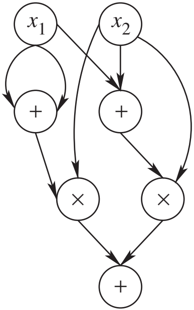
Arithmetic circuits

Arithmetic circuit over $\mathbb{F}$

- Labeled directed acyclic graph (DAG) with vertices (*gates*) with indegree (*fanin*) 0 or 2.
- Input gates have fanin 0 and are labeled with constants from $\mathbb{F}$ or variables $X_1, X_2, \ldots, X_n$.
- Computation gates have fanin 2 and are labeled with $\times$ or $+$.
- One gate with fanout 0: the output gate
- The *size* of the circuit is the number of gates
- The *depth* is the length of the longest path form an input gate to the output gate

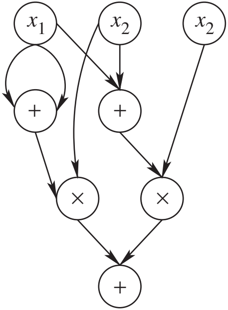
Arithmetic circuit: example

$$\begin{aligned} P(X_1, X_2) &= 2X_1X_2 + (X_1 + X_2)X_2 \\ &= 3X_1X_2 + X_2^2 \end{aligned}$$



Multiplicatively disjoint circuits

A circuit is *multiplicatively disjoint* if, for each $\times$-gate, its two input subcircuits are disjoint.



Arithmetic circuits

Valiant's setting for computation of families of polynomials by families of circuits.

Same computational power

- polynomial size circuits of polynomial degree
- multiplicatively disjoint circuits
- logarithmic depth semi-unbounded circuits.

Algebraic equivalent of **LOGCFL**...

See: Valiant, Burgisser, Koiran, Malod, ...

Polynomials representing queries

Q-Polynomials

$\Phi = (\mathcal{S}, \phi)$: conjunctive query with domain D. The following polynomial $Q(\Phi)$ in the variables $\{X_d \mid d \in D\}$ is assigned to Φ:

$$Q(\Phi) := \sum_{a \in \phi(\mathcal{S})} \prod_{x \in \mathrm{var}(\phi)} X_{a(x)} = \sum_{a \in \phi(\mathcal{S})} \prod_{d \in D} X_d^{\mu_d(a)},$$

where $\mu_d(a) = |\{x \in \mathrm{var}(\phi) \mid a(x) = d\}|$ is the number of variables mapped to d by a.

Example

$\Phi = (\mathcal{S}, \phi)$ such that
$\phi(\mathcal{S}) = \{(1,3,2,5),(1,2,1,3),(5,2,3,1),(1,1,2,1),(4,2,1,2)\}$.
Then:

$$Q(\Phi) := 2X_1X_2X_3X_5 + X_1^2X_2X_3 + X_1^3X_2 + X_2^2X_1X_4.$$

Polynomials representing queries

- The number of variables in $Q(\Phi)$ is $|D|$, the size of the domain of $\mathcal{S}$
- $Q(\Phi)$ is homogeneous of degree $|\mathrm{free}(\Phi)|$
- No bijective correspondence between solutions of the query and monomials
- Existential quantification in formulas does not correspond to projection on some polynomial variable.

$\#_{\mathbb{F}}\mathrm{ACQ}$ for quantifier free acyclic formulas

Here and after: joint work with Stefan Mengel.

$\|\Phi\| = |\mathcal{S}| + |\phi|$

Theorem

Given an acyclic quantifier free conjunctive query $\Phi = (\mathcal{S}, \phi)$, we can in time polynomial in $\|\Phi\|$ compute a multiplicatively disjoint arithmetic circuit C that computes $Q(\Phi)$.

Corollary

The problem $\#_{\mathbb{F}}\mathrm{ACQ}$ is computable in polynomial time

Skip proof

Computation of $Q(\Phi)$ for Φ acyclic: proof

By induction, adapting Yannakakis algorithms for **ACQ**

- $\Phi = (\mathcal{S}, \phi)$: the input
- $(\mathcal{T}, \lambda)$: the join tree associated with ϕ
- For $t \in V_T$, ϕ_t: conjunction of constraints corresponding to the subtree $\mathcal{T}_t$ rooted at t
- Set $e_t = \mathrm{var}(\phi_t) = \bigcup_{t' \in \mathcal{T}_t} \mathrm{var}(\lambda(t'))$

Skip rest of proof

Computation of $Q(\Phi)$ for Φ acyclic: proof

Compute the more general polynomial:

$$f_{t,\bar{a},c} = \sum_{\substack{\bar{\alpha}\in\phi_t(\mathcal{S})\\ \bar{\alpha}\sim\bar{a}}} \prod_{x\in c} X_{\alpha(x)}.$$

Where $c \subseteq \lambda(t)$, $\bar{a}$ is an assignment of some variable of ϕ and $\bar{\alpha} \sim \bar{a}$ means "$\bar{\alpha}$ agrees with $\bar{a}$ on common variables".

