Principles of Managing Uncertain Data

Lecture 11: Query Answering in Probabilistic Databases
# Table of Contents

1. Querying Probabilistic Databases
2. Complexity Basics
3. Dichotomy in CQ Complexity
4. Approximate Query Evaluation
5. Recap
# Table of Contents

1. Querying Probabilistic Databases
2. Complexity Basics
3. Dichotomy in CQ Complexity
4. Approximate Query Evaluation
5. Recap
What is the outcome of “evaluating a query” over a probabilistic database?

Two semantics have been studied:

- Representation of the answer space
- Answer confidence (marginal probability)
Let $\mathcal{P}$ be a probabilistic database over a signature $\mathcal{R}$, and let $Q$ be a query over $\mathcal{R}$

- Recall that a query $Q$ maps every instance $I$ over $\mathcal{R}$ into a relation over the heading of $Q$ (which is also a database)

- Then $\mathcal{P}$ and $Q$ define a new probabilistic database $(\Omega, p)$, which we denote by $\text{Eval}^{\text{rep}}(Q, \mathcal{P})$:
  
  - $\Omega = \{ Q(J) \mid J \text{ is a sample of } \mathcal{P} \}$
  - $p(J) = \Pr(Q(\mathcal{P}) = J)$

Note the abuse of notation: inside $\Pr(\cdot)$, we view $\mathcal{P}$ as the random element that is equal to the random sample; hence, the meaning $Q(\mathcal{P})$ is the ordinary (deterministic) one.
Closure of a Representation System

- Given a representation system REP and query language $\mathcal{QL}$, we ask whether we can represent $\text{Eval}^{\text{rep}}(Q, \mathcal{P})$ in REP for every $Q \in \mathcal{QL}$ and $\mathcal{P} \in \text{REP}$.
- In this case we say that the representation system REP is *closed* under the query language $\mathcal{QL}$.
- A representation system is *finitely complete* if it can represent every finite probabilistic database.
Recalling pc-Instances (by Example)

<table>
<thead>
<tr>
<th>Lectures</th>
<th>Teachings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$y$</td>
</tr>
<tr>
<td>$\neg x \lor y$</td>
<td>$\neg x \lor z$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Courses</th>
</tr>
</thead>
</table>
| $\pi(x, true) = 0.5$
| $\pi(x, false) = 0.5$
| $\pi(y, true) = 0.3$
| $\pi(y, false) = 0.7$
| $\pi(z, true) = 0.4$
| $\pi(z, false) = 0.6$ |
Completeness of pc-Instances

**Theorem**

The class of pc-Instances (with Boolean variables) is a finitely complete representation system.

**Corollary**

The class of pc-Instances is closed under the relational algebra.
Answer Confidence

- In the semantics of *answer confidence*, we associate each answer (tuple) with its individual (marginal) probability:

\[
\text{EVAL}^{\text{mrg}}(Q, \mathcal{P}) \overset{\text{def}}{=} \{(a, p) \mid a \in Q(J) \text{ for some sample } J \text{ and } p = \Pr(a \in Q(\mathcal{P}))\}
\]

- That is, we concatenate to each tuple the probability that it is indeed an answer in a random possible world
Example

<table>
<thead>
<tr>
<th>ObsLects</th>
<th>ObsTAs</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>lecturer</td>
<td>ta</td>
<td>course</td>
</tr>
<tr>
<td>Ahuva</td>
<td>Asma</td>
<td>DB</td>
</tr>
<tr>
<td>0.5</td>
<td>0.4</td>
<td>Ahuva</td>
</tr>
<tr>
<td>Avia</td>
<td>Alon</td>
<td>PL</td>
</tr>
<tr>
<td>0.3</td>
<td>0.9</td>
<td>Avia</td>
</tr>
</tbody>
</table>

Which are the courses with their full staff observed?

\[ Q(z) :\neg \text{ObsLects}(x), \text{ObsTAs}(y), \text{Courses}(z, x, y) \]

DB: \[0.5 \times 0.9\]  
PL: \[0.3 \times 0.4\]

Does any course have a full staff observed?

\[ Q() :\neg \text{ObsLects}(x), \text{ObsTAs}(y), \text{Courses}(z, x, y) \]

Yes: \[1 - (1 - 0.5 \times 0.9) \times (1 - 0.3 \times 0.4)\]  
No: \[1 - \Pr(Yes)\]
Focus on Confidence

- We will focus on the semantics $\text{Eval}^{\text{mrg}}(Q, \mathcal{P})$
- Various reasons:
  - Representation systems with interesting closure properties are capable of expressing significant correlations, and hence, cast query evaluation highly intractable
  - We will see that query evaluation is challenging already over tuple-independent databases, which do not have any interesting closure properties
  - There is a rich literature on $\text{Eval}^{\text{mrg}}(Q, \mathcal{P})$
    - Not so much on $\text{Eval}^{\text{rep}}(Q, \mathcal{P})$
Query evaluation in probabilistic databases entails computing/estimating probabilities (a.k.a. probabilistic inference).

