Query Evaluation: Basics and Recent Developments

Miika Hannula
University of Tartu

December 16, 2024

Short bio:

- PhD (2015) from the University of Helsinki in mathematical logic
- Research during PhD/postdoc:
  - Dependence logic $\forall, \exists, \wedge, \vee, \neg, =\!(x, y)$
  - Implication problem $\Sigma \models \tau$? for database dependencies
- Since 2024 assoc. prof. in data management at Tartu

This talk: General overview[1] of one of the most fundamental problems in database theory:

- Query evaluation

[1] Main source: [Arenas et al., 2022]

Introduction

Complexity of CQ-Evaluation

Joins with Information Theory

Conclusion

Output of a query

```
SELECT e.emp_id, e.emp_name
FROM Employees AS e, Departments AS d
WHERE e.dep_id = d.dep_id AND d.dep_name = 'Sales';
```

Employees

emp_id	emp_name	dep_id
12345	Alice	10
67890	Bob	20
23456	Charlie	10

Departments

dep_id	dep_name
10	Sales
20	Engineering

$\longrightarrow$

Output

emp_id	emp_name
12345	Alice
23456	Charlie

Notation $q(D)$ = output of query q on database D

Problem:	Query-Evaluation
Input:	A query q, a database D, a tuple of values $\bar{a}$
Output:	`true` if $\bar{a} \in q(D)$, and `false` otherwise

Previous example:

Input:

- query q = given SQL query
- database D = given database
- tuple $\bar{a}$ = (Alice, Sales)

Output: `true` / (12345, Alice)$\in q(D)$

Notation $q(D)$ = output of query q on database D

Problem:	Query-Evaluation
Input:	A query q, a database D, a tuple of values $\bar{a}$
Output:	`true` if $\bar{a} \in q(D)$, and `false` otherwise

Previous example:

Input:

- query q = given SQL query
- database D = given database
- tuple $\bar{a}$ = (Alice, Sales)

Output: `true` / (12345, Alice)$\in q(D)$

Mathematics behind

How to analyse the complexity of Query-Evaluation?

We need a mathematical description of a

- (relational) database
- (SQL) query
- (data tuple)

Database

Employees

emp_id	emp_name	dep_id
12345	Alice	10
67890	Bob	20
23456	Charlie	10

Departments

dep_id	dep_name
10	Sales
20	Engineering

- Each entry viewed as a fact, i.e., an expression such as Employees(12345, Alice, 10)
  - let's use here a shorthand: Emp(12345, Alice, 10)
- A database D defined as a finite set of facts:

  {Emp(12345, Alice, 10), Emp(67890, Bob, 20), Emp(23456, Charlie, 10), Dep(10, Sales), Dep(20, Engineering)}

- D is a database of a schema $\mathbf{S} = \{E[3], D[2]\}$ specifying its structure

Query

A query q over schema **S** is a function that maps databases D of **S** to finite sets of sequences (of the same length)

$$q(D) = \{(a_1, \ldots, a_k), (b_1, \ldots, b_k), \ldots\}$$

Example

In our example, $q(D) = \{(12345, \text{Alice}), (23456, \text{Charlie})\}$

Such queries can be described using query languages. Two paradigms:

- Declarative languages (logic)
- Procedural languages (algebra)

Queries and logic

To simplify analysis, let us restrict attention to the so-called "Core SQL", formed using only commands `SELECT`, `FROM`, `WHERE` with equality comparisons

vs.

First-order logic (FO):

- Atomic formulas $R(x, y), x = y, \ldots$
- Connectives $\wedge, \vee, \neg$
- Quantifiers $\exists, \forall$

Queries and logic

To simplify analysis, let us restrict attention to the so-called "Core SQL", formed using only commands `SELECT`, `FROM`, `WHERE` with equality comparisons

$=$

Conjunctive query (CQ):

- Atomic formulas $R(x, y)$, ~~$x = y, \ldots$~~
- Connectives $\wedge$, ~~$\vee, \neg$~~
- Quantifiers $\exists$, ~~$\forall$~~

Queries and logic

```
SELECT e.emp_id, e.emp_name
FROM Employees AS e, Departments AS d
WHERE e.dep_id = d.dep_id AND d.dep_name = 'Sales';
```

. . . corresponds to the CQ . . .

$$\phi(x, y) := \exists z \exists w(\mathrm{Emp}(x, y, z) \wedge \mathrm{Dep}(z, \text{'Sales'}))$$

where

- Variables x, y are free and correspond to the output
- Variables z is bound by $\exists$; 'Sales' called a constant

Queries and logic

```
SELECT e.emp_id, e.emp_name
FROM Employees AS e, Departments AS d
WHERE e.dep_id = d.dep_id AND d.dep_name = 'Sales';
```

. . . corresponds to the CQ . . .

$$\text{Answer}(x, y) :- \text{Emp}(x, y, z), \text{Dep}(z, \text{'Sales'})$$

where

- $\text{Emp}(x, y, z)$, $\text{Dep}(z, \text{'Sales'})$ forms the body
- $\text{Answer}(x, y)$ forms the head

CQ Semantics

Given a database

$$D = \{\text{Emp}(12345,\ \text{Alice},\ 10),\ \text{Emp}(67890,\ \text{Bob},\ 20),\ \text{Emp}(23456,\ \text{Charlie},\ 10),\ \text{Dep}(10,\ \text{Sales}),\ \text{Dep}(20,\ \text{Engineering})\}$$

and a query

$$q = \text{Answer}(x, y) \coloneq \text{Emp}(x, y, z), \text{Dep}(z, \text{'Sales'})$$

we define the output $q(D)$ as the set of all pairs $(h(x), h(y))$, where

- h is a mapping from variables to constants, and
- $\text{Emp}(h(x), h(y), h(z))$ and $\text{Dep}(h(z), \text{'Sales'})$ are in D

This can be represented...

```
SELECT e.emp_id, e.emp_name
FROM Employees AS e, Departments AS d
WHERE e.dep_id = d.dep_id AND e.dep_name = 'Sales';
```

Employees

emp_id	emp_name	dep_id
12345	Alice	10
67890	Bob	20
23456	Charlie	10

Departments

dep_id	dep_name
10	Sales
20	Engineering

$\longrightarrow$

Output

emp_id	emp_name
12345	Alice
23456	Charlie

...as this

- query $q = \text{Answer}(x, y) \coloneq \text{Emp}(x, y, z), \text{Dep}(z, \text{'Sales'})$
- database
  $D = \{$Emp(12345, Alice, 10), Emp(67890, Bob, 20), Emp(23456, Charlie, 10), Dep(10, Sales), Dep(20, Engineering)$\}$
- output $q(D) = \{(12345, \text{Alice}), (23456, \text{Charlie})\}$

Query-Evaluation revisited

Problem:	Query-Evaluation
Input:	A query q, a database D and a tuple of values $\bar{a}$
Output:	`true` if $\bar{a} \in q(D)$, and `false` otherwise

Our example:

Input:

