An introduction to algorithms and complexity of counting and enumeration problems

Arnaud Durand

Université Paris Cité

Tartu
June 10, 2025

Introduction and examples

Some algorithmic tasks

Decision Problem decide if an input x satisfies a given property P

Counting Problem count the number of (solutions) y that witness that x satisfies P

Enumeration problem generate one by one all such witnesses y

Example

- Count/Generate all satisfying assignments of a propositional formula
- Count/Generate all independent sets (or cliques) of maximum size of a graph
- Count/Generate all maximal (for inclusion) independent sets

Another context: query problem(s)

Teacher	Course
James	A21
Galois	B21
Cauchy	B21
Johnson	C10
Galois	A21

Course	Level
A21	Advanced
B21	Basic
C10	Advanced

Typical queries:

- Is there a teacher for each basic course: yes
- List of teachers of advanced courses : James, Johnson, Galois
- Number of teachers of advanced courses : 3

Obtaining the results one by one makes sense

Objectives of the talk

- Give an introduction to the complexity of counting and enumeration
- Through examples in graphs and satisfiability problem and query answering in data management
- Present the main complexity measures in particular to define tractability notions (not obvious for enumeration)
- Illustrate, in the context of query answering in databases, some conditions that makes counting/enumeration problems tractable. Among them:
  - graphs decomposition and sparsity notions for data
  - nice decomposition properties of queries

Counting problems

Examples

- Natural decision problems have a counting extension
  - Count the number of satisfying assignments of a propositional formula: ♯SAT
  - Count the number of Hamiltonian paths in a graph: ♯HAM
- Sometimes counting is hidden
  - Related to polynomial evaluation or computation of monomials coefficients (graph, knots and link polynomials).
  - Permanent of matrices

Permanent (and determinant) of a matrix

Permanent of a $n \times n$ matrix:

$$permanent(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} A_{i,\pi(i)}$$

Determinant of a $n \times n$ matrix:

$$det(A) = \sum_{\pi \in S_n} \sigma(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}$$

where σ is the signature of the permutation i.e. $\sigma(\pi) = -1$ if π is the product of an odd number of transpositions and $\sigma(\pi) = 1$ if not.

Permanent for a $\{0,1\}$-matrix : 0,1-Perm

Why is 0,1-Perm a counting problem?

Equivalent to

- Counting the number of perfect matchings of a bipartite graph
- Counting the number of cycle covers in a graph

P and NP

Polynomial time (only prerequisite of this talk....)

A $\in$ P if it can be decided in polynomial time by a Turing machine

Definition

A problem B is in NP if there exists a binary predicate A : $\Sigma^* \times \Sigma^* \to \{0,1\}$ and a polynomial $p \in \mathbb{N}[X]$ such that

- A $\in$ P
- For all $x \in A$:

$$x \in \mathsf{B} \Leftrightarrow \exists y \in \Sigma^*,\ \mathsf{A}(x,y) \wedge |y| \leq p(|x|)$$

Example: 3-colorability of a graph G.
$A(G,c)$: check whether $c : V \to \{1,2,3\}$ describe a suitable 3-coloring of G.

Defining hard counting problems

Definition

A function $f : \Sigma^* \to \mathbb{N}$ is in ♯P if there exists a binary predicate $\mathsf{A} : \Sigma^* \times \Sigma^* \to \{0, 1\}$ and a polynomial $p \in \mathbb{N}[X]$ such that

- $\mathsf{A} \in \mathsf{P}$
- For all $x \in \Sigma^*$:

$$f(x) = \sharp\{y : \mathsf{A}(x, y) \wedge |y| \leq p(|x|)\}$$

Notations

♯A: counting problem.

Hardness of counting problems

Informally...

- Some counting problems with polynomial time witness checking predicates are harder than others: ♯P-completeness
- Several notion of completeness:
  - Build over the underlying decision problem (many-one parsimonious: bijection between solution sets)
  - Direct reduction between counting functions (Turing reduction)
  - Some technics : polynomial interpolation, subtractive and set-based reductions, holographic reductions
- Counting version of most NP-complete decision problems are ♯P-complete (if it is hard to decide the existence of one solution, it must be hard to count them all...).

