A Toolbox of Query Evaluation Techniques for Probabilistic Databases

Dan Olteanu, Oxford University Computing Laboratory
Uncertain and Probabilistic Data

Uncertain and probabilistic data is commonplace:
- (Web) information extraction
- Processing manually entered data (such as census forms)
- Data integration, data cleaning
- Risk management: Decision support queries, hypothetical queries
- Social network analysis
- Managing scientific data; sensor data
- Crime fighting, surveillance, plagiarism detection, predicting terrorist actions

Relational DBMSs are not flexible enough to accommodate such data
- one tries to find a “good” yet partial fit of the data in the relational format

Recent years have seen advances in developing
- representation models for uncertain/probabilistic data,
- uncertainty-aware query languages, and
- scalable query evaluation techniques for such data.
Probabilistic Databases Today

Many active projects, for instance
- Mystiq, Lahar (Washington U.)
- Trio (Stanford)
- MCDB (IBM Almaden & Florida)
- BayesStore (Berkeley)
- Orion (Purdue)
- PrDB (Maryland)
- Also at UMass, Waterloo, Hong Kong, Florida State, Wisconsin, MIT, …

Projects I am involved in
- MayBMS (Cornell+Oxford)
  - Uncertainty-aware query languages and data representation models
- SPROUT = Scalable Query PROcessing on Uncertain Tables (Oxford)
  - Query engine that extends PostgreSQL backend

MayBMS and SPROUT are available at maybms.sourceforge.net.
Outline of this talk

- Two (discrete) probabilistic data models
  - U-relational databases: a complete but “hard” model
  - Tuple-independent databases: a restricted but “easier” model

- Query evaluation in probabilistic databases
  - The non-probabilistic case
  - Data complexity in the probabilistic case

- Exact query evaluation techniques
  - Evaluation of tractable conjunctive queries
    - using OBDDs (ordered binary decision diagrams)
    - using relational plans
  - Detection of large tractable query & data sub-instances

- Approximate query evaluation techniques with error guarantees
  - Monte Carlo techniques (FPRAS)
  - Incremental evaluation techniques based on lineage factorization
U-relational Probabilistic Databases

Syntax.
Probabilistic databases are relational databases where

- There is a finite set of independent random variables $X = \{x_1, \ldots, x_n\}$ with finite domains $\text{Dom}_{x_1}, \ldots, \text{Dom}_{x_n}$.
- Tuples are associated with lineage, i.e., conjunctions of atomic events of the form $x_i = a$ or $x_i \neq a$ where $x_i \in X$ and $a \in \text{Dom}_{x_i}$.
- There is a probability distribution over the assignments of each variable.

Semantics.

- Possible worlds defined by total assignments $\theta$ over $X$.
- The world defined by assignment $\theta$
  - consists of all tuples with condition $\phi$ such that $\theta(\phi) = \text{true}$.
  - has probability defined by the product of probabilities of each assignment in $\theta$.

This formalism can represent any discrete probability distribution over relational databases.
Example: Probabilistic Databases

Consider a simplified TPC-H scenario with customers (Cust) and orders (Ord):

<table>
<thead>
<tr>
<th>Cust</th>
<th>V1</th>
<th>P1</th>
<th>V2</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ckey</td>
<td>cname</td>
<td>x1</td>
<td>0.1</td>
<td>x3</td>
</tr>
<tr>
<td>1</td>
<td>Joe</td>
<td></td>
<td>0.9</td>
<td>x4</td>
</tr>
<tr>
<td>2</td>
<td>Dan</td>
<td></td>
<td>0.3</td>
<td>x4</td>
</tr>
<tr>
<td>3</td>
<td>Li</td>
<td></td>
<td>0.7</td>
<td>x5</td>
</tr>
<tr>
<td>4</td>
<td>Mo</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ord</th>
<th>okey</th>
<th>ckey</th>
<th>odate</th>
<th>V1</th>
<th>P1</th>
<th>V2</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1995-01-10</td>
<td>y1</td>
<td>0.1</td>
<td>x5</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1996-01-09</td>
<td>y2</td>
<td>0.2</td>
<td>x4</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>1994-11-11</td>
<td>y3</td>
<td>0.3</td>
<td>x3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