- query $q = \text{Answer}(x, y) \coloneq \text{Emp}(x, y, z), \text{Dep}(z, \text{'Sales'})$
- database
  $D = \{$Emp(12345, Alice, 10), Emp(67890, Bob, 20), Emp(23456, Charlie, 10), Dep(10, Sales), Dep(20, Engineering)$\}$
- tuple $\bar{a} = (12345, \text{Alice})$

Output: `true`

CQ-Evaluation

Problem:	CQ-Evaluation
Input:	A Boolean conjunctive query q, a database D
Output:	`true` if D satisfies q, and `false` otherwise

Our example:

Input:

- query $q = \text{Answer} \coloneq \text{Emp}(12345, \text{'Alice'}, z), \text{Dep}(z, \text{'Sales'})$
- database
  $D = \{\text{Emp}(12345,\ \text{Alice},\ 10),\ \text{Emp}(67890,\ \text{Bob},\ 20),\ \text{Emp}(23456,\ \text{Charlie},\ 10),$
  $\text{Dep}(10,\ \text{Sales}),\ \text{Dep}(20,\ \text{Engineering})\}$

Output: `true`

Introduction

Complexity of CQ-Evaluation

Joins with Information Theory

Conclusion

Theorem

CQ-Evaluation is NP*-complete.*

Proof.

CQ-Evaluation is in NP: Guess the function h from variables to constants and check that the body of q is mapped into D.

CQ-Evaluation is in NP-hard: Next page. □

Hardness: Reduction from Clique to CQ-Evaluation

Clique:

- Given a natural number k and an undirected graph G with vertex set V and edge set E (without self-loops $\{u, u\}$), decide if G has a clique of size k.
- NP-complete

CQ-Evaluation:

- Construct a database D as follows:

$$D = \{\text{Node}(v) \mid v \in V\} \cup \{\text{Edge}(u, v) \mid \{u, v\} \in E\}$$

$$q = \exists x_1 \ldots \exists x_k \left(\bigwedge_{i=1}^{k} \text{Node}(x_i) \wedge \bigwedge_{\substack{i,j \in [k] \\ i \neq j}} \text{Edge}(x_i, x_j) \right)$$

$\Longrightarrow$ D satisfies q if and only if G contains a clique of size k

Hardness analysed

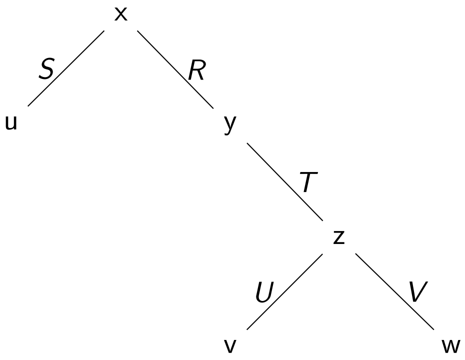
- Hardness of CQ-Evaluation can arise from queries shaped as **cliques**
- Such queries not common / typically queries shaped as **trees**

Hardness analysed cont.

Consider

$$\text{Answer} \coloneq S(u,x), R(x,y), T(y,z), U(z,v), V(z,w)$$

having shape:



Semi-Join

For two relations R and S the semi-join $R \ltimes S$ returns all rows from R that have matching rows in S

Example

R	x	y
	1	a
	2	b
	3	c
	4	d

$\ltimes$

T	y	z
	a	10
	a	10
	c	20
	e	30

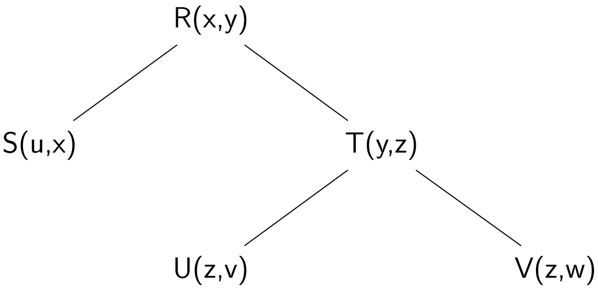
$=$

	x	y
	1	a
	3	c

Evaluating tree queries

Computation of Answer:

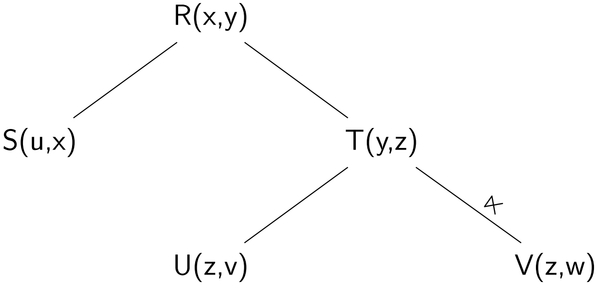
1. Take the line graph of prev. graph



Evaluating tree queries

Computation of Answer:

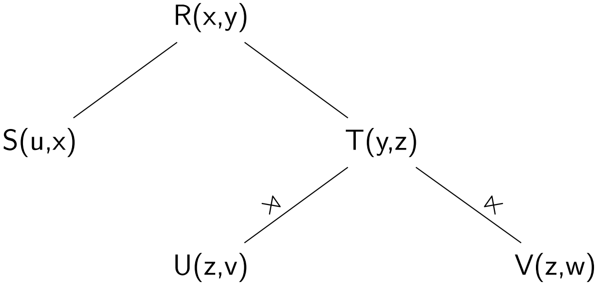
1. Take the line graph of prev. graph
2. $T[y,z] := T[y,z] \ltimes V[z,w]$



Evaluating tree queries

Computation of Answer:

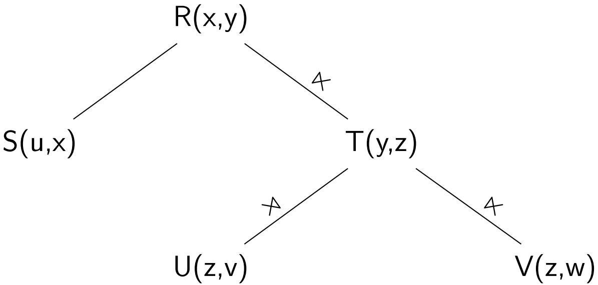
1. Take the line graph of prev. graph
2. $T[y,z] := T[y,z] \ltimes V[z,w]$
3. $T[y,z] := T[y,z] \ltimes U[z,v]$



Evaluating tree queries

Computation of Answer:

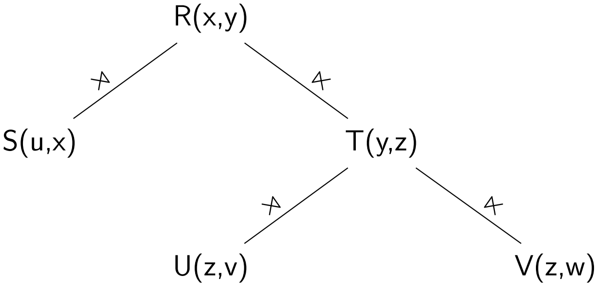
1. Take the line graph of prev. graph
2. $T[y,z] := T[y,z] \ltimes V[z,w]$
3. $T[y,z] := T[y,z] \ltimes U[z,v]$
4. $R[x,y] := R[x,y] \ltimes T[y,z]$