Complexity of 0,1-Perm

Theorem (Valiant'79)

0,1-Perm *is* ♯P*-complete (for Turing reduction).*

- Deciding the existence of a perfect matching in a bipartite graph can be done in polynomial time.
- First example of an "easy-to-decide but hard-to-count" behaviour...
- Many natural polynomial time decidable problems share this property.

Remark (informal): proving that the permanent of any $n \times n$-matrix can be "equivalently replaced" by the determinant of another matrix of reasonable size (polynomial in n) is stronger than proving P $=$ NP

Counting covering trees in graphs

Let $G = (V, E)$ be a graph.

Covering tree

$T = (V', E')$ is a covering tree of a graph $G = (V, E)$ if T is a connected acyclic graph such that:

- $V' = V$
- $\forall x, y \in V \ xy \in E' \Rightarrow xy \in E$

Let $\mathcal{T}_G$ be the set of covering trees of G (potentially of exponential size).

$V = \{v_1, \ldots, v_n\}$ and $G = (V, E)$: connex, without self-loops
A : adjacency matrix of G
D : diagonal matrix with, for all $i \leq n$:

$$d_{ii} = deg_G(v_i) = |\{v : vv_i \in E\}|$$

Counting covering trees in graphs

How hard is it to count the number of covering trees of a graph?

Let $i, j \leq n$, A_{ij}: minor matrix of A without the i^{th} line and j^{th} column.

Theorem (Kirchhoff)

For all $i \in \{1, ..., n\}$*:*

$$|\mathcal{T}_G| = \det(D - A)_{ii}$$

- The choice of a minor has no importance
- Counting can be done in polynomial time (since computing a determinant is easy)

Summary (for complexity)

- Counting is harder than deciding the existence of a solution
- Frequent "easy-to-decide but hard-to-count" behaviour...
- Very few natural polynomial time decision problems are also easy to count
  - Counting covering trees
  - Counting perfect matchings in planar graphs
  - Hard graph counting problems but on "nicely"-decomposable graphs (*bounded treewidth*,...)
- Some particular : counting the number of isomorphisms between two graphs is not so far (in terms of hardness) from deciding isomorphism

Enumeration of solutions of combinatorial problems

Enumeration problems

- Task : generate all solutions without repetition of a given instance of a combinatorial problem.
- Complexity focuses on the dynamic of the generation process
- Measures of complexity (historically) defined for problems whose solutions can be verified in polynomial time (i.e. NP decision problem).
- Analog of NP: EnumP.
  A function f is in EnumP if there exists a polynomial time binary predicate B, such that, for all input x:

$$f(x) = \{y : (x, y) \in B\}$$

- Relatively recent but vast literature centered on tractable problems

Complexity measures for enumeration

The number of solutions does not matter, only the dynamics

Tractable classes inside EnumP**:**

Total Polynomial Time the solution set can be computed in time polynomial in the input and output size.

Incremental Polynomial Time the $i+1$-th solution set can be computed in time polynomial in the input size and i.

Polynomial Delay all solutions can be computed one after the other with polynomial time between them.

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. Johnson, Papadimitriou, Yannakakis (88):
  - polynomial delay (and memory size) 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

Flashlight Method Given a propositional formula φ with variables $x_1, \ldots, x_n$.

- If $\varphi \wedge \neg x_1 \in SAT$: branch on enumerating all solutions of $\varphi \wedge \neg x_1$
- If $\varphi \wedge x_1 \in SAT$: branch on enumerating 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.
Easy (i.e. polynomial delay) fragments of SAT are Horn, Dual-Horn, 2-CNF and affine.

Minimal satisfiability problem

Related to circumscription in A.I.

$\varphi(x_1, \ldots, x_n)$ propositional formula with $x_1, \ldots, x_n$.
Pointwise ordering: Let $v = (v_1, \ldots, v_n)$ and $v' = (v'_1, \ldots, v'_n)$ then:

$$v' \leq v \iff v'_i \leq v_i \text{ for } 1 \leq i \leq n$$

Minimal model of φ : $v = (v_1, \ldots, v_n)$ is a minimal model of φ if:

1. $v \models \varphi$
2. $\forall v' \leq v,\ v' \models \varphi$ implies $v = v'$.

Example:
$\varphi = (p \vee q \vee r)$. $v = (1, 0, 0)$ is minimal but not $v' = (1, 1, 0)$.