To understand and analyze the complexity of numerical computations, we need appropriate complexity measures:

- The standard theory from undergrad studies is mainly about deciding yes/no, not about computing numbers.
Table of Contents

1. Querying Probabilistic Databases
2. Complexity Basics
3. Dichotomy in CQ Complexity
4. Approximate Query Evaluation
5. Recap
A decision problem is a mapping $F : \{0, 1\}^* \rightarrow \{0, 1\}$

- For example, is the given string representing a:
  - connected graph?
  - satisfiable CNF formula? (CNF-SAT)
  - database that satisfies some fixed Boolean query?

A function problem is a mapping $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$

- For example:
  - Mapping a (representation of a) given graph to (a representation of) its connected components
  - Mapping a given CNF formula into the number of its satisfying truth assignments
  - Mapping a tuple-independent probabilistic DB into the probability that it satisfies some fixed Boolean query

The natural associated task: given $x \in \{0, 1\}^*$, compute $F(x)$
A *complexity class* is a class $C$ of computational problems such that each problem in $C$ can be solved by a Turing machine that satisfies some *properties*. These properties are typically limitations on the machine’s power, e.g., determinism, bound on the execution time or used space.
Examples of Complexity Classes

- **P**: Decision problems solvable in polynomial time
  - That is, problems $D$ where there exists a Turing machine $M$ and a polynomial $p$, such that $M$ solves $D$ and $M$ terminates on input $x$ after at most $p(|x|)$ steps

- **FP**: Function problems solvable in polynomial time

- **NP**: Decision problems solvable in polynomial-time by a non-deterministic Turing machine
  - That is, $B(x) = 1$ if and only if there is an accepting path on $x$

- **coNP**: Decision problems with complement in NP
An **NP relation** is a relation \( R \subseteq \{0, 1\}^* \times \{0, 1\}^* \) that satisfies the following:

- Membership (of a given pair) in \( R \) can be decided in polynomial time.
- There is a polynomial \( p \) such that for every \((x, y) \in R\) we have \( |y| \leq p(|x|) \).

- If \((x, y) \in R\), then we often refer to \( y \) as a **witness** for \( x \).

- **NP** can be described as the class of all decision problems of the form

\[
B(x) = \begin{cases} 
1 & \text{if } \exists y[(x, y) \in R]; \\
0 & \text{otherwise.}
\end{cases}
\]

where \( R \) is an NP relation.
Leslie Valiant (2010 Turing Award Winner) has established complexity theory for *counting problems* [Val79]

- \#P is the class of problems that count the number of witnesses in an NP relation; formally, \#P contains all function problems of the following form, where $R$ is an NP relation:

$$F(x) = |\{y \mid (x, y) \in R\}|$$

- Equivalently: count the number of accepting paths of a polynomial-time non-deterministic Turing machine
Reductions

- Let $H$ and $W$ be two computational problems
  - Typically, $H$ is some *hard* problem, and $W$ is a problem that we *wish* to prove hardness for
- A *reduction* of $H$ to $W$ is a way of showing:
  - We could solve $H$ if we could solve $W$
  - $W$ is “at least as hard” as $H$
- There is more than one way to define a reduction
Types of Reductions

- The common type of (polynomial-time) **reduction** from $H$ to $W$ is that of **Karp reduction**:

  Given input for $H$, transform it (in polynomial time) into input for $W$, such that both inputs have the same answer.

- Another common type (more applicable to function problems) is a polynomial time **Turing reduction** (a.k.a. **Cook reduction**):

  Given input $x$ for $H$, solve it in polynomial time using $W$ via an **oracle**: an $O(1)$ subroutine that returns $W(x')$ given $x'$. 
#P-Hardness

A function problem $F$ is **hard for #P** (or **#P-hard**) if for every problem $G$ in #P there is a Turing reduction from $G$ to $F$.

If $F$ itself is in #P, then $F$ is **#P-complete**.
Examples of \#P-Complete Problems

- \#CNF: Number of satisfying assignments to a given CNF
- \#DNF: Number...DNF Why?
- Number of tuples in $Q(I)$, when $Q$ is a given CQ and $I$ is a given database instance
- Number of vertex covers in a graph
- Number of perfect matchings in a bipartite graph (a.k.a. the permanent of a 0/1 matrix)
- Number of paths between two given nodes in a graph
- Number of topological orderings for a given DAG
How Hard is \#P-Hard?

- What can we do with an oracle to a \#P-hard problem?
- More formally, which problems are Turing-reducible to, e.g., \#DNF?
  - Every problem in NP, and every problem in coNP
  - In fact, every problem in the polynomial hierarchy (Toda’s Theorem [TO92]), for example:

> Given a CNF formula $\varphi(x, y, z)$, is there a tuple $a$ such that for every tuple $b$ there exists a tuple $c$ such that $\varphi(a, b, c)$?
How is probability related to counting?