• Variables are Boolean (wlog); write x instead of \( x = 1 \), \( \overline{x} \) instead of \( x = 0 \).
• A pair \((V_i, P_i)\) states that the variable assignment given by \( V_i \) has the probability given by \( P_i \).
• Lineage can represent arbitrary correlations between tuples, eg,
  ▶ (1,Joe) and (3,Li) are independent: They use disjoint sets of variables.
  ▶ (1,Joe) and (2,Dan) are mutually exclusive: \( x_1 \) is either true or false.
Example: Probabilistic Databases

Consider the world $\mathcal{A}$ defined by a total assignment $\theta$:

- $x_1, x_2, y_1, y_2$ are true, and all other variables are false.

<table>
<thead>
<tr>
<th>Cust</th>
<th>Ord</th>
</tr>
</thead>
<tbody>
<tr>
<td>ckey</td>
<td>ckey</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Dan</td>
</tr>
<tr>
<td>3</td>
<td>Li</td>
</tr>
<tr>
<td>4</td>
<td>Mo</td>
</tr>
</tbody>
</table>
Example: Probabilistic Databases

Consider the world $\mathcal{A}$ defined by a total assignment $\theta$:

- $x_1, x_2, y_1, y_2$ are true, and all other variables are false.

<table>
<thead>
<tr>
<th>Cust</th>
<th></th>
<th>Ord</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ckey</td>
<td>ckey</td>
<td>odate</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1995-01-10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1996-01-09</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1994-11-11</td>
<td></td>
</tr>
</tbody>
</table>

The world $\mathcal{A}$ is as follows:

<table>
<thead>
<tr>
<th>Cust</th>
<th>Ord</th>
</tr>
</thead>
<tbody>
<tr>
<td>ckey</td>
<td>ckey</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Probability of $\mathcal{A} = \text{product of probabilities of the assignments in } \theta$:

$$Pr(\mathcal{A}) = Pr(\theta) = Pr(x_1) \cdot Pr(x_2) \cdot Pr(y_1) \cdot Pr(y_2) \cdot Pr(x_3) \cdot Pr(x_4) \cdot Pr(x_5) \cdot Pr(y_3).$$
Tuple-independent Probabilistic Databases

Tuple-independent: Tuples have independent lineage, or equivalently

- Each tuple $t$ is associated with a Boolean random variable $x_t$.
- Tuple $t$ is in the world defined by $\theta$ if $x_t = \text{true}$ holds in $\theta$.

<table>
<thead>
<tr>
<th></th>
<th>Cust</th>
<th></th>
<th>Ord</th>
<th></th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ckey</td>
<td>ckey</td>
<td>odate</td>
<td>V</td>
<td>P</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1995-01-10</td>
<td>y_1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1996-01-09</td>
<td>y_2</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1994-11-11</td>
<td>y_3</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1993-01-08</td>
<td>y_4</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>1995-08-15</td>
<td>y_5</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1996-12-25</td>
<td>y_6</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Query Evaluation in Probabilistic Databases

Subsumed by general probabilistic inference, which was investigated in AI for many years. Is there something left to do for the database community?
Query Evaluation in Probabilistic Databases

Subsumed by general probabilistic inference, which was investigated in AI for many years. Is there something left to do for the database community?

The database approach is based on two fundamental observations:
- the separation of (very large) data and (small and fixed) query, and
- the use of mature relational query engines to achieve scalability.
Query Evaluation in Probabilistic Databases

The MayBMS/SPROUT approach:

- Given probabilistic database \( T = \{A_1, \ldots, A_n\} \) and query \( q \).
- Under *possible world semantics*: \( q \) is evaluated in each possible world of \( T \).
- Is there \( \bar{q} \) such that \( \bar{q}(T) = \{q(A_1), \ldots, q(A_n)\} \)? [Imielinski&Lipski84]

![Diagram](image_url)

- Compute the probability of each distinct tuple in \( \bar{q}(T) \).