Evaluating tree queries

Computation of Answer:

1. Take the line graph of prev. graph
2. $T[y,z] := T[y,z] \ltimes V[z,w]$
3. $T[y,z] := T[y,z] \ltimes U[z,v]$
4. $R[x,y] := R[x,y] \ltimes T[y,z]$
5. $R[x,y] := R[x,y] \ltimes S[u,x]$



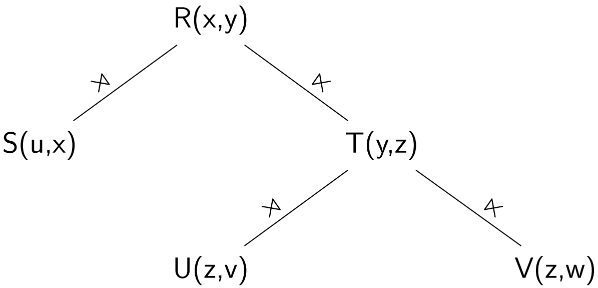
Evaluating tree queries

Computation of Answer:

1. Take the line graph of prev. graph
2. $T[y,z] := T[y,z] \ltimes V[z,w]$
3. $T[y,z] := T[y,z] \ltimes U[z,v]$
4. $R[x,y] := R[x,y] \ltimes T[y,z]$
5. $R[x,y] := R[x,y] \ltimes S[u,x]$

$\rightarrow$ Answer is `true` iff R is non-empty in the end

Time complexity:
$O(||D|| \cdot \log ||D|| \cdot ||q||)$



Yannakakis

- Previous algorithm known as the Yannakakis algorithm [Yannakakis, 1981]
- Follows a bottom-up dynamic programming approach

But, we can do better (Yannakakis algorithm is actually more general, as we will see):

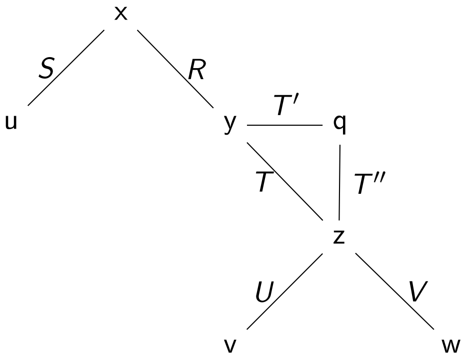
- What if q contains only **small** cliques?

Small clique

Consider

$$\text{Answer} :- S(u,x), R(x,y), T(y,z), T'(y,q), T''(q,z), U(z,v), V(z,w)$$

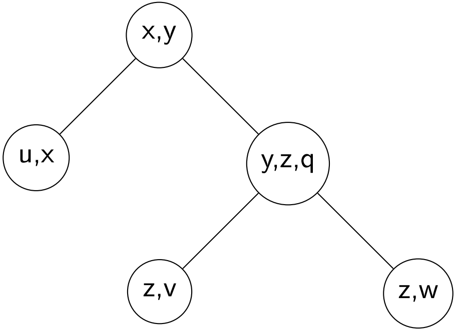
having shape:



Small clique re-organised

Atoms are grouped into nodes:

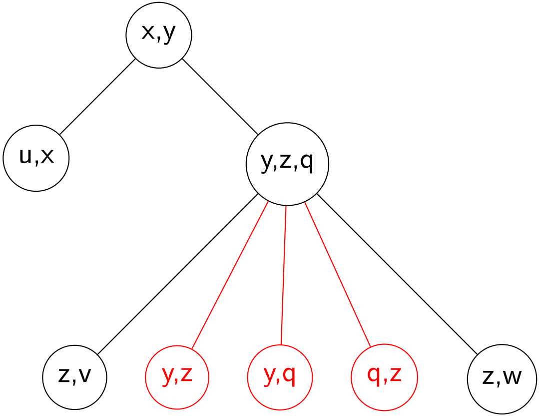
- $S(u, x)$
- $R(x, y)$
- $T(y, z)$, $T'(y, q)$, $T''(q, z)$
- $U(z, v)$
- $V(z, w)$



Small clique further re-organised

Add 3 new children for the clique
$\rightarrow$

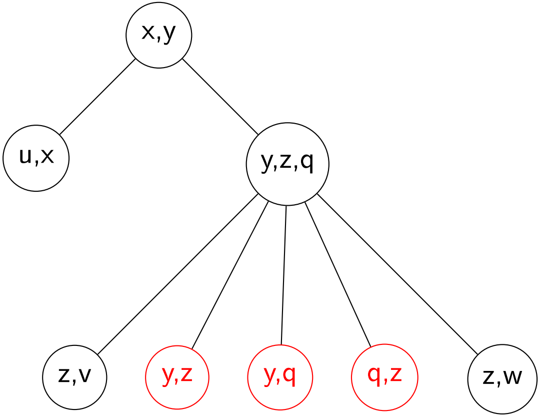
- $S(u, x)$
- $R(x, y)$
- $T(y, z)$, $T'(y, q)$, $T''(q, z)$
- $U(z, v)$
- $V(z, w)$



Computation of answer

adom(D) = the set of all values appearing in D

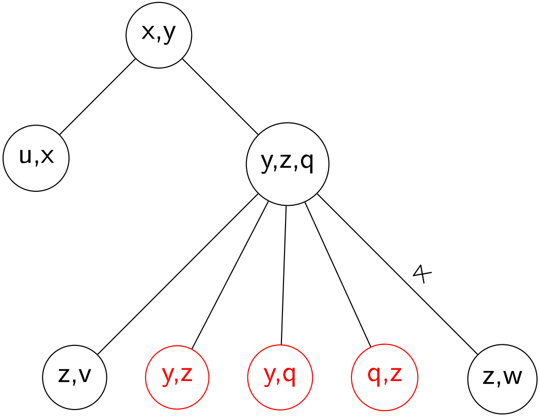
1. $A[y, z, q] := \text{adom}(D)^3$



Computation of answer

adom(D) = the set of all values appearing in D

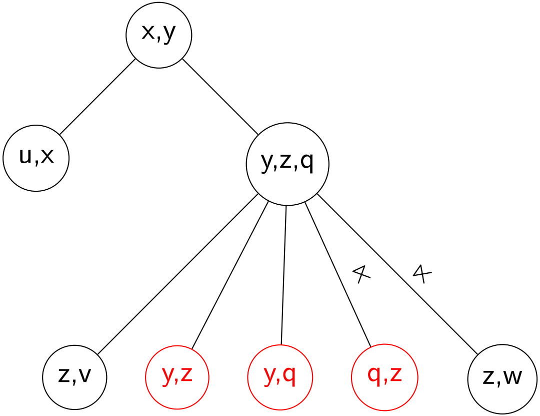
1. $A[y, z, q] := \text{adom}(D)^3$
2. $X[y, z, q] := X[y, z, q] \ltimes V[z, w]$