We explain by an example

Consider a propositional formula $\varphi(x_1, \ldots, x_n)$

Now suppose that every variable takes a random truth value:
$\Pr(x_i = \text{true}) = \Pr(x_i = \text{false}) = 0.5$

Also assume that the $x_i$’s are probabilistically independent

Could you compute $\Pr(\varphi(x))$?
A Turing Reduction

\[ \varphi(x_1, \ldots, x_n): \text{reduce counting to probability computation} \]

\[
\Pr(\varphi(x)) = \sum_{a|\varphi(a)} \Pr(x = a) = \sum_{a|\varphi(a)} 0.5^n = 2^{-n} \cdot |\{a \mid \varphi(a)\}| 
\]

We get the following Turing reduction:

To compute the number of satisfying assignments for \(\varphi\), compute \(\Pr(\varphi(x))\) and multiply by \(2^n\)

**Note:** \(2^n\) is represented by \(n\) bits; multiplication in PTime in \(n\)

Hence, computing the probability of a given propositional formula over given random Boolean variables is hard for \#P
Comment on Numerical Representation

- For complexity analysis, we need to assume something about the representation of numbers.
- We will assume that numbers are given in their *rational* representation, namely, \( a = n/m \) is represented as \((n, m)\).
  - Each of \( n \) and \( m \) is given in binary.

Numerical representation will not be an issue for us; but generally, ignoring the size of numerical representation may easily lead to wrong statements.
## Table of Contents

1. Querying Probabilistic Databases
2. Complexity Basics
3. Dichotomy in CQ Complexity
4. Approximate Query Evaluation
5. Recap
Hierarchical Queries

- Let $Q$ be a Boolean CQ without self joins
- For a variable $x \in \text{Var}(Q)$, denote by $\text{Atoms}(x)$ the set of all $\alpha \in \text{Atoms}(Q)$ such that $x$ occurs in $\alpha$; in notation:

\[
\text{Atoms}(x) \overset{\text{def}}{=} \{ \alpha \in \text{Atoms}(Q) \mid x \in \text{Var}(\alpha) \}
\]

- We say that $Q$ is *hierarchical* if for every two variables $x$ and $y$ in $Q$, at least one of the following holds:
  - $\text{Atoms}(x) \subseteq \text{Atoms}(y)$
  - $\text{Atoms}(y) \subseteq \text{Atoms}(x)$
  - $\text{Atoms}(x) \cap \text{Atoms}(y) = \emptyset$
Examples

- Every CQ with two or fewer atoms is hierarchical
- \( R(x), S(x, y), T(y) \) is not hierarchical; Why?
- The following is hierarchical:

\[
R(x, y, u), S(x, y, z), T(x), U(w), V(w, v)
\]

- If \( Q \) is hierarchical, then every subquery of \( Q \) (subset of \( Q \)'s atoms) is hierarchical
Let $Q$ be a Boolean CQ without self joins.

- If $Q$ is hierarchical, then it can be evaluated in polynomial time over tuple-independent databases.
- Otherwise, its evaluation over tuple-independent databases is \#P-hard.
Proof of Hardness Side

Two parts:

1. Prove \#P-hardness for a specific CQ:

\[ Q_{RST}() := R(x), S(x, y), T(y) \]

2. Prove \#P-hardness for an arbitrary CQ \( Q \) w/o self joins that is non-hierarchical ... by reducing query evaluation over \( Q_{RST} \) to query evaluation over \( Q \)
Lemma

Evaluating $Q_{RST}$ over a tuple-independent database is $\#P$-hard.

Proof: Reduction from monotone bipartite $\#$DNF that has been proved hard [PB83]
A **monotone bipartite DNF** formula is a 2-DNF formula such that there are two **disjoint** sets $X$ and $Y$ of variables where each clause has the form $x \land y$ for $x \in X$ and $y \in Y$

Example: $(x_1 \land y_1) \lor (x_1 \land y_2) \lor (x_2 \land y_1) \lor (x_2 \land y_3)$

In particular, no negation

Monotone bipartite #DNF is the problem of counting how many satisfying assignments does a given such formula have

Equivalently: given a bipartite graph, how many subsets of the nodes include at least one edge?
\[ Q_{\text{RST}}() := R(x), S(x, y), T(y) \]

\[(x_1 \land y_1) \lor (x_1 \land y_2) \lor (x_2 \land y_1) \lor (x_2 \land y_3)\]

\[
\begin{array}{c|c}
R & A \\
\hline
x_1 & 0.5 \\
\hline
x_2 & 0.5 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
S & A & B \\
\hline
x_1 & y_1 & 1.0 \\
x_1 & y_2 & 1.0 \\
x_2 & y_1 & 1.0 \\
x_2 & y_3 & 1.0 \\
\end{array}
\]

\[
\begin{array}{c|c}
T & B \\
\hline
y_1 & 0.5 \\
y_2 & 0.5 \\
y_3 & 0.5 \\
\end{array}
\]
Let $Q$ be an arbitrary CQ such that:
- $Q$ has no self joins
- $Q$ is non-hierarchical

Claim: we can choose variables $x$ and $y$ and atoms $\alpha_x, \alpha_y$ and $\alpha_{x,y}$ such that:
- $x \in \text{Var}(\alpha_x)$ and $y \notin \text{Var}(\alpha_x)$
- $y \in \text{Var}(\alpha_y)$ and $x \notin \text{Var}(\alpha_y)$
- $x, y \in \text{Var}(\alpha_{x,y})$

(Discussion until everyone is convinced)