How hard is query evaluation in probabilistic databases?

- \( \bar{q}(T) \) can be computed in PTIME (wrt data complexity) for relational algebra queries and U-relational databases [AJK&O.08]
- Probability computation is in general very hard, but there are tractable cases
Example: Query Evaluation

Query asking for the dates of discounted orders shipped to customer 'Joe':

\[
Q(o\text{date}) :\neg \text{Cust(ckey,' Joe')}, \text{Ord(okey, ckey, odate)}, \text{Item(okey, disc, ckey), disc > 0}
\]

<table>
<thead>
<tr>
<th>odate</th>
<th>(V_c)</th>
<th>(P_c)</th>
<th>(V_o)</th>
<th>(P_o)</th>
<th>(V_i)</th>
<th>(P_i)</th>
<th>tuple probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995-01-10</td>
<td>(x_1)</td>
<td>0.1</td>
<td>(y_1)</td>
<td>0.1</td>
<td>(z_1)</td>
<td>0.1</td>
<td>0.1 \cdot 0.1 \cdot 0.1</td>
</tr>
<tr>
<td>1995-01-10</td>
<td>(x_1)</td>
<td>0.1</td>
<td>(y_1)</td>
<td>0.1</td>
<td>(z_2)</td>
<td>0.2</td>
<td>0.1 \cdot 0.1 \cdot 0.2</td>
</tr>
</tbody>
</table>

- Query \(\overline{Q}\) is \(Q\) changed so that the lineage of input tuples is copied in the answer tuples.
- Probability of distinct answer tuple (1995-01-10) is the probability of the associated lineage \(x_1 y_1 z_1 + x_1 y_1 z_2\).

Difficulty:
- The sets of satisfying assignments of any two clauses may overlap.
- It may require to iterate over its (exponentially many) satisfying assignments.
Dichotomy Property

Discussed here [Dalvi&Suciu07]:
- Conjunctive queries without self-joins (CQ\(^1\)) on
- Tuple-independent databases.

The data complexity of any CQ\(^1\) query is either FP or \#P-hard.

- \#P = class of functions \(f(x)\) for which there exists a PTIME non-deterministic Turing machine \(M\) such that \(f(x) = \) number of accepting computations of \(M\) on input \(x\).
- FP = class of functions that can be solved by a deterministic Turing machine in PTIME. These functions can have any output, not only true/false.

Further tractability results not discussed here:
- Dichotomy for conjunctive queries with self-joins [Dalvi&Suciu07b]
- Tractable queries with inequalities (\(<, \leq, \neq\)) [O.&Huang08, O.&Huang09]
- Extension to the block-independent disjoint model follow easily [DRS07]
All tractable CQ\(^1\) queries are hierarchical

A query is *hierarchical* if for any two non-head variables, either their sets of subgoals are disjoint, or one set is contained in the other.

\[ Q(odate) :- \text{Cust}(ckey,'Joe'), \text{Ord}(okey, ckey, odate), \text{Item}(okey, disc, ckey), disc > 0. \]

is hierarchical; also without odate as head variable.

\[ \text{subgoals}(disc)=\{\text{Item}\}, \text{subgoals}(okey)=\{\text{Ord, Item}\}, \text{subgoals}(ckey)=\{\text{Cust, Ord, Item}\}. \]

It holds that \(\text{subgoals}(disc) \subseteq \text{subgoals}(okey) \subseteq \text{subgoals}(ckey)\).
Exact Query Evaluation
Exact Query Evaluation using SPROUT

Cast the query evaluation problem as a decision diagram construction problem.

- Given a query $q$ and a probabilistic database $D$, each distinct tuple $t \in q(D)$ is associated with a DNF expression $\phi_t$.