Computation of answer

adom(D) = the set of all values appearing in D

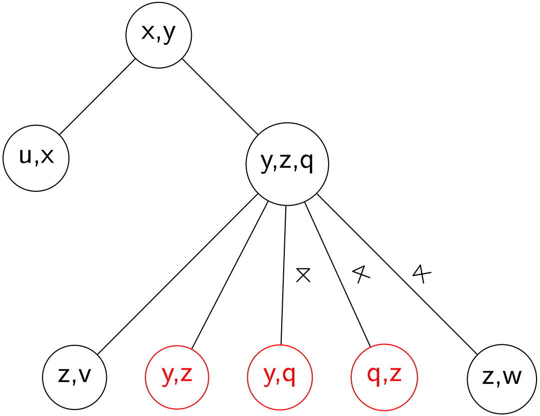
1. $A[y, z, q] := \text{adom}(D)^3$
2. $X[y, z, q] := X[y, z, q] \ltimes V[z, w]$
3. $X[y, z, q] := X[y, z, q] \ltimes T''[q, z]$



Computation of answer

adom(D) = the set of all values appearing in D

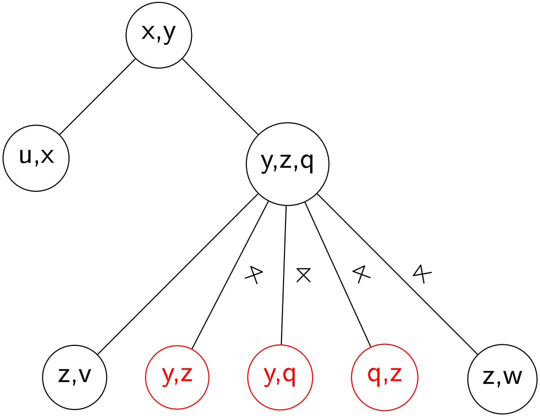
1. $A[y,z,q] := \text{adom}(D)^3$
2. $X[y,z,q] := X[y,z,q] \ltimes V[z,w]$
3. $X[y,z,q] := X[y,z,q] \ltimes T''[q,z]$
4. $X[y,z,q] := X[y,z,q] \ltimes T'[y,q]$



Computation of answer

adom(D) = the set of all values appearing in D

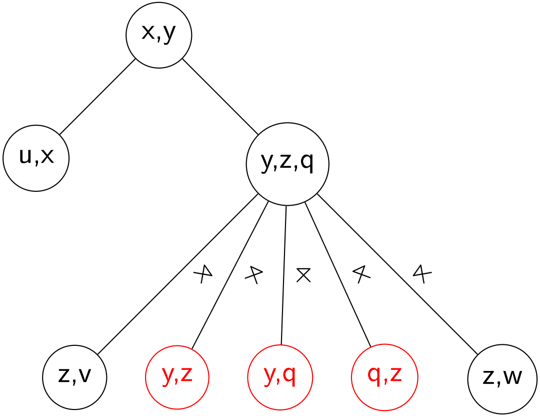
1. $A[y,z,q] \coloneqq \text{adom}(D)^3$
2. $X[y,z,q] \coloneqq X[y,z,q] \ltimes V[z,w]$
3. $X[y,z,q] \coloneqq X[y,z,q] \ltimes T''[q,z]$
4. $X[y,z,q] \coloneqq X[y,z,q] \ltimes T'[y,q]$
5. $X[y,z,q] \coloneqq X[y,z,q] \ltimes T[y,z]$



Computation of answer

adom(D) = the set of all values appearing in D

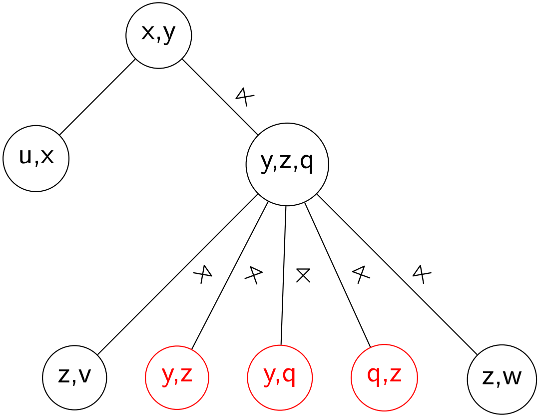
1. $A[y,z,q] := \text{adom}(D)^3$
2. $X[y,z,q] := X[y,z,q] \ltimes V[z,w]$
3. $X[y,z,q] := X[y,z,q] \ltimes T''[q,z]$
4. $X[y,z,q] := X[y,z,q] \ltimes T'[y,q]$
5. $X[y,z,q] := X[y,z,q] \ltimes T[y,z]$
6. $X[y,z,q] := X[y,z,q] \ltimes U[z,v]$



Computation of answer

adom(D) = the set of all values appearing in D

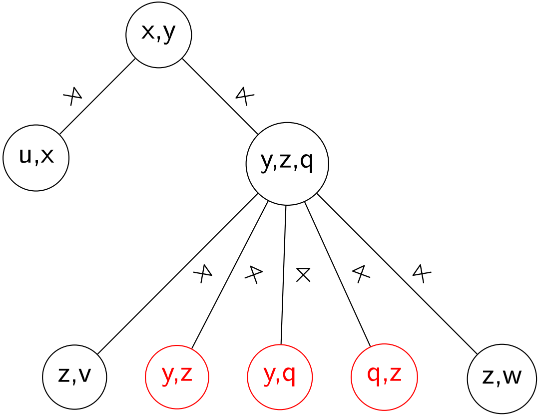
1. $A[y, z, q] := \text{adom}(D)^3$
2. $X[y, z, q] := X[y, z, q] \ltimes V[z, w]$
3. $X[y, z, q] := X[y, z, q] \ltimes T''[q, z]$
4. $X[y, z, q] := X[y, z, q] \ltimes T'[y, q]$
5. $X[y, z, q] := X[y, z, q] \ltimes T[y, z]$
6. $X[y, z, q] := X[y, z, q] \ltimes U[z, v]$
7. $R[x, y] := R[x, y] \ltimes X[y, z, q]$



Computation of answer

adom(D) = the set of all values appearing in D

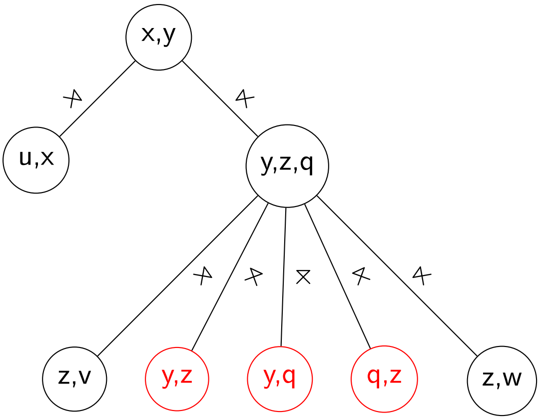
1. $A[y, z, q] := \text{adom}(D)^3$
2. $X[y, z, q] := X[y, z, q] \ltimes V[z, w]$
3. $X[y, z, q] := X[y, z, q] \ltimes T''[q, z]$
4. $X[y, z, q] := X[y, z, q] \ltimes T'[y, q]$
5. $X[y, z, q] := X[y, z, q] \ltimes T[y, z]$
6. $X[y, z, q] := X[y, z, q] \ltimes U[z, v]$
7. $R[x, y] := R[x, y] \ltimes X[y, z, q]$
8. $R[x, y] := R[x, y] \ltimes S[u, x]$