W.l.o.g. we assume that in both $Q_{RST}$ and $Q$ these are the same $x$ and $y$

We will show how to reduce query evaluation over $Q_{RST}$ to query evaluation over $Q$
Example of Reduction

\[ R(x), S(x, y), T(y) \]

\[ \downarrow \]

\[ U(x, z), V(x, u), W(x, y, z), Y(y, a) \]

\[ \alpha_x, \alpha_{x y}, \alpha_y \]
Example of Reduction

\[ R(x), S(x, y), T(y) \]

\[
\begin{array}{c|c|c}
R & p_1 & p_2 \\
\hline
1 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c}
S & q_{13} & q_{23} \\
\hline
1 & 2 & 3 & 4 \\
\end{array}
\quad
\begin{array}{c|c|c}
T & r_3 & r_4 \\
\hline
3 & 4 \\
\end{array}
\]

\[
\downarrow
\]

\[
\begin{array}{c|c|c}
U & c & p_1 \\
\hline
1 & 2 & c & p_2 \\
\end{array}
\quad
\begin{array}{c|c|c}
V & c & 1 \\
\hline
1 & 2 & 4 \\
\end{array}
\quad
\begin{array}{c|c|c}
W & c & q_{13} & q_{23} \\
\hline
1 & 2 & 3 & 4 \\
\end{array}
\quad
\begin{array}{c|c|c}
Y & a & r_3 \\
\hline
3 & 4 & a & r_4 \\
\end{array}
\]

\[ U(x, z), V(x, u), W(x, y, z), Y(y, a) \]

\[ \alpha_x, \alpha_{xy}, \alpha_y \]
Reduction

- We are given a tuple-independent database \( \mathcal{P} = (I, \pi) \), and we wish to evaluate \( \Pr(\mathcal{P} \models Q_{RST}) \) by reducing to query evaluation over \( Q \).
- We will construct a tuple-independent database \( \mathcal{P}' = (I', \pi') \) such that
  \[
  \Pr(\mathcal{P} \models Q_{RST}) = \Pr(\mathcal{P}' \models Q)
  \]
Constructing $I'$

- Start with an empty $I'$
- Select some constant value $c$
- For each homomorphism (or *grounding*) $\mu$ from $Q_{RST}$ to $I$ and atom $\alpha \in \text{Atoms}(Q)$
  1. Extend $\mu$ to a mapping over $\text{Var}(Q)$ by mapping every variable other than $x$ and $y$ to $c$
  2. Create the fact $f = \mu(\alpha)$
  3. Add $f$ into $I'$
Constructing $\pi'$

- Recall: $Q_{RST}() :- R(x), S(x, y), T(y)$
- For each extension of a homomorphism $\mu$ from $Q_{RST}$ to $I$ and atom $\alpha \in \text{Atoms}(Q)$ we define:

$$\pi' (\mu(\alpha)) = \begin{cases} 
\pi (R(\mu(x))) & \text{if } \alpha = \alpha_x; \\
\pi (S(\mu(x), \mu(y))) & \text{if } \alpha = \alpha_{xy}; \\
\pi (T(\mu(y))) & \text{if } \alpha = \alpha_y; \\
1 & \text{otherwise.}
\end{cases}$$
How do we prove that $\Pr(\mathcal{P} \models Q_{\text{RST}}) = \Pr(\mathcal{P}' \models Q)$?

First, we can ignore all the facts of $I$ that are not in the image of homomorphisms from $Q_{\text{RST}}$; with that:

There is a (straightforward) bijection $\varphi$ between the samples $J$ of $\mathcal{P}$ and $\mathcal{P}'$ such that:

- $\Pr(J) = \Pr(\varphi(J))$
- $J \models Q_{\text{RST}}$ if and only if $\varphi(J) \models Q$

(Discussion until everyone is convinced)

From that, the conclusion $\Pr(\mathcal{P} \models Q_{\text{RST}}) = \Pr(\mathcal{P}' \models Q)$ is simply due to the definition of these probabilities.
Let $Q$ be a hierarchical CQ without self joins

- A root variable of $Q$ is a variable $x \in \text{Var}(Q)$ such that $\text{Atoms}(x)$ is maximal w.r.t. set containment

Which are the root variables in $R(x,y), S(x), T(z)$?
Recursive algorithm

Input: tuple-independent $\mathcal{P} = (I, \pi)$

```plaintext
if Var($Q$) = $\emptyset$ then
    return $\Pr(\mathcal{P} \models Q)$ // How?

Select a root variable $x$

if Atoms($x$) $\not\subseteq$ Atoms($Q$) then
    Apply an independent-join reduction
else
    Apply an independent-project reduction
```
Independent-Join Reduction

- Suppose that $x$ is a root variable that occurs in some (but not all) of the atoms
- Write $Q = Q_x \land Q'$ where $Q_x$ consists of the atoms that contain $x$, and $Q'$ consists of the remaining atoms
- Claim: $Q_x$ and $Q'$ do not share any atoms or variables
  - *(Discussion until everyone is convinced)*
- Therefore, the events $\mathcal{P} \models Q_x$ and $\mathcal{P} \models Q'$ are **probabilistically independent**
  - *(Discussion until everyone is convinced so far)*
- Hence, $\Pr(\mathcal{P} \models Q) = \Pr(\mathcal{P} \models Q_x) \times \Pr(\mathcal{P} \models Q')$
- We reduced from $Q$ to $Q_x$ and $Q'$, each having fewer atoms
Independent-Project Reduction