- Probability of $t$ is probability of $\phi_t$.

- Compile $\phi_t$ into an equivalent binary decision diagram (BDD).

- Probability of $\phi_t$ is then the probability of its BDD.

- SPROUT employs secondary-storage techniques for BDD construction and probability computation.
BDDs

- Commonly used to represent compactly large Boolean expressions.

- Idea: Decompose Boolean expressions using variable elimination and avoid redundancy in the representation.
  Variable elimination by Shannon’s expansion: $\phi = x \cdot \phi \mid x + \overline{x} \cdot \phi \mid \overline{x}$.

- Supports linear-time probability computation.

  $$Pr(\phi) = Pr(x \cdot \phi \mid x + \overline{x} \cdot \phi \mid \overline{x})$$
  $$= Pr(x \cdot \phi \mid x) + Pr(\overline{x} \cdot \phi \mid \overline{x})$$
  $$= Pr(x) \cdot Pr(\phi \mid x) + Pr(\overline{x}) \cdot Pr(\phi \mid \overline{x})$$

Ordered BDDs (OBDDs):
- Variable order $\pi$ = order of variable eliminations;
  the same variable order on all root-to-leaf paths $\Rightarrow$ ordered BDDs (OBDDs)
- An OBDD for $\phi$ is uniquely identified by the pair $(\phi, \pi)$. 
Compilation example

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>V_r</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_1</td>
<td>b_1</td>
<td>x_1</td>
</tr>
<tr>
<td></td>
<td>a_2</td>
<td>b_1</td>
<td>x_2</td>
</tr>
<tr>
<td></td>
<td>a_2</td>
<td>b_2</td>
<td>x_3</td>
</tr>
<tr>
<td></td>
<td>a_3</td>
<td>b_3</td>
<td>x_4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>C</th>
<th>V_s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_1</td>
<td>c_1</td>
<td>y_1</td>
</tr>
<tr>
<td></td>
<td>a_1</td>
<td>c_2</td>
<td>y_2</td>
</tr>
<tr>
<td></td>
<td>a_2</td>
<td>c_1</td>
<td>y_3</td>
</tr>
<tr>
<td></td>
<td>a_4</td>
<td>c_2</td>
<td>y_4</td>
</tr>
</tbody>
</table>

\[ q : R(A, B), S(A, C) \]

<table>
<thead>
<tr>
<th>V_r</th>
<th>V_s</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>y_1</td>
</tr>
<tr>
<td>x_1</td>
<td>y_2</td>
</tr>
<tr>
<td>x_2</td>
<td>y_3</td>
</tr>
<tr>
<td>x_3</td>
<td>y_3</td>
</tr>
</tbody>
</table>

Query \( q \) has lineage \( \phi = x_1y_1 + x_1y_2 + x_2y_3 + x_3y_3 \).
Assume variable order: \( \pi = x_1y_1y_2x_2x_3y_3 \).
Task: Construct the OBDD \( (\phi, \pi) \).
Compilation example

Task: Construct OBDD \((\phi, \pi)\), where

- \(\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3\) and \(\pi = x_1 y_1 y_2 x_2 x_3 y_3\).

Step 1: Eliminate variable \(x_1\) in \(\phi\).
Compilation example

Task: Construct OBDD ($\phi, \pi$), where

- $\phi = x_1y_1 + x_1y_2 + x_2y_3 + x_3y_3$ and $\pi = x_1y_1y_2x_2x_3y_3$.

Step 2: Eliminate variable $y_1$. 

![Diagram showing the OBDD construction process with nodes and edges representing variables and their relationships. The diagram includes nodes labeled $x_1$, $y_1$, and $1$, with edges illustrating the flow of variables.]
Compilation example

Task: Construct OBDD $(\phi, \pi)$, where

- $\phi = x_1y_1 + x_1y_2 + x_2y_3 + x_3y_3$ and $\pi = x_1y_1y_2x_2x_3y_3$.

Step 3: Eliminate variable $y_2$.