Enumeration of *minimal* satisfiability problems

- General case difficult (decision for minimal model is **coNP**-complete).
- Krom formulas (*two literals per clause*): polynomial delay (Kavvadias, Sideri, Stavropoulos'00).
- Horn formulas (*at most one positive literal*): only one minimal model...
- Dual-Horn formulas (*at most one negative literal*): no incremental polynomial time algorithm if $\mathbf{P} \neq \mathbf{NP}$ (Kavvadias, Sideri, Stavropoulos'00).
- Positive monotone formulas (*no negative literal*) : equivalent to Hypergraph Transversal. *Open Problem*.
- Affines formulas (conjunction of $\oplus$ clauses i.e. linear system of Boolean equations): ...

Incremental polynomial time: Affine formulas

Related to enumeration of all circuits of a matroid. Homogeneous case: equations of the form $x_1 \oplus ... \oplus x_k = 0$.

Closure axiom

If $v = (v_1, \ldots, v_n)$ and $v' = (v'_1, \ldots, v'_n)$ (with some $i \leq n$ s.t. $v_i = v'_i = 1$) are two minimal solutions then there exists a minimal solution smaller than $v \oplus v'$.

Khachyan, Boros, Elbassioni, Gurvich, Makino 05:

There is an incremental polynomial algorithm for enumeration of minimal model of affine formulas.

How ?

- Start with some suitable easily computable set of minimal solutions (the *basic* solutions).
- Iterate on all possible pairs the test of the closure axiom (goes through many *non* solutions.. but not too many).

Is there a polynomial delay algorithm?

Counting and enumeration for database query problems

Query problem(s): back to the example

Teacher	Course
James	A21
Galois	B21
Cauchy	B21
Johnson	C10
Galois	A21

Course	Level
A21	Advanced
B21	Basic
C10	Advanced

Typical queries:

- Is there a teacher for each basic course: yes
- List of teachers of advanced courses : James, Johnson, Galois
- Number of teachers of advanced courses : 3

Is polynomial delay adequate as a measure of *efficiency* in this context?

Some notations on query problems

Classical Boolean query (a.k.a model checking) problem: Given a finite structure $\mathbf{D}$ and a sentence φ (i.e. a Boolean *query*) decides if

$$\mathbf{D} \models \varphi. \qquad (i.e.\ \mathbf{D} \in \mathsf{Mod}(\varphi))$$

More generally, for a formula $\varphi(\mathbf{x})$ compute

$$\varphi(\mathbf{D}) = \{\mathbf{a} : (\mathbf{D}, \mathbf{a}) \models \varphi(\mathbf{x})\}$$

The counting problem: $\varphi(\mathbf{D}) = \sharp\{\mathbf{a} : (\mathbf{D}, \mathbf{a}) \models \varphi(\mathbf{x})\}$

First-order query problems

Queries: All about first-order queries, for short FO:

$$\varphi(x) := \forall y \ (B(y) \wedge E(x,y)) \vee \neg R(x)$$

Let finite colored graph $\mathbf{D} = \langle D, E, B, R \rangle$ where D is the domain, E is binary, B, R are unary relations.

Then $\varphi(\mathbf{D})$ is the set of vertices $a \in A$ such that

"a is not R(ed) or a is E-linked to all B(lue) node"

Special attention to Conjunctive Queries or CQ, the existential conjunctive fragment of FO:

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

or, for short, $q(x,y) := E(x,z), E(z,y)$

Query problems: input sizes

These are computational problems, decidable for FO. One needs to have notions of sizes:

- $\|\varphi\|$: informally, length of an encoding of φ. For example, as a binary word.
- **Database:** finite (not always) relational structure $\mathbf{D}$ on domain D, on a vocabulary σ.
- Database size: $\|\mathbf{D}\|$

$$\|\mathbf{D}\| = |\sigma| + |D| + \sum_{R \in \sigma} |R| \cdot ar(R)$$

  where $ar(R)$ is the arity of R.

- **Model of Computation:** Random Access Machines, RAM, with uniform cost (but reasonable set of operations)

Complexity of first-order query problem: basic results

- The combined complexity of the FO Boolean query problem is PSPACE-complete
- Simplified view: algorithm in $O(\|\varphi\| \cdot \|\mathbf{D}\|^{w(\varphi)})$ where $w(\varphi)$ is the maximal number of free variables in a sub-formula of φ.

Can we do better? For example, make the exponent independent of φ?