- Suppose that $x$ is a variable that occurs in every atom
- Let $A$ be the set of all the constants that $x$ can take in $I$
- We have the following:

$$\Pr(\mathcal{P} \models Q) = 1 - \prod_{a \in A} \left(1 - \Pr(\mathcal{P} \models Q[x \rightarrow a])\right)$$

- (Discussion until everyone is convinced)

- We reduced from $Q$ to $Q[x \rightarrow a]$, which has fewer variables
- Claim: $Q[x \rightarrow a]$ is hierarchical
Input: tuple-independent $\mathcal{P} = (I, \pi)$

$$\text{Eval}(Q)(\mathcal{P})$$

if $\text{Var}(Q) = \emptyset$ then
  return $\Pr(\mathcal{P} \models Q)$

Select a root variable $x$

if $\text{Atoms}(x) \subsetneq \text{Atoms}(Q)$ then
  Return $\text{Eval}(Q_A)(\mathcal{P}) \times \text{Eval}(Q')(\mathcal{P})$
else
  Return $1 - \prod_{a \in A} \left(1 - \text{Eval}(Q[x \rightarrow a])(\mathcal{P})\right)$

That’s it!
Discussion on Complexity

Why is the algorithm considered “polynomial time”?

What is the dependence on the size of the CQ?
Dichotomy for non-Boolean CQs Without Self Joins

Generalized Dichotomy [DS04]

Let $Q(x)$ be a CQ without self joins. Let $Q_b$ be the Boolean CQ $Q[x \rightarrow a]$ for some tuple $a$ of constants.

- If $Q_b$ is hierarchical, then $Q$ can be evaluated in polynomial time over tuple-independent database.
- Otherwise, evaluating $Q$ is $\#P$-hard.
Examples

Mark of the following as PTime / #P-hard:

\[ Q() \leftarrow R(x), S(x, y), T(y) \quad \text{#P-hard} \]
\[ Q(y) \leftarrow R(x), S(x, y), T(y) \quad \text{PTime} \]
\[ Q(x) \leftarrow R(x, z), S(x, y), T(y, z) \quad \text{#P-hard} \]
\[ Q(y) \leftarrow R(x, z), S(x, y), T(y, z) \quad \text{#P-hard} \]
\[ Q(x, y) \leftarrow R(x, z), S(x, y), T(y, z) \quad \text{PTime} \]
Suciu and Dalvi generalized their dichotomy into general UCQs (and in particular CQs with self joins) [DS12].

The generalized dichotomy has:

- A substantially more involved separation condition
- A substantially longer (87 pages) and more complicated proof
Let $\mathcal{R}$ be a signature

- Given: a CQ $Q(x)$ over $\mathcal{R}$ such that
  - $Q$ has no self joins
  - $Q_b$ is hierarchical
SQL Rewriting

- Let $\mathcal{R}$ be a signature
- Given: a CQ $Q(x)$ over $\mathcal{R}$ such that
  - $Q$ has no self joins
  - $Q_b$ is hierarchical
- Goal: Compute a SQL query $Q_p(x, z)$ over $\mathcal{R}^+$, such that for every tuple-independent database $\mathcal{P} = (I, \pi)$ over $\mathcal{R}$ we have:

\[
Q_p(I^+) = \{(a, p) \mid a \in Q(I) \land p = \Pr(a \in Q(\mathcal{P}))\}
\]

where $\mathcal{R}^+$ is obtained from $\mathcal{R}$ by adding a probability attribute to each relation symbol, and $I^+$ is obtained from $I$ by setting this attribute according to $\pi$
We denote $Q$ as the following SQL query:

```
SELECT X FROM R WHERE AC
```

Where:
- $R$ is a sequence $R_1, \ldots, R_m$ of relation names with a distinguished attribute $P$
- $X$ is a sequence of attributes $A$
  - For simplicity, we assume that different relations use disjoint sets of attribute names, except for $P$
- $AC$ is a conjunction of conditions of the form $A = B$ or $A = t$ where $t$ is a constant
- None of the above attributes $A$ and $B$ is the distinguished $P$
As a first step, we will extend $X$ with \textit{every} attribute that corresponds to an output variable; call the result $X^+$

$$\text{SELECT } X, P \text{ FROM ViewFor}(Q, X^+)$$

We will apply the recursive algorithm by building multiple views of the form $\text{ViewFor}(Q_0, X_0)$
Recall: in this case all the attributes are in the output (except for those that are constants in the CQ)
CREATE VIEW ViewFor(Q, Y) AS
SELECT Y, Q1.P * Q2.P AS P
FROM
  ViewFor(Q_x, Y \cap \text{Var}(Q_x)) Q1,
  ViewFor(Q', Y \cap \text{Var}(Q')) Q2
WHERE AC \cap Y

AC \cap Y denotes the restriction of AC to comparisons involving attributes only from Y
Independent-Project Reduction