Strategy and Problems

- Compute inductively from some $f_{t_i,\bar{a}_i,c_i}$ where t_i is a child of t.
- If two children share a variable there is a risk of overcounting so...
- Partition the variables in $\lambda(t)$ and assign parts to children

Skip rest of proof

Computation of $Q(\Phi)$ for Φ acyclic: proof

- The case of a leaf $t \in V_T$ is immediate.
- Let $t \in V_T$ and $t_1, ..., t_k$ its children in V_T
- $c_0, c_1, ..., c_k$: partition of c into disjoint sets such that $c_i \subseteq e_i \cap c$, for $i = 1, ..., k$ and $c_0 \subseteq c \setminus \bigcup_{i=1}^{k} e_{t_i}$

$$
\begin{aligned}
f_{t,\bar{a},c} &= \sum_{\substack{\bar{\alpha} \in \phi_t(\mathcal{S}) \\ \bar{\alpha} \sim \bar{a}}} \prod_{x \in c} X_{\alpha(x)} \\
&= \sum_{\substack{\bar{\alpha} \in \phi_t(\mathcal{S}) \\ \bar{\alpha} \sim \bar{a}}} \prod_{x \in c_1} X_{\bar{\alpha}(x)} \cdots \prod_{x \in c_k} X_{\bar{\alpha}(x)} \prod_{x \in c_0} X_{\bar{\alpha}(x)}
\end{aligned}
$$

Computation of $Q(\Phi)$ for Φ acyclic: proof

Last remark

- Let $A_t = ((\lambda_t(\mathcal{S}) \ltimes \phi_{t_1}(\mathcal{S})) \ltimes \phi_{t_2}(\mathcal{S})) \ltimes \ldots \ltimes \phi_{t_k}(\mathcal{S})$.
- Each solution $\bar{\alpha} \in \phi_t(\mathcal{S})$ can be uniquely expressed as the natural join of a $\bar{\beta} \in A_t$ with some $\bar{\alpha}_i \in \phi_{t_i}(\mathcal{S})$, $i = 1, \ldots, k$, compatible with β (converse also true)

$$
\begin{aligned}
f_{t,\bar{a},c} &= \sum_{\substack{\bar{\alpha}\in\phi_t(\mathcal{S}) \\ \bar{\alpha}\sim\bar{a}}} \prod_{x\in c_1} X_{\bar{\alpha}(x)} \cdots \prod_{x\in c_k} X_{\bar{\alpha}(x)} \prod_{x\in c_0} X_{\bar{\alpha}(x)} \\
&= \sum_{\substack{\bar{\beta}\in A_t \\ \bar{\beta}\sim\bar{a}}} \sum_{\substack{\bar{\alpha}_1\in\phi_{t_1}(\mathcal{S}) \\ \bar{\alpha}_1\sim\bar{\beta}}} \cdots \sum_{\substack{\bar{\alpha}_k\in\phi_{t_k}(\mathcal{S}) \\ \bar{\alpha}_k\sim\bar{\beta}}} \prod_{x\in c_1} X_{\bar{\alpha}_1(x)} \cdots \prod_{x\in c_k} X_{\bar{\alpha}_k(x)} \prod_{x\in c_0} X_{\bar{\beta}(x)} \\
&= \sum_{\substack{\bar{\beta}\in A_t \\ \bar{\beta}\sim\bar{a}}} f_{t_1,\beta,c_1} \cdots f_{t_k,\beta,c_k} \cdot \prod_{x\in c_0} X_{\bar{\beta}(x)}
\end{aligned}
$$

Union and intersection of acyclic queries

Theorem

*Computing the size of the union and the intersection of query results to two quantifier free $\#\mathrm{ACQ}$-instances are both $\#$**P**-complete. This result remains true for $\#\mathrm{ACQ}$ on boolean domain and arity at most 3.*

Proof by chain of reductions

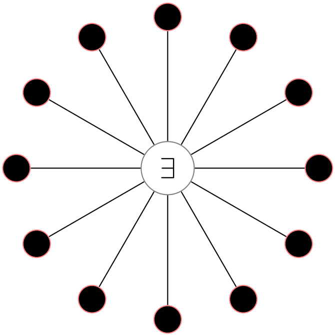
- $\#\mathrm{circuitSAT}$
- $\#(\wedge\text{-}\neg\text{-grid})\text{-circuitSAT}$: circuit sat with $\wedge$ and $\neg$ gate on a grid.
- $\#\mathrm{CQ}$ on a grid

Pause to ponder

Bad news

- The conjunction (or disjunction) of two **ACQ** makes counting hard
- Idem for **ACQ** for one existentially quantified variable.