- No algorithm in $O(\|\mathbf{D}\|^c)$ for some fixed $c \in \mathbb{N}$ if P $\neq$ PSPACE. Stolboushkin, Taistlin'94
- No algorithm in $O(f(\|\varphi\|) \cdot \|\mathbf{D}\|^c)$, for some computable function f and some fixed $c \in \mathbb{N}$ under some parameterized complexity hypothesis (AW[$*$] $\neq$ FPT).
  To build the intuition (k-clique problem):

$$\varphi := \exists x_1 \ldots \exists x_k \bigwedge_{i<j=1}^{k} x_i \neq x_j \wedge E(x_i, x_j).$$

$w(\varphi) = k$.

Still about cliques and complexity of first-order model checking

FO^k: First-order sentences with k quantified variables.

Williams'14 (on n vertices graphs):

- For $k \geq 3$, there is a $O(n^{k-3+\omega})$ algorithm where ω is the exponent of the matrix multiplication problem (currently: $\omega = 2,371$).
- For $k \geq 9$, there is a $O(n^{k-1+o(1)})$ algorithm
- $O(n^{k-1})$ is essentially optimal: For $k \geq 4$, no $O(n^{k-1-\epsilon})$ algorithms for all $\epsilon > 0$ under the *Strong Exponential Time Hypothesis*[1]

Remark: improving algorithms in the general case not possible unless breaking widely believed hypotheses

[1] For every constant $\epsilon < 1$, there exists k, such that $k - SAT$ cannot be solved in time $2^{\epsilon n}$ on formulas with n variables

Complexity measures for enumeration... in DB context

- In the DB context, polynomial (even linear) delay is not adequate as a definition of tractability
- Better consider delay
  - depending on *each* output (tuple) size
  - depending on the query size a.k.a. constant delay
  - depending on some other parameter (?)
- But then, need to isolate the time before computing the first solution: the preprocessing

Measuring enumeration for query problems

Let $\delta, \mathbf{p} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, Logic $\mathscr{L}$, Class $\mathscr{C}$

Enumeration Algorithm

The *enumeration problem* of $\mathscr{L}$ over $\mathscr{C}$ can be solved with delay δ after preprocessing $\mathbf{p}$, if there exists a RAM algorithm which, on input $q \in \mathscr{L}$ of size k and $\mathbf{D} \in \mathscr{C}$ of size n, performs:

- a preprocessing phase in time $\mathbf{p}(k, n)$, that produces the first solution and
- an enumeration phase that outputs elements of $q(\mathbf{D})$ without repetition and delay $\delta(k, n)$ between two consecutive outputs.

First order query as an enumeration problem

Objective when computing $\varphi(\mathbf{D}) = \{\mathbf{a} : (\mathbf{D}, \mathbf{a}) \models \varphi(\mathbf{x})\}$: after some preprocessing, enumerate every elements of $\varphi(\mathbf{D})$ one by one without repetition with the best delay possible between two consecutive output.

Constant Delay (D-Grandjean'07)

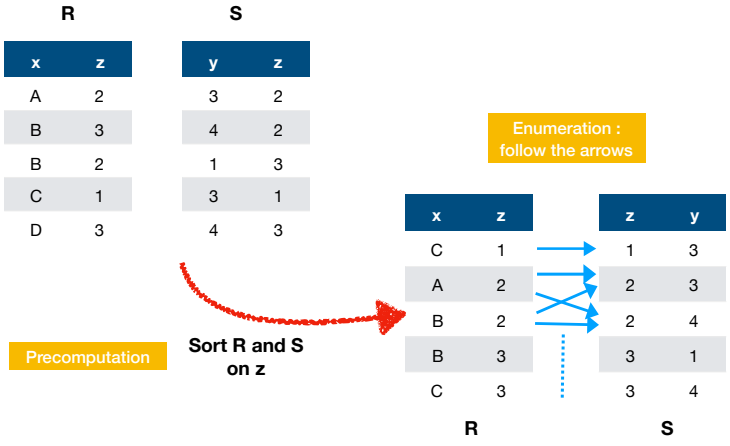
Let $f : \mathbb{N} \to \mathbb{N}$, computable. The *enumeration problem* of $\mathscr{L}$ over $\mathscr{C}$ can be solved with constant delay after linear preprocessing, if there exists a RAM algorithm which, on input $\varphi \in \mathscr{L}$ of size k and $\mathbf{D} \in \mathscr{C}$ of size n, performs:

- a preprocessing phase in time $f(k) \cdot n$, that produces the first solution and
- an enumeration phase that outputs elements of $\varphi(\mathbf{D})$ without repetition and delay $f(k)$ between two consecutive outputs.