CREATE VIEW ViewFor(Q, Y) AS
SELECT Y, 1-EXP(SUM(LN(1-Q1.P))) AS P
FROM ViewFor(Q, Y ∪ Z_x) Q1
GROUP BY Y

- EXP(SUM(LN(x))) means Prod(x), which is not supported by PSQL
- Z_x consists of all the attributes where x occurs
- Recall that x is a root variable that occurs in every atom
Example

\[ Q(z) := R(x, y), S(x, z), T(z, a) \]

Representation with disjoint attribute sets:
\[
R(x_R, y_R, P) \quad S(x_S, z_S, P) \quad T(z_T, a_T, P)
\]

SELECT z_S
FROM R, S, T
WHERE x_R = x_S AND z_S = z_T AND a_T = 'a'
Step 0

```
SELECT zS, P FROM ViewFor(Q, zS, zT)
```

```
SELECT zS
FROM R, S, T
WHERE xR=xS AND zS=zT AND aT='a'
```
CREATE VIEW ViewFor($Q$,zS,zT) AS
SELECT zS, zT, Q1.P * Q2.P AS P
FROM
ViewFor($R(x,y), S(x,z)$,zS) Q1,
ViewFor($T(z,a)$, zT) Q2
WHERE zS=zT
ViewFor((R(x, y), S(x, z)), zS) via Independent-Project Reduction

CREATE VIEW ViewFor((R(x, y), S(x, z)), zS) AS
SELECT zS, 1-EXP(SUM(LN(1-Q1.P))) AS P
FROM ViewFor((R(x, y), S(x, z)), zS, xR, xS) Q1
GROUP BY zS
ViewFor((R(x, y), S(x, z)), zS, xR, xS)) via Independent-Join Reduction

CREATE VIEW ViewFor((R(x, y), S(x, z)), zS, xR, xS) AS
SELECT zS, xR, xS, Q1.P * Q2.P AS P
FROM
ViewFor(R(x, y), xR) Q1, ViewFor(S(x, z), zS, xS) Q2
WHERE xR=xS
CREATE VIEW ViewFor($R(x, y), xR$) AS
SELECT xR, 1-EXP(SUM(LN(1-Q1.P))) AS P
FROM ViewFor($R(x, y), xR, yR$) Q1
GROUP BY xR
**ViewFor**\((R(x,y), xR, yR)\) via Base Case

```sql
CREATE VIEW ViewFor(R(x,y), xR, yR) AS
SELECT xR, yR, P
FROM R
```

- It simply copies \(R\)!
- Similar to \(ViewFor(S(x,z), zS, xS)\)
CREATE VIEW ViewFor(T(z, a), zT) AS
SELECT zT, P
FROM T
WHERE aT='a'
Table of Contents

1. Querying Probabilistic Databases
2. Complexity Basics
3. Dichotomy in CQ Complexity
4. Approximate Query Evaluation
5. Recap
Additive and Multiplicative Approximations

- Let $a$ and $b$ be two numbers, and let $\epsilon > 0$ be a small number (e.g., 0.01)
- We say that $b$ is an *additive* (or *absolute*) approximation of $a$ with error $\epsilon$ if
  \[ a - \epsilon < b < a + \epsilon \]
- We say that $b$ is a *multiplicative* (or *relative*) approximation of $a$ with error $\epsilon$ if
  \[ \frac{a}{1 + \epsilon} < b < (1 + \epsilon) a \]
Let $f : \{0, 1\}^* \rightarrow \mathbb{R}$ be a numeric function.

A *polynomial-time approximation scheme* (PTAS) for $f$ is an algorithm $A$ such that:

- $A$ takes as input an input $x \in \{0, 1\}^*$ and $\epsilon > 0$, and computes an approximation of $f$ with error $\epsilon$.
- For every fixed $\epsilon > 0$, the running time of $A$ is polynomial in $|x|$.

A *fully polynomial-time approximation scheme* (FPTAS) for $f$ is a PTAS for $f$ that runs in time polynomial in $|x|$ and $1/\epsilon$.

Note: PTAS and FPTAS for $f$ mean that an approximation for $f$ is (theoretically) tractable for as small an error as one desires; in the case of FPTAS, the error has a “manageable” impact on the running time.
Randomized Approximation Schemes

- Let \( f : \{0, 1\}^* \rightarrow \mathbb{R} \) be a numeric function

- A \textit{polynomial-time randomized approximation scheme} (PRAS) for \( f \) is a randomized algorithm \( A \) such that:
  - \( A \) takes as input an input \( x \in \{0, 1\}^* \) and \( \epsilon > 0 \), and with a probability of at least \( 2/3 \) computes an approximation of \( f \) with error \( \epsilon \)
  - For every fixed \( \epsilon > 0 \), the running time of \( A \) is polynomial in \( |x| \)

- A \textit{fully polynomial-time randomized approximation scheme} (FPRAS) for \( f \) is a PRAS for \( f \) that runs in time polynomial in \( |x| \) and \( 1/\epsilon \)
> The reliability factor $2/3$ is arbitrary; every number greater than $1/2$ would work.

> Given an FPRAS, we can decrease the probability of failure to any desired number (e.g., 0.001) by repeatedly applying the algorithm and taking the median number.

> By applying known concentration bounds (e.g., Hoeffding’s inequality), we get that the failure probability decreases exponentially in the number of trials.