But... formula in Pichler and Skritek's hardness proof looks like that:



Is it a sign of hardness?

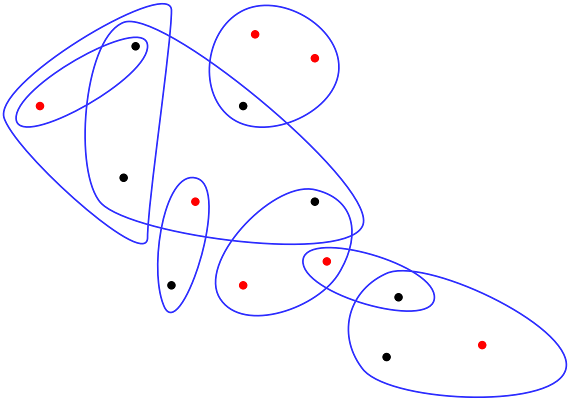
S-components

A hypergraph $\mathcal{H} = (V, E)$ and $S \subseteq V$.
Let $E_{\not\subseteq S} = \{e \in E : e \not\subseteq S\}$.

S-component

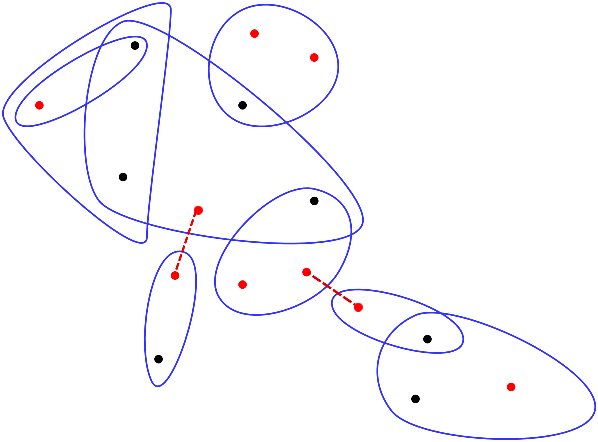
The *S-component* of $e \in E_{\not\subseteq S}$ is the hypergraph $\mathcal{H}[E']$ where E' is the set of all edges $e' \in E_{\not\subseteq S}$ such that there is a path from $e - S$ to $e' - S$ in $\mathcal{H}[V - S]$.
A subhypergraph $\mathcal{H}'$ of $\mathcal{H}$ is an S-component if there is an edge $e \in E_{\not\subseteq S}$ such that $\mathcal{H}'$ is the S-component of e.

Decomposition into S-components: example



S vertices in red

Decomposition into S-components: example



S vertices in red

Definition (S-k-star, S-star size)

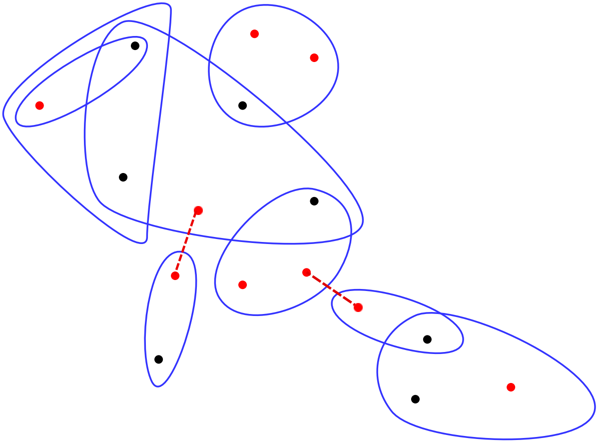
Let $\mathcal{H} = (V, E)$ be a hypergraph, $S \subseteq V$ and $k \in \mathbb{N}$. The subhypergraph $\mathcal{H}' = (V', E')$ of $\mathcal{H}$ is a *S-k-star* if:

- $\mathcal{H}'$ is an S-component of $\mathcal{H}$.
- there exist $y_1, ..., y_k \in V' \cap S$ such that there is no edge $e \in E$ that contains more than one of the y_i.

$y_1, \ldots, y_k$ are "independant" and form the S-k-star.

The S-star size of $\mathcal{H}$ is the *maximum* k such that there is a S-k-star in $\mathcal{H}$.