Computation of answer

1. $A[y, z, q] := \text{adom}(D)^3$
2. $X[y, z, q] := X[y, z, q] \ltimes V[z, w]$
3. $X[y, z, q] := X[y, z, q] \ltimes T''[q, z]$
4. $X[y, z, q] := X[y, z, q] \ltimes T'[y, q]$
5. $X[y, z, q] := X[y, z, q] \ltimes T[y, z]$
6. $X[y, z, q] := X[y, z, q] \ltimes U[z, v]$
7. $R[x, y] := R[x, y] \ltimes X[y, z, q]$
8. $R[x, y] := R[x, y] \ltimes S[u, x]$

Answer = `true` iff R non-empty in the end. Time complexity (here) $O((||q|| + 1)(||D||^3 \log ||D||^3))$



CQ as a hypergraph

Let's generalise this idea:

- Queries can have more than two terms in an atom $\rightarrow$ hypergraphs
- Hypergraphs can be re-structured as a tree that group $\leq k$ variables (leading to the $||D||^k$-factor in time complexity) $\rightarrow$ treewidth to measure growth of complexity

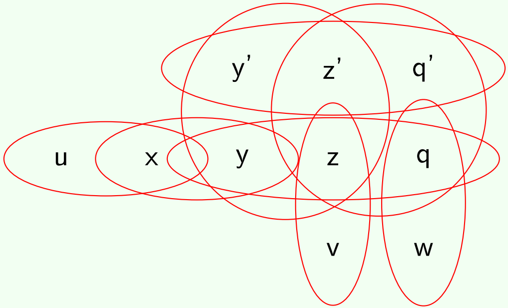
Definition

A hypergraph is a pair $H = (V, E)$ consisting of a set V of nodes and a set E of subsets of V, called hyperedges.

Example

$$\text{Answer} :- S(u,x), R(x,y), T(y,z,q), U(z,v), V(z,w),$$
$$A(y',z',q'), B(y,z,y',z'), C(z,q,z',q')$$

as a hypergraph:



Treewidth

Definition

A tree decomposition of a hypergraph $G = (V, E)$ is a tree $T = (B, E_T)$, where the vertex set $B \subseteq \mathcal{P}(V)$ is a collection of bags and E_T is a set of edges as follows:

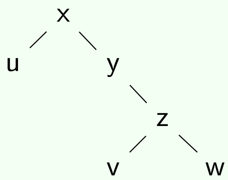
1. $\bigcup_{b \in B} b = V$,
2. for every $e \in E$ there is a bag $b \in B$ with $e \subseteq b$, and
3. for all $v \in V$ the subtree of T induced by the bags containing v is connected.

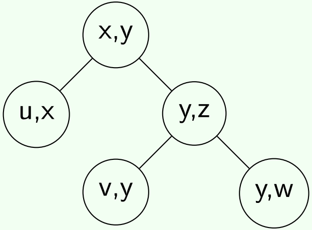
The width of the tree decomposition T is the size of the largest bag decreased by one: $\max_{b \in B} |b| - 1$. The treewidth of G is the minimum width over all tree decompositions of G.

G is tree iff it has treewidth 1

Example

Left: our first "tree query". Right: its tree decomposition of width 1.

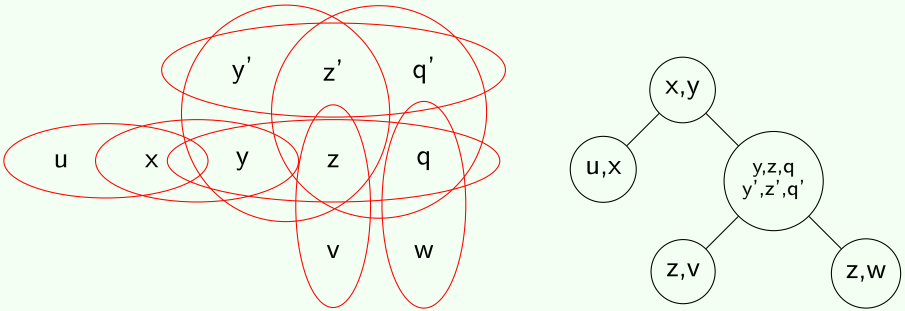




Last example

Example

Left: our last example query. Right: its tree decomposition of width 5.



CQ-Evaluation for fixed treewidth

Theorem

Fix $k \geq 1$. Then CQ-Evaluation, restricted to queries with treewidth at most k, can be solved in time

$$O(||D||^{k+1} \cdot ||q||^4 \cdot (\log ||D|| + \log ||q||)).$$

Key pointers:

- Tree decomposition can be constructed in time $2^{O(k^3)}||G||$ for a (hyper)graph G with treewidth k [Bodlaender, 1996] $\implies$ linear time for fixed k
- Treewidth = largest bag size - 1 $\implies$ $||D||^{k+1}$-factor

Happy?

Not happy. Efficient evaluation can be possible even with unbounded treewidth

Example

Consider "n-clique"+"one giant atom":

$$\text{Answer} :- R(x_i, x_j)_{\substack{i,j\in[n]\\ i\neq j}}, S(x_1, \ldots, x_n)$$

Treewidth is $n-1$, yet the following procedure is efficient:

- Step 1. $S[x_1, \ldots, x_n] := S[x_1, \ldots, x_n] \ltimes R[x_1, x_1]$
- Step 2. $S[x_1, \ldots, x_n] := S[x_1, \ldots, x_n] \ltimes R[x_1, x_2]$
- ...
- Step n^2. $S[x_1, \ldots, x_n] := S[x_1, \ldots, x_n] \ltimes R[x_n, x_n]$

Yannakakis revisited

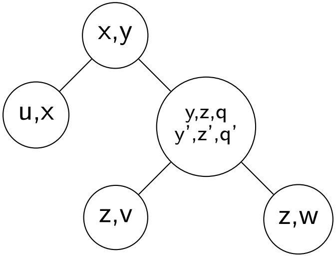
The max size of a bag (in a tree decomposition) is not crucial; the number of hyperedges covering a bag is. Bottom right: tree decomposition of our example query

$$\text{Answer} :- S(u,x), R(x,y), T(y,z,q), U(z,v), V(z,w),$$
$$A(y',z',q'), B(y,z,y',z'), C(z,q,z',q')$$

Computation of Answer

1. $X[y,z,q,y',z',q'] := B \bowtie C$
2. $X[y,z,q,y',z',q'] := X \ltimes V$
3. ...

First step yields $||D||^2$-factor since the big bag is covered by 2 atoms B and C



Treewidth revisited

Definition (Generalised hypertreewidth [Gottlob et al., 2003])

A generalised hypertree decomposition of a hypergraph $H = (V, E)$ is a triple (B, E_T, λ), where

1. (B, E_T) is a tree decomposition of H.
2. λ is a mapping that assigns a subset of E to each bag $b \in B$.
3. For each bag $b \in B$, $b \subseteq \bigcup_{e \in \lambda(b)} e$.