Total time : $O(\|\mathbf{D}\| + |\varphi(\mathbf{D})|)$

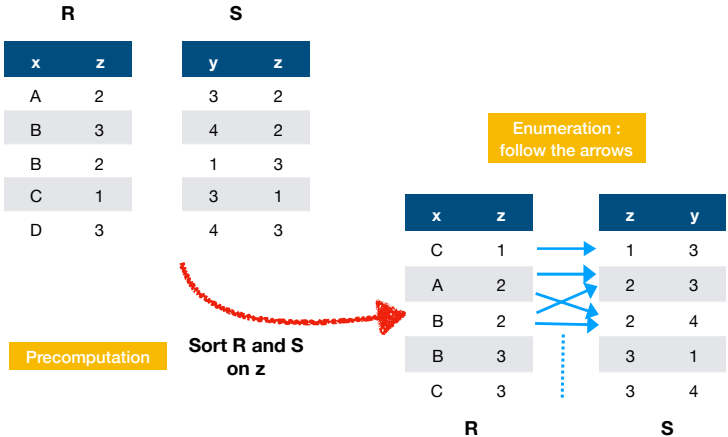
Are there problems computable in constant delay?
Examples

- $\varphi(x, y) := E(x, y)$. Easy...
- $\varphi(x, y, z) := R(x, z) \wedge S(z, y)$ and $\phi(x, y) := \exists z R(x, z) \wedge S(z, y)$?



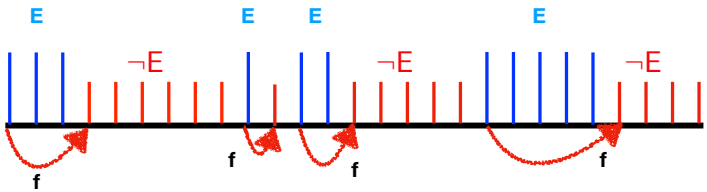
Are there problems computable in constant delay?
Examples

- $\varphi(x, y) := E(x, y)$. Easy...
- $\varphi(x, y, z) := R(x, z) \wedge S(z, y)$ and $\phi(x, y) := \exists z R(x, z) \wedge S(z, y)$?



Example: Query $q(x, y) := \neg E(x, y)$

- Can not go through pairs directly: consecutive edges prevents to guarantee the delay.
- Precomputation steps:
  - Fix an order of vertices, sort the pairs $(x, y) \in E$ lexicographically.
  - Built a function f that maps each pair $(x, y) \in E$ to the first $(x', y') > (x, y)$ such that $(x', y') \notin E$. linear time process
- Enumeration:
  - By default, go through successive pairs. If $(x, y) \notin E$, output (x, y)
  - If $(x, y) \in E$, continue with $f(x, y)$.



Using constant delay as a measure of efficiency

- First-order query are hard (polynomial exponent depends on the query size), enumeration and counting as well
- Many works have looked for a classification of tractable/intractable classes of queries. Either by :
  - restricting the class of structures
  - restricting the class of queries
- Constant delay enumeration appears as a central class for tractability of many types of query problems

Case studies

- FO queries on restricted data
- Complexity classification that illustrate the role of sparsity for structure
- Convergence with similar notions in model theory and combinatorics (stability, NIP, VC dimension)

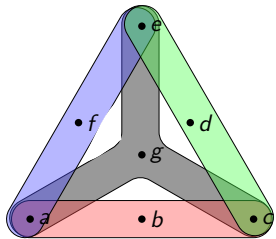
- Conjunctive queries on arbitrary data
- Role of acyclicity (and more) and of free variables
- Tractability map different for enumeration and counting

Tractability of query answering for fragments of conjunctive queries

- FO and even CQ queries are hard
- Under which conditions on the query can we hope for
  - A decision and counting algorithm in $O(f(\|q\|) \cdot \|\mathbf{D}\|^c)$, for some $c \in \mathbb{N}$ or, even better with $c = 1$.
  - An enumeration algorithm with constant delay and linear preprocessing
- It turns out that acyclicity and topological properties of free variables are key notions

Finite hypergraph $\mathcal{H} = (V, E)$, V finite, $E \subseteq \mathscr{P}(V)$.