> We will prove it in a few slides.
Hoeffding’s Inequality

Consider $n$ independent tosses of a coin with probability $p$ of heads. Let $H$ be the number of heads. For all $\epsilon > 0$ we have:

$$\Pr \left( p - \epsilon < \frac{H}{n} < p + \epsilon \right) \geq 1 - 2e^{-2\epsilon n}$$

In particular, by running $O(1/\epsilon)$ trials we get an additive approximation of $p$ with probability greater than $2/3$. 
FPRAS Amplification (1)

- Let $f$ be a numeric function, and suppose that $A$ is an FPRAS (additive or relative) that gives an $\epsilon$-approximation with failure probability $\delta_A \leq 1/3$.
- Also, suppose that we are given a number $\delta > 0$ (e.g., $\delta = 0.01$), and we wish to reduce failure probability to $\delta$.
- Given input $x$, let $a_1, \ldots, a_n$ be random numbers that we get by $n$ independent runs of $A$.
  - We decide later on what $n$ should be and how it depends on $\delta$.
- Let $b_1, \ldots, b_n$ be $n$ Boolean random variables such that
  
  $b_i = \begin{cases} 
  1 & \text{if } a_i \text{ is an } \epsilon\text{-approximation of } f(x); \\
  0 & \text{otherwise.} 
  \end{cases}$

- $b_i$ are viewed as the tosses of a coin with probability $(1 - \delta_A) \geq 2/3$ for $1$.
FPRAS Amplification (2)

- Let $a$ be the median of $a_1, \ldots, a_n$
- What is the probability that $a$ is not an $\epsilon$-approximation?
- If $a$ is not an $\epsilon$-approximation, then at least half of the $a_i$ are not $\epsilon$-approximations, and at least half of the $b_i$s are zero
- As usual, let $H$ be the number of 1s (heads); by Hoeffding’s:

$$
\Pr(a \text{ not an } \epsilon\text{-approx}) \leq \Pr\left(\frac{H}{n} \leq \frac{1}{2}\right) \leq \Pr\left(\frac{H}{n} \leq (1 - \delta_A) - \frac{1}{6}\right) \\
\leq 2e^{-2 \times (1/6) \times n} = 2e^{-n/3} < e^{-n/3+1}
$$

- To get the failure probability at most $\delta$, we need $e^{-n/3+1} \leq \delta$, or equivalently, $n \geq 3(\log \frac{1}{\delta} + 1)$
Additive Approximation of Boolean Queries

- Consider a Boolean query \( Q \), such that \( Q \) can be evaluated in polynomial time
- Then there is an additive FPRAS for \( Q \) over a pc-instance:

\[
\begin{align*}
c &= 0 \\
\text{for } i = 1, \ldots, n \text{ do} & \\
& \quad \text{Sample a random possible world } J \\
& \quad \text{if } Q(J) \text{ then} \\
& \quad \quad ++c \\
\text{Return } c/n
\end{align*}
\]
We will show that there is a multiplicative FPRAS for every CQ (in fact, UCQ) over tuple-independent (in fact, BID) databases.

We will first translate the evaluation of a Boolean UCQ into the computation of a probability of a disjunctive event.

Then, we will show an FPRAS for disjunctive events, under assumptions that hold in our case.
Translating a Boolean UCQ into a Disjunctive Event

- Let $P$ be a probabilistic database, where all the samples are subinstances of some instance $I$
- Let $Q$ be a UCQ
- The event “$P \models Q$” can be phrased as a disjunction $E_1 \lor \cdots \lor E_m$ of events
  - Each $E_i$ corresponds to a set $F_i$ of facts in $I$, and states that “every fact in $F_i$ is in the sample”
  - The facts in $F_i$ are those obtained by applying a homomorphism from one of the CQs of $Q$ to $I$
    - In common terminology, each $E_i$ is a *grounding* of one of $Q$’s CQs
  - The number $m$ of these $E_i$ is polynomial in $|I|$, since we assume that $Q$ is fixed
Example on a BID Database

**Advises**

<table>
<thead>
<tr>
<th>key</th>
<th>lecturer</th>
<th>student</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Asma</td>
<td>Ahuva</td>
<td>0.5</td>
</tr>
<tr>
<td>a</td>
<td>Alon</td>
<td>Avia</td>
<td>0.3</td>
</tr>
<tr>
<td>b</td>
<td>Asma</td>
<td>Anna</td>
<td>0.3</td>
</tr>
<tr>
<td>b</td>
<td>Adi</td>
<td>Ariel</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Teaches**

<table>
<thead>
<tr>
<th>key</th>
<th>teacher</th>
<th>course</th>
<th>0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Asma</td>
<td>DB</td>
<td>0.4</td>
</tr>
<tr>
<td>a</td>
<td>Asma</td>
<td>PL</td>
<td>0.3</td>
</tr>
<tr>
<td>b</td>
<td>Ahuva</td>
<td>OS</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[ Q() := \left( \text{Advises}(u, x, y) \land \text{Teaches}(v, x, z) \right) \lor \left( \text{Advises}(u, y, x) \land \text{Teaches}(v, x, z) \right) \]

\[ Q \text{ is equivalent to:} \]