The width of a hypertree is the cardinality of its largest λ-label, i.e., $\max_{b \in B} |\lambda(b)|$.
The generalised hypertreewidth of H is the minimum over over widths of generalised hypertree decompositions of H.

CQ-Evaluation for fixed generalised hypertreewidth

Theorem

Fix $k \geq 1$. Then CQ-Evaluation, restricted to queries of generalised hypertreewidth at most k, can be solved in time

$$2^{||q||^c} + O(||D||^k \cdot ||q||), \quad \textit{for some integer } c \geq 1.$$

Key pointers:

- Finding generalised hypertree decompositions is generally hard. $2^{||q||^c}$ accounts for what is essentially a brute-force construction.
- Generalised hypertreewidth = largest bag cover $\implies$ $||D||^k$-factor

Generalised hypertreewidth can be further generalised with the notion a fractional hypertreewidth [Grohe and Marx, 2006], but let's move on...

Introduction

Complexity of CQ-Evaluation

Joins with Information Theory

Conclusion

Efficient CQ-Evaluation: Recap

Recall CQ-Evaluation has two-partite input: (q, D)

Structural properties of q:

- Treewidth
- Generalised hypertreewidth

...lead to efficient query evaluation.

Properties of D?

- Cardinalities of relations?
- Constraints on relations?

Can we combine information from q **and** D to obtain efficient algorithms?

Target

Let us consider again the case where the CQ q is non-Boolean, i.e., the output $q(D)$ is a relation

Target: Devise efficient algorithms for computing $q(D)$

Strategy:

1. Given structural properties of q and information about D, determine the worst-case size of the output $q(D)$.
2. Devise algorithms that run in time proportional to this worst-case output size.

Join queries

We will consider join queries q.

The (natural) join of R_1 and R_2, denoted $R_1 \bowtie R_2$, is the set of tuples that are formed by combining tuples from R_1 and R_2 which agree on their common attributes.
"Formally":

$$R \bowtie S = \{t \mid t[\text{att}(R)] \in R, t[\text{att}(S)] \in S\},$$

where $t[A]$ is the projection of a tuple t on an attribute set A.

Note: Join is associative and commutative

3-way join example

Example

Input Tables:

$R[A, B] =$

A	B
1	10
2	20
3	30

$S[A, C] =$

A	C
1	100
2	200
4	400

$T[B, C] =$

B	C
10	100
20	300
30	300

Result of the Join:

$R[A, B] \bowtie S[A, C] \bowtie T[B, C] =$

A	B	C
1	10	100

Joins vs. CQs

A join query $R_1 \bowtie \cdots \bowtie R_n$ can be viewed as a CQ of the form

$$\text{Answer}(\vec{x}) \coloneq R_1(\vec{y}_1), \ldots, R_n(\vec{y}_n)$$

in which:

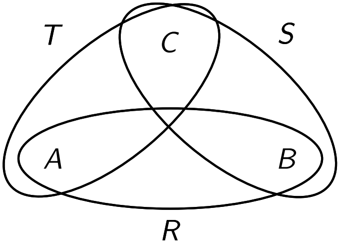
1. Each relation name R_i occurs exactly once.
2. For every $i \in [n]$, no variables are repeated in $\vec{y}_i$.
3. Every variable in $\vec{y}_i$ also appears in $\vec{x}$.

Example: "Triangle Query" $q_\triangle$

Consider the join query:

$$q_\triangle = R[A, B] \bowtie S[B, C] \bowtie T[C, A]$$

The hypergraph of $q_\triangle$ visualizes its structure:



Question: How many tuples can there be in $q_\triangle(D)$, where D is a database?

Evaluating $q_\triangle(D)$

Assume R, S, T each have N tuples

Trivially, $|q_\triangle(D)| \leq N^3$ (size of Cartesian product $|R| \cdot |S| \cdot |T|$)

However, $|q_\triangle(D)| \leq N^2$ due to attribute constraints. Consider the following computation:

1. Compute $R \bowtie S$, selecting tuples matching on B.
   - Output size at most $|R \times S| \leq N^2$
2. Join the result with T, selecting tuples matching on A and C.
   - Can only remove tuples from the previous join
3. $\implies$ Final output size at most $|R \times S| \leq N^2$.

Our naïve analysis yields:

$$|q_{\triangle}(D)| \leq N^2$$

The so-called AGM bound [Atserias et al., 2013] yields:

$$|q_{\triangle}(D)| \leq N^{3/2} \qquad (1)$$

Next: Derivation of (1)

Entropy

(Shannon) entropy of a random variable X with a finite domain D and a probability mass function p:

$$H(X) := -\sum_{x \in D} p(x) \log p(x).$$

Useful bounds: $0 \leq H(X) \leq \log |D|$

Entropic function $h(\boldsymbol{X}_\alpha) := H(\boldsymbol{X}_\alpha)$ $(\alpha \subseteq [n])$ for Shannon entropies H arising from marginals of some joint distribution of random variables $X_1, \ldots, X_n$

Entropic region Γ_n^*: the subset of $\mathbb{R}^{2^n}$ consisting of the entropic functions/vectors over n random variables

Laws of information

Example: conditioning decreases entropy

$$h(\boldsymbol{X} \mid \boldsymbol{Y}) = h(\boldsymbol{X}\boldsymbol{Y}) - h(\boldsymbol{Y}) \leq h(\boldsymbol{X})$$

Polymatroid axioms:

- $h(\emptyset) = 0$
- $h(\boldsymbol{X}) \leq h(\boldsymbol{X}\boldsymbol{Y})$ (monotonicity)
- $h(\boldsymbol{X}) + h(\boldsymbol{XYZ}) \leq h(\boldsymbol{X}\boldsymbol{Y}) + h(\boldsymbol{X}\boldsymbol{Z})$ (submodularity)

Polymatroid axioms sound but incomplete [Zhang and Yeung, 1998]; there is no finite axiomatic characterisation for entropic functions [Matús, 2007]

Derivation of AGM bound I

Instance of **Shearer's Lemma**:

$$
\begin{aligned}
& \tfrac{1}{2}\big(h(XY) + && h(XZ) + && h(YZ)\big) \\
= \; & \tfrac{1}{2}\big(h(X) + h(Y \mid X) + && h(X) + h(Z \mid X) + && h(Y) + h(Z \mid Y)\big) \\
\geq \; & \tfrac{1}{2}\big(h(X) + h(Y \mid X) + && h(X) + h(Z \mid XY) + && h(Y \mid X) + h(Z \mid XY)\big) \\
= \; & h(X) + h(Y \mid X) + h(Z \mid XY) \\
= \; & h(XYZ)
\end{aligned}
$$

Derivation of AGM bound II

Consider $q_\triangle = R[A, B] \bowtie S[B, C] \bowtie T[C, A]$ and a database $D = \{R, S, T\}$ where each relation at most of size N

If h is the entropic function of the output $q_\triangle(D)$ uniformly distributed:

$$\begin{aligned}\log|q_\triangle(D)| = h(ABC) &\leq \frac{1}{2}\big(h(AB) + h(AC) + h(BC)\big)\\ &\leq \frac{1}{2}\big(\log|R| + \log|S| + \log|T|\big)\\ &\leq \frac{3}{2}\log N\end{aligned}$$