To each query φ, one can associate an hypergraph $\mathcal{H} = (V, E)$ whose vertex set is the set *var*(φ) of variables of φ and hyperedge set is *atom*(φ) the set of atoms of φ.



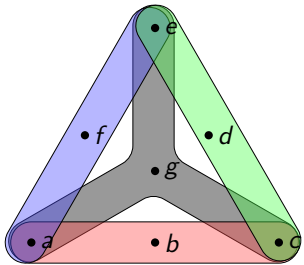
$$\varphi := R(a, b, c), R(c, d, e), R(e, f, a), S(a, c, e, g)$$

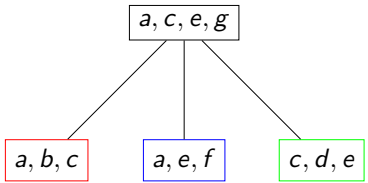
ACQ: acyclic conjunctive queries

A **join tree** of an hypergraph $\mathcal{H} = (V, E)$: tree T whose set of nodes is E and whose edge set is such that:

- for all $v \in V$, $\{e \in E : v \in e\}$ the set of nodes of T in which v occurs is a connected sub-tree of T.

ACQ A CQ is acyclic if its associated hypergraph has a join-tree (very natural class).





$$\varphi := R(a,b,c), R(c,d,e), R(e,f,a), S(a,c,e,g)$$

Computing acyclic queries

Theorem (Yannakakis'81)

There is an algorithm which, upon input of a database $\mathbf{D}$ *and* $\varphi(\mathbf{x}) \in \mathrm{ACQ}$*, computes the set* $\varphi(\mathbf{D})$ *in time*

$$O(\|\varphi\| \cdot \|\mathbf{D}\| \cdot \|\varphi(\mathbf{D})\|).$$

Boolean or queries with 1 free variable: $O(\|\varphi\| \cdot \|\mathbf{D}\|)$

ACQ: Enumeration in linear delay

Yannakakis algorithm can be easily turned into an enumeration algorithm with linear delay.

Input: $\varphi(x_1, x_2, ..., x_p)$
if $p = 1$ **then**
  for $a \in \varphi(\mathbf{D})$ **do**
    output a
else
  let $\psi(x_1) \equiv \exists x_2 \dots \exists x_p \varphi(x_1, \dots x_p)$ 1 free variable
  for $a \in \psi(\mathbf{D})$ **do**
    let $\varphi_a \equiv \varphi(a, x_2, \dots, x_p)$ $p-1$ free variables: induction
    for $\bar{b} \in \varphi_a(\mathbf{D})$ **do**
      output $(a, \bar{b})$

Can we do better?

- Is it possible to obtain constant delay enumeration?
- If yes, in which cases ?
- Can we characterize the complexity of enumeration for ACQ or even CQ?

First observation (by analyzing the preceding algorithm):

- Delay can be constant if all variables are free.
- Probably not true (unless surprise) if some variables can be projected (i.e. existentially quantified). Recall Boolean Matrix Multiplication : $\Pi(x, y) := A(x, z), B(z, y)$.

ACQ: improving the delay

An ACQ $\varphi(\mathbf{x})$ is free-connex if the extended query

$$\varphi'(\mathbf{x}) := \varphi(\mathbf{x}), R(\mathbf{x})$$

is acyclic where R is an arbitrary new symbol.

Example

- $\varphi(x, y) := E(x, w), E(y, z), B(z)$ is free-connex
  -> Query $\varphi'(x, y) := E(x, w), E(y, z), B(z), R(x, y)$ is still acyclic.
- Boolean matrix multiplication $\Pi(x, y) := A(x, z), B(z, y)$.
  -> Since $\Pi'(x, y) = A(x, z), B(z, y), R(x, y)$ is not acyclic, $\Pi(x, y)$ is not free-connex.

free-connexity and tractability of enumeration of acyclic queries

From Bagan, D., Grandjean'07

- For every free-connex acyclic conjunctive query φ, one can enumerate the set $\varphi(\mathbf{D})$ with linear-time preprocessing and constant-time delay.
- Every non free-connex acyclic conjunctive query (without self-join) φ can "encode" the Boolean Matrix Multiplication problem