- \( E_1 \): both \text{Advises}(a, Asma, Ahuva) and \text{Teaches}(a, Asma, DB) are chosen; or
- \( E_2 \): both \text{Advises}(a, Asma, Ahuva) and \text{Teaches}(b, Ahuva, OS) are chosen; or
- \( E_3 \): both \text{Advises}(b, Asma, Anna) and \text{Teaches}(a, Asma, PL) are chosen; or
- ...
Let $\mathcal{P} = (\Omega, p)$ be a probability space and $E_1, \ldots, E_m$ a sequence of events in $\mathcal{P}$

- All given in some representation

Assume that for each $E_i$ we can efficiently:

1. Compute the probability of $E_i$
2. Test whether $E_i$ is true in a given random sample
3. Sample from the subspace conditioned on $E_i$

Claim: These assumptions hold in our case (UCQs over BIDs)

(Discussion until everyone is convinced)

We will show the Karp-Luby-Madras FPRAS for $\Pr(\bigvee_{i=1}^m E_i)$ under these assumptions [KLM89]
Algorithm (1)

\[
\Pr \left( \bigvee_{i=1}^{m} E_i \right) \overset{\text{easy}}{=} \Pr \left( \bigvee_{i=1}^{m} \neg E_1 \land \cdots \land \neg E_{i-1} \land E_i \right)
\]

\overset{\text{disjoint}}{=} \sum_{i=1}^{m} \Pr \left( \neg E_1 \land \cdots \land \neg E_{i-1} \land E_i \right)

\overset{\text{definition}}{=} \sum_{i=1}^{m} \Pr (E_i) \times \Pr \left( \neg E_1 \land \cdots \land \neg E_{i-1} \mid E_i \right)

We can compute \( \Pr(E_i) \)

We can additively approximate \( \Pr(\neg E_1 \land \cdots \land \neg E_{i-1} \mid E_i) \)

Using Hoeffding’s inequality, since we can sample the conditional subspace of each \( E_j \), and test for the conjunction in the sample

Let \( a_i \) be an additive approx. of \( \Pr(\neg E_1 \land \cdots \land \neg E_{i-1} \mid E_i) \) with error \( \epsilon' \) and failure probability \( \delta \)

We will determine \( \epsilon' \) in the end

\( \delta \) should be small enough so that with probability at least \( 2/3 \) all the \( a_i \) are good
Algorithm (3)

We will approximate \( \Pr(\bigvee_{i=1}^{m} E_i) \) using \( \sum_{i=1}^{m} \Pr(E_i) \times a_i \)

\[
\sum_{i=1}^{m} \Pr(E_i) \times a_i \leq \sum_{i=1}^{m} \Pr(E_i) \times \left( \Pr(\neg E_1 \land \cdots \land \neg E_{i-1} \mid E_i) + \epsilon' \right)
\]

\[
= \sum_{i=1}^{m} \Pr(E_i) \times \Pr(\neg E_1 \land \cdots \land \neg E_{i-1} \mid E_i) + \sum_{i=1}^{m} \Pr(E_i) \times \epsilon'
\]

\[
= \Pr\left(\bigvee_{i=1}^{m} E_i\right) + \epsilon' \sum_{i=1}^{m} \Pr(E_i) \leq \Pr\left(\bigvee_{i=1}^{m} E_i\right) + \epsilon' m \times \Pr\left(\bigvee_{i=1}^{m} E_i\right)
\]

\[
= (1 + \epsilon' m) \times \Pr\left(\bigvee_{i=1}^{m} E_i\right)
\]

Similarly:

\[
\sum_{i=1}^{m} \Pr(E_i) \times a_i \geq (1 - \epsilon' m) \times \Pr\left(\bigvee_{i=1}^{m} E_i\right)
\]
In conclusion, we have:

\[(1 - \epsilon'm)\Pr\left(\bigvee_{i=1}^{m} E_i\right) \leq \sum_{i=1}^{m} \Pr(E_i) \times a_i \leq (1 + \epsilon'm)\Pr\left(\bigvee_{i=1}^{m} E_i\right)\]

So, if we want the error \(\epsilon\) we should use \(\epsilon' = \epsilon/m\)
Corollary

FPRAS for UCQs

Let \( R \) be a signature, and let \( Q \) be a UCQ over \( R \).
\( \text{Eval}^{\text{mrg}}(Q, P) \) has a multiplicative FPRAS over BID databases.
<table>
<thead>
<tr>
<th>Inference</th>
<th>QL</th>
<th>Data Rep.</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>CQs w/o self joins</td>
<td>Tuple-Independent</td>
<td>Hierarchical CQs in PTime; otherwise #P-hard (Suciu-Dalvi dichotomy)</td>
</tr>
<tr>
<td>Additive Approx.</td>
<td>Any PTime language</td>
<td>pc-instances</td>
<td>FPRAS (average over samples)</td>
</tr>
<tr>
<td>Relative Approx.</td>
<td>UCQs (positive RA)</td>
<td>BID-instances</td>
<td>FPRAS (Karp-Luby-Madras)</td>
</tr>
</tbody>
</table>
References


End of lecture 11

Query Answering in Probabilistic Databases