Some leaves have the same expressions $\Rightarrow$ Represent them only once!
Compilation example

Task: Construct OBDD $(\phi, \pi)$, where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 4: Merge leaves with the same expressions.
Compilation example

Task: Construct OBDD $((\phi, \pi))$, where

- $\phi = x_1y_1 + x_1y_2 + x_2y_3 + x_3y_3$ and $\pi = x_1y_1y_2x_2x_3y_3$.

Step 5: Eliminate variable $x_2$. 

![Diagram of OBDD](image-url)
Compilation example

Task: Construct OBDD \((\phi, \pi)\), where
\[
\phi = x_1y_1 + x_1y_2 + x_2y_3 + x_3y_3 \quad \text{and} \quad \pi = x_1y_1y_2x_2x_3y_3.
\]
Step 6: Replace \(y_3 + x_3y_3\) by \(y_3\).
Task: Construct OBDD \((\phi, \pi)\), where

\[
\phi = x_1y_1 + x_1y_2 + x_2y_3 + x_3y_3 \quad \text{and} \quad \pi = x_1y_1y_2x_2x_3y_3.
\]

Step 7: Eliminate variable \(x_3\).
Compilation example

Task: Construct OBDD \((\phi, \pi)\), where

\[ \phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3 \] and \[ \pi = x_1 y_1 y_2 x_2 x_3 y_3. \]

Step 8: Merge leaves with the same expression \(y_3\).
Compilation example

Task: Construct OBDD $(\phi, \pi)$, where

- $\phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3$ and $\pi = x_1 y_1 y_2 x_2 x_3 y_3$.

Step 9: Eliminate variable $y_3$. 

Step 9: Eliminate variable $y_3$. 

\begin{align*}
\phi &= x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3 \\
\pi &= x_1 y_1 y_2 x_2 x_3 y_3
\end{align*}
Compilation example

Task: Construct OBDD \((\phi, \pi)\), where

\[ \phi = x_1 y_1 + x_1 y_2 + x_2 y_3 + x_3 y_3 \quad \text{and} \quad \pi = x_1 y_1 y_2 x_2 x_3 y_3. \]

Step 10 (final): Merge leaves with the same expression (0 or 1).
Compilation example: Summing Up

OBDD \((\phi, \pi)\) has size bounded in the number of literals in \(\phi\).
(exactly one node per variable in \(\phi\) in our example)

Questions

1. Is this property shared by the BDDs of many queries?
2. Can we efficiently construct such succinct BDDs?
Compilation example: Summing Up

OBDD \((\phi, \pi)\) has size bounded in the number of literals in \(\phi\).
(exactly one node per variable in \(\phi\) in our example)

Questions

1. Is this property shared by the BDDs of many queries?
2. Can we efficiently construct such succinct BDDs?

The answer is in the affirmative for both questions!
Tractable Queries and Succinct BDDs

For any hierarchical query $q$ and database $D$, $\forall t \in q(D)$, and lineage $\phi_t$,

- There is a variable order $\pi$ computable in time $O(|\phi_t| \cdot \log^2 |\phi_t|)$ such that
- The OBDD $(\phi_t, \pi)$ has size $O(|\text{Vars}(\phi_t)|)$ and can be computed in time $O(|\phi_t| \cdot \log |\phi_t|)$.
- $\phi_t$ can be factorized into read-once functions! [O.&Huang08]

BDD construction in polynomial time for a class of tractable conjunctive queries with inequalities. [O.&Huang09]

Good variable orders can be *statically* derived from the query structure!
Static Query Analysis: Query Signatures

Query signatures for $TQ$ queries capture
- the structures of queries and
- the one/many-to-one/many relationships between the query tables;
- variable orders for succinct BDDs representing compiled lineage!

$$A \quad \text{R}(A,B) \quad \text{S}(A,C)$$

Query $q :: R(A,B), S(A,C)$ has signature $(R*S*)*$. 
- There may be several $R$-tuples with the same $A$-value, hence $R^*$
- There may be several $S$-tuples with the same $A$-value, hence $S^*$
- $R$ and $S$ join on $A$, hence $R*S^*$
- There may be several $A$-values in $R$ and $S$, hence $(R*S^*)^*$