Fractional Edge Cover

Definition

A fractional edge cover of a hypergraph $H = (V, E)$ is a function

$$f : E \to \mathbb{Q}_{\geq 0}$$

such that, for each node $v \in V$, it holds that

$$\sum_{v \in e} f(e) \geq 1.$$

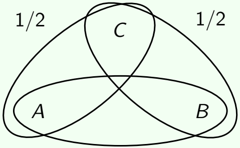
The cover f is called minimal when it minimises the weight

$$\sum_{e \in E} f(e).$$

Fractional Edge Cover for Triangle query

Example

Let $f(e) = 1/2$ for all $e \in E$. Then,
$f(e) \geq 0$ for all hyperedges e, and:



$$\sum_{A \in e} f(e) = f(\{A, B\}) + f(\{A, C\}) = 1/2 + 1/2 \geq 1$$

$$\sum_{B \in e} f(e) = f(\{A, B\}) + f(\{B, C\}) = 1/2 + 1/2 \geq 1$$

$$\sum_{C \in e} f(e) = f(\{A, C\}) + f(\{B, C\}) = 1/2 + 1/2 \geq 1$$

Minimality of f

A minimal fractional edge cover for $q_\triangle = R[A, B] \bowtie S[B, C] \bowtie T[C, A]$ can be found by solving the following linear program:

$$\begin{aligned} \text{minimize} \quad & x_R + x_S + x_T \\ \text{subject to} \quad & x_R + x_T \geq 1, \\ & x_R + x_S \geq 1, \\ & x_S + x_T \geq 1, \\ \text{and} \quad & x_R \geq 0, \quad x_S \geq 0, \quad x_T \geq 0. \end{aligned}$$

minimal fractional edge cover = values of x_R, x_S, x_T at the optimal solution

AGM Bound for Join Queries

Theorem (AGM Bound)

Consider a join query $q = R_1 \bowtie \cdots \bowtie R_n$ over schema $\boldsymbol{S}$ and a fractional edge cover f of q. Then, for every database D, we have:

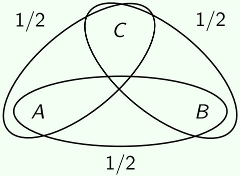
$$|q(D)| \leq \prod_{i=1}^{n} |R_i|^{f(\boldsymbol{S}(R_i))}. \tag{2}$$

If f is a minimal, there are arbitrarily large databases D for which Eq. (2) is an equality,

Optimal AGM bound for $q_\triangle$

Example

Assume $|R| = |S| = |T| = N$. Then:



$$|q_\triangle(D)| \leq \prod_{i=1}^{n} |R_i|^{f(\mathbf{S}(R_i))} = |R|^{1/2} \cdot |S|^{1/2} \cdot |T|^{1/2} = \sqrt{|R| \cdot |S| \cdot |T|} = N\sqrt{N}$$

Inoptimality of binary joins

Suppose each of R, S, T has the form

1	1
1	2
⋮	⋮
1	$(N+1)/2$
2	1
⋮	⋮
$(N+1)/2$	1

} N tuples

- AGM bound $= N\sqrt{N}$
- Any intermediate binary join $R \bowtie S, R \bowtie T, S \bowtie T$ contains more than $(N/2)^2 = \frac{1}{4}N^2$ tuples

Q: Possible to compute $R \bowtie S \bowtie T$ in $O(N\sqrt{N})$?

Attribute-Elimination Join for $R[A, B] \bowtie S[B, C] \bowtie T[A, C]$

1. Compute $L_1 := \pi_A(R \bowtie T)$.
2. For each $a \in L_1$:
   - Compute values $b \in \pi_B(R \bowtie S)$ s.t. $(a, b) \in R$ and $(b, c) \in S$.
   - Add pairs (a, b) to L_2.
3. For each $(a, b) \in L_2$:
   - Compute values $c \in \pi_C(S \bowtie T)$ s.t. $(b, c) \in S$ and $(c, a) \in T$.
   - Add triples (a, b, c) to L_3.
4. Return L_3.

Relations R, S, T:

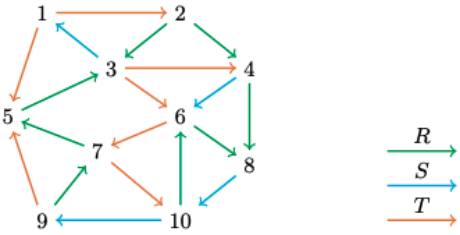


Fig. source: [Arenas et al., 2022]

We obtain:

- $L_1 = \{5, 2, 6, 7, 4, 10\}$
- $L_2 = \{(5, 3), (2, 3), (2, 4), (6, 8), (4, 8)\}$
- $L_3 = \{(5, 3, 1), (2, 3, 1)\}$

Complexity of AEJoin

Theorem

Consider a join query $q = R_1 \bowtie \ldots \bowtie R_n$ *over attributes* $A_1, \ldots, A_m$*. Then the Attribute-Elimination Join algorithm computes the output in time* $\tilde{O}\left(n \cdot m \cdot \prod_{j=1}^{n} |R_j|^{x_j}\right)$*, where* $(x_1, \ldots, x_n)$ *is a fractional edge cover of* q*.*

Note: For a function $f(\vec{x})$, we write $\tilde{O}(f(\vec{x})) = O(f(\vec{x}) \log f(\vec{x}))$

A join algorithm with running time $\tilde{O}(nm|q(D)|)$ is called a worst-case optimal join algorithm.

Efficient CQ-Evaluation: Recap 2

CQ-Evaluation has two-partite input: (q, D). We considered:

Structural properties of q:

- **Fractional edge cover**

Properties of D:

- **Cardinalities of relations**

...to sketch a worst-case optimal join algorithm

Efficient CQ-Evaluation: Recap 2

CQ-Evaluation has two-partite input: (q, D). We considered:

Structural properties of q:

- **Fractional edge cover**

Properties of D:

- **Cardinalities of relations**
- Constraints on relations?

...to sketch a worst-case optimal join algorithm

More input information: constraints

Target: Estimate $|q(D)|$ given

1. join query $q = R_1[\boldsymbol{X}_1] \bowtie \ldots \bowtie R_n[\boldsymbol{X}_n]$
2. degree constraints w.r.t. bounds $\boldsymbol{B}$

Degree constraints

Degree $\deg_R(\boldsymbol{V} \mid \boldsymbol{U} = \boldsymbol{u})$: number of distinct values of $\boldsymbol{V}$ in R under $\boldsymbol{U} = \boldsymbol{u}$

Max-degree $\deg_R(\boldsymbol{V} \mid \boldsymbol{U})$: maximum of degrees $\deg_R(\boldsymbol{V} \mid \boldsymbol{U} = \boldsymbol{u})$ over $\boldsymbol{u}$

- Functional dependencies $\boldsymbol{U} \to \boldsymbol{V}$ definable by $\deg_R(\boldsymbol{V} \mid \boldsymbol{U}) \leq 1$
- Size bounds $|R| \leq B$ definable by $\deg_R(\boldsymbol{U} \mid \emptyset) \leq B$

Degree statistics: Set Σ of conditionals $(\boldsymbol{V} \mid \boldsymbol{U})$. A conditional $(\boldsymbol{V} \mid \boldsymbol{U})$ is guarded by a relation $R[\boldsymbol{X}]$ if $\boldsymbol{U}\boldsymbol{V} \subseteq \boldsymbol{X}$.