Follow-up

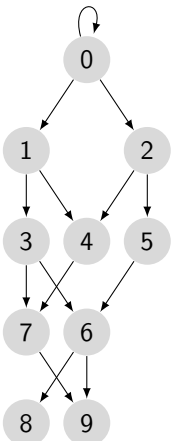
Constant delay has served as the central notion of tractability for many classes of queries

- Union of conjunctive queries
- Queries with self-joins
- Queries with negations and constraint satisfaction problems
- Queries for (SLP) compressed data, factorized representations
- Queries with updates
- etc

Is it easy to count results of ACQ?

$$\varphi(v, x_1, x_2, w) \equiv E(v, x_1) \wedge E(x_1, x_2) \wedge E(x_2, w)$$

To be evaluated on the following structure $\mathcal{A} = (V, E)$:

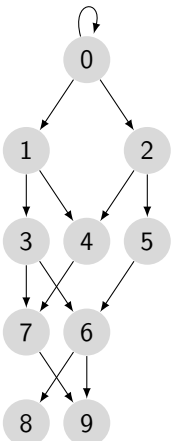


One associates the polynomial $Q(\Phi)(X_0, ..., X_9)$ below :

Is it easy to count results of ACQ?

$$\varphi(v, x_1, x_2, w) \equiv E(v, x_1) \wedge E(x_1, x_2) \wedge E(x_2, w)$$

To be evaluated on the following structure $\mathcal{A} = (V, E)$:



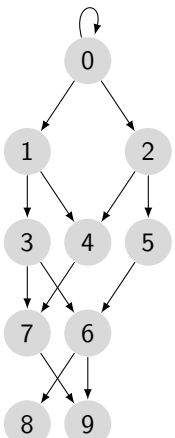
One associates the polynomial $Q(\Phi)(X_0, ..., X_9)$ below :

$$X_0^4 + X_0^3 X_1 + X_0^3 X_2 +$$
$$X_0^2 X_1 X_3 + X_0^2 X_1 X_4 + X_0^2 X_2 X_4 + X_0^2 X_2 X_5 +$$
$$X_0 X_1 X_3 X_6 + X_0 X_1 X_3 X_7 + X_0 X_1 X_4 X_7 + X_0 X_2 X_4 X_7 +$$
$$X_0 X_2 X_5 X_6 + X_1 X_3 X_6 X_8 + X_1 X_3 X_6 X_9 + X_1 X_3 X_7 X_9 +$$
$$X_1 X_4 X_7 X_9 + X_2 X_4 X_7 X_9 + X_2 X_5 X_6 X_8 + X_2 X_5 X_6 X_9$$

Is it easy to count results of ACQ?

$$\varphi(v, x_1, x_2, w) \equiv E(v, x_1) \wedge E(x_1, x_2) \wedge E(x_2, w)$$

To be evaluated on the following structure $\mathcal{A} = (V, E)$:



One associates the polynomial $Q(\Phi)(X_0, ..., X_9)$ below :

$$X_0^4 + X_0^3 X_1 + X_0^3 X_2 +$$
$$X_0^2 X_1 X_3 + X_0^2 X_1 X_4 + X_0^2 X_2 X_4 + X_0^2 X_2 X_5 +$$
$$X_0 X_1 X_3 X_6 + X_0 X_1 X_3 X_7 + X_0 X_1 X_4 X_7 + X_0 X_2 X_4 X_7 +$$
$$X_0 X_2 X_5 X_6 + X_1 X_3 X_6 X_8 + X_1 X_3 X_6 X_9 + X_1 X_3 X_7 X_9 +$$
$$X_1 X_4 X_7 X_9 + X_2 X_4 X_7 X_9 + X_2 X_5 X_6 X_8 + X_2 X_5 X_6 X_9$$

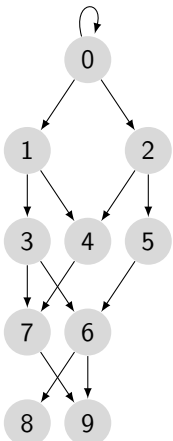
It holds that : $Q(\Phi)(1, ..., 1) = |\varphi(\mathcal{A})|$

Nice but too long... needs to know all the solutions

Is it easy to count results of ACQ?

$$\varphi(v, x_1, x_2, w) \equiv E(v, x_1) \wedge E(x_1, x_2) \wedge E(x_2, w)$$

To be evaluated on the following structure $\mathcal{A} = (V, E)$:



One associates the polynomial $Q(\Phi)(X_0, ..., X_9)$ below :

$$X_0^4 + X_0^3 X_1 + X_0^3 X_2 +$$
$$X_0^2 X_1 X_3 + X_0^2 X_1 X_4 + X_0^2 X_2 X_4 + X_0^2 X_2 X_5 +$$
$$X_0 X_1 X_3 X_6 + X_0 X_1 X_3 X_7 + X_0 X_1 X_4 X_7 + X_0 X_2 X_4 X_7 +$$
$$X_0 X_2 X_5 X_6 + X_1 X_3 X_6 X_8 + X_1 X_3 X_6 X_9 + X_1 X_3 X_7 X_9 +$$
$$X_1 X_4 X_7 X_9 + X_2 X_4 X_7 X_9 + X_2 X_5 X_6 X_8 + X_2 X_5 X_6 X_9$$

It holds that : $Q(\Phi)(1, ..., 1) = |\varphi(\mathcal{A})|$

Nice but too long... needs to know all the solutions

Example

But $Q(\Phi)(X_0, ..., X_9)$ can be factorized knowing φ and $\mathcal{A}$:

$$\begin{aligned} Q(\Phi)(X_0, ..., X_9) &= X_0^4 + X_0^3 X_1 + X_0^3 X_2 + \\ & X_0^2 X_1 X_3 + X_0^2 X_1 X_4 + X_0^2 X_2 X_4 + X_0^2 X_2 X_5 + \\ & X_0 X_1 X_3 X_6 + X_0 X_1 X_3 X_7 + X_0 X_1 X_4 X_7 + X_0 X_2 X_4 X_7 + \\ & X_0 X_2 X_5 X_6 + X_1 X_3 X_6 X_8 + X_1 X_3 X_6 X_9 + X_1 X_3 X_7 X_9 + \\ & X_1 X_4 X_7 X_9 + X_2 X_4 X_7 X_9 + X_2 X_5 X_6 X_8 + X_2 X_5 X_6 X_9 \\ &= \sum_{(i,j)\in E} X_i X_j \left(\sum_{k:(j,k)\in E} X_k \left(\sum_{h:(h,i)\in E} X_h\right)\right) \end{aligned}$$

- This polynomial can be succinctly represented (by this expression i.e. an arithmetic circuit). Then...
- It can be easily evaluated on any (reasonable) set of points.

Arithmetization of queries

Given $\Phi = (\mathcal{A}, \varphi)$, the polynomial $Q(\Phi)$ is defined as :

$$Q(\Phi)(X_1, .., X_n) := \sum_{\mathbf{a} \in \phi(\mathcal{A})} \prod_{i=1}^{k} X_{a_i}.$$

where $n = |A|$, size of the domain of the database.

- Counting : evaluating $Q(\Phi)(1, .., 1)$
- Weighted counting by evaluating on particular values
- Weights can be put directly on tuples too.

But...

- Evaluation is feasible when $Q(\Phi)$ admits a succint representation...

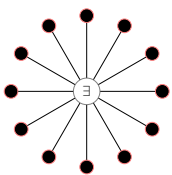
So...

- Which queries have succintly representable Q-polynomials?

What about counting for ACQ?

- $\sharp\mathrm{CQ}$ is $\sharp\cdot$NP-complete Bauland et al'05
- $\sharp\mathrm{ACQ}$: $\sharp$P-complete Pichler-Skritek'13
- Projection free $\sharp\mathrm{ACQ}$: $O(\|\varphi\| \cdot \|\mathbf{D}\|^2)$

There exists an ACQ of the following quantifier form that capture the complexity of counting perfect matchings in a graph:



Quantified star size

- Measure that refines free-connexity: star size
- $\sharp\mathrm{ACQ}$ in polynomial time for every fixed value of this parameter D-Mengel'13->
- Also for more general criterion than acyclicity

Conclusion

Conclusion

A brief introductory tour in counting and enumeration

- Counting and enumeration are natural algorithmic tasks
- Counting is a well established field of complexty
- Enumeration (i.e. generation of solutions) is more recent and has deserved many studies in the context of graph algorithms and data management
- Many algorithms but very few notions of complexity (no reduction...)
- Alternative notions of tractability are still under definition