Variable orders captured by $(R*S*)^*$ ($x_i$’s are from $R$, $y_j$’s are from $S$): 
$\{[x_1(y_1y_2)][(x_2x_3)y_3]\}$, $\{[(x_2x_3)y_3][x_1(y_1y_2)]\}$, $\{[y_3(x_3x_2)][x_1(y_2y_1)]\}$, etc.
Query Rewriting under Functional Dependencies (FDs)

FDs on tuple-independent databases can help deriving better query signatures.

Given a set of FDs \( \Sigma \) and a conjunctive query of the form

\[
Q = \pi_{A_0}(\sigma_{\phi}(R_1(A_1) \bowtie \ldots \bowtie R_n(A_n)))
\]

where \( \phi \) is a conjunction of unary predicates. Let \( \Sigma_0 = CLOSURE_\Sigma(A_0) \).

Then, the Boolean query

\[
\pi_\emptyset(\sigma_{\phi}(R_1(CLOSURE_\Sigma(A_1) - \Sigma_0) \bowtie \ldots \bowtie R_n(CLOSURE_\Sigma(A_n) - \Sigma_0)))
\]

is called the **FD-reduct** of \( Q \) under \( \Sigma \). \[O.\&HK09\]

If there is a sequence of chase steps under \( \Sigma \) that turns \( Q \) into a hierarchical query, then the fixpoint of the chase (the FD-reduct) is hierarchical.
Importance of FD-reducts

The signature of $Q$’s FD-reduct captures the structure of $Q$’s lineage.

Two relevant cases

1. Intractable queries may admit tractable FD-reducts.

Under $X \rightarrow Y$, the hard query $Q : R(X), S(X, Y), T(Y)$ admits the hierarchical FD-reduct $Q' : R(X, Y), S(X, Y), T(Y)$ with signature $((RS)^* T)^*$. 

2. FD-reducts have more precise query signatures.

In the presence of keys $ckey$ and $okey$, the query $Q(odate) : Cust(ckey, cname), Ord(okey, ckey, odate), Item(okey, disc, ckey)$ with signature $(Cust^*(Ord^*Item^*))^*$ rewrites into

$Q' : Cust(ckey, cname), Ord(okey, ckey, cname), Item(okey, disc, ckey, cname)$ with signature $(Cust(Ord Item^*)^*)^*$. 
Case Study: TPC-H Queries

Considered the conjunctive part of each of the 22 TPC-H queries

- Boolean versions (B)
- with original selection attributes, but without aggregates (O)

Hierarchical in the absence of key constraints

- 8 queries (B)
- 13 queries (O)

Hierarchical in the presence of key constraints

- 8+4 queries (B)
- 13+4 queries (O)

In-depth study at

http://www.comlab.ox.ac.uk/people/dan.olteanu/papers/icde09queries.html
Secondary-storage Query Evaluation

Query evaluation approached in two logically-independent steps
1. Compute query answer using a relational query plan of your choice.
2. Compute probabilities of each distinct answer (or temporary) tuple.

Probability computation supported by a new aggregation operator that can
- blend itself in any relational query plan,
- be placed on top of the query plan, or partially pushed down past joins.
- compute in parallel different fragments of the BDD for the lineage without materializing the BDD.

Our aggregation operator is a sequence of
- aggregation steps. Effect on query signature: $\alpha^* \rightarrow \alpha$
- propagation steps. Effect on query signature: $\alpha\beta \rightarrow \alpha$
Example of Probability Computation

How to proceed?