S-star size : example



S-star size is 4

Quantified star size

The *quantified star size* of an acyclic conjunctive formula $\phi(\bar{x})$ is the S-star size of the hypergraph $\mathcal{H}$ associated to $\phi(\bar{x})$, where S is the set of free variables in $\phi(\bar{x})$.

quantified-star size: example

Formula $\phi(x, y) \equiv \exists t \exists z R(x, y, t) \wedge S(x, z, t)$

Quantified star size $= 1$

Path formulas (of arbitrary length), e.g.
$\phi(x, y, z) \equiv \exists t_1 \exists t_2 R(x, t_1) \wedge R(t_1, z) \wedge R(z, t_2) \wedge R(t_2, y)$

Quantified star size $= 2$.

Star formulas, e.g.
$\phi(x, y, z, t) \equiv \exists u R(u, x) \wedge R(u, y) \wedge R(u, z) \wedge R(u, t)$

Quantified star size $=$ degree of the center of the star (here 4).

S-star size: main results I

Query evaluation is easy

Theorem

There is an algorithm that given an acyclic conjunctive query Φ computes an arithmetic circuit C that computes $Q(\Phi)$. The runtime of the algorithm is $\|\Phi\|^{O(k)}$ where k is the quantified star size of Φ.

Corollary

There is an algorithm for the problem $\#\mathrm{ACQ}$ that runs in time $\|\Phi\|^{O(k)}$ where k is the quantified star size of the input query Φ.

Some remarks on the proof

Star size expresses how pieces of the result are spread

- Computation of the result is made component by component
- For each component, if quantified-star size is k and $R_1(\bar{z}_1), ..., R_k(\bar{z}_k)$ atoms of ϕ containing all free variables of ϕ, one needs to pre-compute all k-tuples

$$\bar{a}_1 \in R_1, ..., \bar{a}_k \in R_k$$

and check if they lead to a solution of $\phi(\mathcal{S})$ (by the method for $Q(\Phi)$ when Φ acyclic and quantified free)

S-star size: main results II

Computing S-star is easy

Theorem

There is a polynomial time algorithm that, given a hypergraph $\mathcal{H} = (V, E)$ and $S \subseteq V$, computes the S-star size of $\mathcal{H}$.

Proof

Equivalent to finding a maximal independant set (and a minimal edge cover) in an acyclic hypergraph.
Ad hoc algorithm.

Conclusion: Classes of $\#\mathrm{ACQ}$-instances of bounded quantified star size are efficiently decidable.

Go to: star size needed

Parametrizations of $\#\mathrm{ACQ}$

- p-star-$\#\mathrm{ACQ}$: counting parameterized by the quantified star size,
- p-var-$\#\mathrm{ACQ}$: counting parameterized by the number of free variables,
- p-$\#\mathrm{ACQ}$: counting parameterized by the size of the conjunctive formula.

Lemma

p-star-$\#\mathrm{ACQ}$, p-var-$\#\mathrm{ACQ}$ and p-$\#\mathrm{ACQ}$ are all $\#\mathbf{W}[\mathbf{1}]$-hard.

Bounded quantified star size is necessary

S-hypergraph: pair $(\mathcal{H}, S)$ where $\mathcal{H} = (V, E)$ is a hypergraph and $S \subseteq V$.

Definition

$\#\mathrm{ACQ}$ is tractable for a class $\mathcal{G}$ of S-hypergraphs if for all $\#\mathrm{ACQ}$ instances Φ with the associated hypergraph $\mathcal{H}$ of Φ and the set S of free variables of Φ with $(\mathcal{H}, S) \in \mathcal{G}$ we can solve $\#\mathrm{ACQ}$ in polynomial time

Bounded quantified star size is necessary

Theorem

Assume $\mathbf{FPT} \neq \#\mathbf{W[1]}$, *and let* $\mathcal{G}$ *be a recursively enumerable class of acyclic* S*-hypergraphs. Then* $\#\mathrm{ACQ}$ *is polynomial time solvable for* $\mathcal{G}$ *if and only if* $\mathcal{G}$ *is of bounded* S*-star size.*

Proof sketch

- First prove that $\#\mathrm{ACQ}$ is hard on $\mathcal{G}_S$ the class of k-stars (in the graph sense)
- Consider an instance $\Phi = (\mathcal{A}, \varphi(\bar{x}))$ with $\varphi(\bar{x}) = \exists y \bigwedge_{i=1}^{k} R_i(y, x_i)$
- Suppose there is a class $\mathcal{G}$ of unbounded S-star size which is tractable.
- Embed efficiently Φ into a suitable instance (with big enough star) Ψ whose associated hypergraph is in $\mathcal{G}$.
- Deduce that $\#\mathrm{ACQ}$ is not hard on $\mathcal{G}_S$.

Conclusion

- Complete characterization of tractability frontier for $\#\mathrm{ACQ}$
- Easy evaluation for counting on bounded S-star size extend to generalized hypertree width
- S-k-star size recognition seem to depend on k for some decomposition