Entropic bound

Target: Estimate $|q(D)|$ given

1. join query $q = R_1[\boldsymbol{X}_1] \bowtie \ldots \bowtie R_n[\boldsymbol{X}_n]$
2. degree statistics Σ and size vector $\boldsymbol{B} = (B_\sigma)_{\sigma\in\Sigma}$ where each $\sigma \in \Sigma$ guarded by some atom $R_\sigma(\boldsymbol{X}_\sigma)$
3. $D \models (\Sigma, \boldsymbol{B})$, meaning $\deg_{R_\sigma}(\boldsymbol{V} \mid \boldsymbol{U}) \leq B_\sigma$ for all $\sigma = (\boldsymbol{V} \mid \boldsymbol{U}) \in \Sigma$

Entropic bound (w.r.t. degree constraints) [Khamis et al., 2017] defined by[2]

$$Ent(q, \boldsymbol{B}, \Sigma) = \sup_{\boldsymbol{w}:\Gamma_n^* \models (3)} \prod_{\sigma\in\Sigma} B_\sigma^{w_\sigma},$$

where

$$\sum_{\sigma\in\Sigma} w_\sigma h(\sigma) \geq h(\boldsymbol{X}) \tag{3}$$

[2] $\Gamma_n^* \models (3)$ denotes that (3) holds for all functions $h \in \Gamma_n^*$ (i.e., all entropic functions h)

Derivation of entropic bound

Theorem

Let $q(\boldsymbol{X})$ be a join query and D a database such that each $\sigma = (\boldsymbol{V} \mid \boldsymbol{U}) \in \Sigma$ is guarded by $R_\sigma[\boldsymbol{X}] \in q$ s.t. $\deg_{R_\sigma^D}(\boldsymbol{V} \mid \boldsymbol{U}) \leq B_\sigma$. If $\Gamma_n^* \models \sum_{\sigma\in\Sigma} w_\sigma h(\sigma) \geq h(\boldsymbol{X})$, then

$$|q(D)| \leq \prod_{\sigma\in\Sigma} B_\sigma^{w_\sigma}$$

Proof.

If h is the entropic function of $q(D)$ uniformly distributed, then

$$h(\sigma) = \mathbb{E}_{\boldsymbol{u}}[h(\boldsymbol{V} \mid \boldsymbol{U} = \boldsymbol{u})] \leq \max_{\boldsymbol{u}} h(\boldsymbol{V} \mid \boldsymbol{U} = \boldsymbol{u}) \leq \max_{\boldsymbol{u}} \log \deg_{q(D)}(\boldsymbol{V} \mid \boldsymbol{U} = \boldsymbol{u})$$
$$\leq \max_{\boldsymbol{u}} \log \deg_{R_\sigma}(\boldsymbol{V} \mid \boldsymbol{U} = \boldsymbol{u}) = \log \deg_{R_\sigma}(\boldsymbol{V} \mid \boldsymbol{U}) \leq \log B_\sigma$$

whence $\log|q(D)| = h(\boldsymbol{X}) \geq \sum_{\sigma\in\Sigma} w_\sigma h(\sigma) \geq \sum_{\sigma\in\Sigma} w_\sigma \log B_\sigma$. □

Computability

The entropic bound

$$Ent(q, \boldsymbol{B}, \Sigma) = \sup_{\boldsymbol{w}:\Gamma_n^* \models \phi} \prod_{\sigma \in \Sigma} B_\sigma^{w_\sigma}, \text{ where } \phi = \sum_{\sigma \in \Sigma} w_\sigma h(\sigma) \geq h(\boldsymbol{X}),$$

- Asymptotically tight
- Not known to be computable.
- Polynomial-time computable if each $\sigma = (\boldsymbol{V} \mid \boldsymbol{U}) \in \Sigma$ s.t. $\boldsymbol{U}$ is a singleton [Im et al. 2022; H. 2024].

Polymatroid bound (obtained by replacing Γ_n^* with Γ_n) not tight but computable in exponential time in n.

Recap: More input information

Target: Estimate $|q(D)|$ given

1. join query $q = R_1[\boldsymbol{X}_1] \bowtie \ldots \bowtie R_n[\boldsymbol{X}_n]$
2. degree constraints w.r.t. bounds $\boldsymbol{B}$

Things get more complicated when incorporating information about constraints. See [Suciu, 2023] for an in-depth review of the applications of information theory in databases.

Introduction

Complexity of CQ-Evaluation

Joins with Information Theory

Conclusion

CQ-Evaluation problem (given a database D and a Boolean CQ q, is q true for D?)

- Generally NP-complete
- Tractable when the hypergraph of the query is nearly acyclic (treewidth, generalised hypertreewidth)

Join computation (outputs of non-Boolean CQs)

- Leverage structural properties of q (fractional edge cover) and cardinalities of relations in D
- Worst-case optimal join algorithms run in time proportional to the worst-case output size, the number of attributes, and the number of relations.

Arenas, M., Barceló, P., Libkin, L., Martens, W., and Pieris, A. (2022).
Database Theory.
Open source at https://github.com/pdm-book/community.

Atserias, A., Grohe, M., and Marx, D. (2013).
Size bounds and query plans for relational joins.
SIAM J. Comput., 42(4):1737–1767.

Bodlaender, H. L. (1996).
A linear-time algorithm for finding tree-decompositions of small treewidth.
SIAM J. Comput., 25(6):1305–1317.

Gottlob, G., Leone, N., and Scarcello, F. (2003).
Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width.
J. Comput. Syst. Sci., 66(4):775–808.

Grohe, M. and Marx, D. (2006).
Constraint solving via fractional edge covers.
In SODA, pages 289–298. ACM Press.

Hannula, M. (2024).
Information inequality problem over set functions.
In ICDT, volume 290 of LIPIcs, pages 19:1–19:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Im, S., Moseley, B., Ngo, H. Q., Pruhs, K., and Samadian, A. (2022).
Optimizing polymatroid functions.
CoRR, abs/2211.08381.

Khamis, M. A., Ngo, H. Q., and Suciu, D. (2017).

What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another?
In PODS, pages 429–444. ACM.

Matús, F. (2007).
Infinitely many information inequalities.
In ISIT, pages 41–44. IEEE.

Suciu, D. (2023).
Applications of information inequalities to database theory problems.
In LICS, pages 1–30.

Yannakakis, M. (1981).
Algorithms for acyclic database schemes.
In VLDB, pages 82–94. IEEE Computer Society.

Zhang, Z. and Yeung, R. W. (1998).
On characterization of entropy function via information inequalities.
IEEE Trans. Inf. Theory, 44(4):1440–1452.