1. **Sort query answer by** \((V_r, V_s)\).
   - Initial signature: \((R^*S^*)^*\)

2. Apply aggregation step \(S^* \rightarrow S\).
   - New signature: \((R*S)^*\)

3. Apply aggregation step \(R^* \rightarrow R\).
   - New signature: \((RS)^*\)

4. Apply propagation step \(RS \rightarrow R\).
   - New signature: \(R^*\)

5. Apply aggregation step \(R^* \rightarrow R\).
   - New signature: \(R\)

\[
\begin{array}{c|cc}
q & R(A, B), S(A, C) \\
\hline
V_r & V_s \\
\hline
x_1 & y_1 \\
x_1 & y_2 \\
x_2 & y_3 \\
x_3 & y_3 \\
\end{array}
\]
Example of Probability Computation

How to proceed?

1. Sort query answer by \((V_r, V_s)\).
   Initial signature: \((R^*S^*)^*\)

2. **Apply aggregation step** \(S^* \rightarrow S\).
   New signature: \((R^*S)^*\)

3. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \((RS)^*\)

4. Apply propagation step \(RS \rightarrow R\).
   New signature: \(R^*\)

5. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \(R\)

### Table

| \(q \vdash R(A, B), S(A, C)\) |
|---|---|
| \(V_r\) | \(V_s\) |
| \(x_1\) | \(y_1 + y_2\) |
| \(x_2\) | \(y_3\) |
| \(x_3\) | \(y_3\) |
Example of Probability Computation

How to proceed?

1. Sort query answer by \((V_r, V_s)\).
   Initial signature: \((R^* S^*)^*\)

2. **Apply aggregation step** \(S^* \rightarrow S\).
   New signature: \((R^* S)^*\)

3. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \((R S)^*\)

4. Apply propagation step \(R S \rightarrow R\).
   New signature: \(R^*\)

5. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \(R\)

\[
q : - R(A, B), S(A, C)
\]

<table>
<thead>
<tr>
<th></th>
<th>(V_r)</th>
<th>(V_s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(y'_1)</td>
<td></td>
</tr>
<tr>
<td>(x_2)</td>
<td>(y_3)</td>
<td></td>
</tr>
<tr>
<td>(x_3)</td>
<td>(y_3)</td>
<td></td>
</tr>
</tbody>
</table>
Example of Probability Computation

\[ q := R(A, B), S(A, C) \]

<table>
<thead>
<tr>
<th>( V_r )</th>
<th>( V_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( y_1' )</td>
</tr>
<tr>
<td>( x_2 + x_3 )</td>
<td>( y_3 )</td>
</tr>
</tbody>
</table>

How to proceed?

1. Sort query answer by \((V_r, V_s)\).
   Initial signature: \((R^*S^*)^*\)

2. Apply aggregation step \(S^* \rightarrow S\).
   New signature: \((R^*S)^*\)

3. **Apply aggregation step** \(R^* \rightarrow R\).
   New signature: \((RS)^*\)

4. Apply propagation step \(RS \rightarrow R\).
   New signature: \(R^*\)

5. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \(R\)
Example of Probability Computation

How to proceed?

1. Sort query answer by \((V_r, V_s)\).
   Initial signature: \((R^*S^*)^*\)

2. Apply aggregation step \(S^* \rightarrow S\).
   New signature: \((R^*S)^*\)

3. **Apply aggregation step** \(R^* \rightarrow R\).
   New signature: \((RS)^*\)

4. Apply propagation step \(RS \rightarrow R\).
   New signature: \(R^*\)

5. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \(R\)
Example of Probability Computation

$q := R(A, B), S(A, C)$

<table>
<thead>
<tr>
<th>$V_r$</th>
<th>$x_1 y_1'$</th>
<th>$x_2 y_3$</th>
</tr>
</thead>
</table>

How to proceed?

1. Sort query answer by $(V_r, V_s)$.
   Initial signature: $(R^* S^*)^*$

2. Apply aggregation step $S^* \rightarrow S$.
   New signature: $(R^* S)^*$

3. Apply aggregation step $R^* \rightarrow R$.
   New signature: $(RS)^*$

4. **Apply propagation step** $RS \rightarrow R$.
   New signature: $R^*$

5. Apply aggregation step $R^* \rightarrow R$.
   New signature: $R$
Example of Probability Computation

How to proceed?

1. Sort query answer by \((V_r, V_s)\).
   Initial signature: \((R^*S^*)^*\)

2. Apply aggregation step \(S^* \rightarrow S\).
   New signature: \((R*S)^*\)

3. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \((RS)^*\)

4. **Apply propagation step** \(RS \rightarrow R\).
   New signature: \(R^*\)

5. Apply aggregation step \(R^* \rightarrow R\).
   New signature: \(R\)
Example of Probability Computation

How to proceed?

1. Sort query answer by \((V_r, V_s)\).
   - Initial signature: \((R S^*)^*\)

2. Apply aggregation step \(S^* \rightarrow S\).
   - New signature: \((R S)^*\)

3. Apply aggregation step \(R^* \rightarrow R\).
   - New signature: \((RS)^*\)

4. Apply propagation step \(RS \rightarrow R\).
   - New signature: \(R^*\)

5. **Apply aggregation step** \(R^* \rightarrow R\).
   - New signature: \(R\)
Example of Probability Computation

$q :: R(A, B), S(A, C)$

Return the probability of $x''''$. How to proceed?

1. Sort query answer by $(V_r, V_s)$.
   Initial signature: $(R*S^*)^*$

2. Apply aggregation step $S^* \rightarrow S$.
   New signature: $(R*S)^*$

3. Apply aggregation step $R^* \rightarrow R$.
   New signature: $(RS)^*$

4. Apply propagation step $RS \rightarrow R$.
   New signature: $R^*$

5. **Apply aggregation step** $R^* \rightarrow R$.
   New signature: $R$
Can we leverage existing results on BDD construction?

- Generic AI compilation techniques construct BDDs whose sizes are exponential in the treewidth of the lineage [Huang&Darwiche01]

- Conjunctive queries do generate lineage of unbounded treewidth.
  - The product query $Q \leftarrow R(X), S(Y)$ generates lineage that has a clause for each pair of random variables of $R$ and $S \Rightarrow$ unbounded treewidth.

- Reconciling the two techniques [Jha,O.&Suciu10]:
  - Partition input query+data into a tractable subinstance and a (usually much smaller) hard subinstance.
  - Tractable subinstance: largest sub-relation satisfying functional dependency
  - Apply scalable database-specific techniques to the tractable part and generic AI compilation techniques to the hard part.
Our previous examples considered lazy plans, where

- probability computation done after the computation of answer tuples
- unrestricted search space for good query plans
- especially desirable when join conditions are selective (e.g., TPC-H)!

\[
(Cust^* (Ord^* Item^*)^*)^* \quad \pi_{odate} \quad \nabla_{ckey, okey} \quad \nabla_{ckey} \quad \sigma_{disc > 0} \quad \sigma_{cname = 'Joe'} \quad \sigma_{ckey} \quad Ord \quad Item \quad Cust
\]

BUT, we can push down probability computation!
Query Optimization: Types of Query Plans

**Eager** plans discard duplicates and compute probabilities on each temporary table.

MystiQ’s safe plans are special cases of eager plans!

- mirror the hierarchical structure of the query signature
- probability computation restricts join ordering!
- suboptimal join ordering, which is more costly than probability computation
Approximate Query Evaluation
Approximate Query Evaluation using Monte Carlo

Using Karp-Luby FPRAS [Karp&Luby83], [Graedel&Gurevitch&Hirsch98]

Input: Boolean formula in DNF $\phi = C_1 + \cdots + C_m$ with variables $V(\phi)$

$Cnt \leftarrow 0; S \leftarrow Pr(C_1) + \cdots + Pr(C_m)$

repeat $N$ times

randomly choose $1 \leq i \leq m$ with probability $Pr(C_i)/S$

randomly choose a total valuation $\lambda$ over $V(\phi)$ such that $\lambda(C_i) = \text{true}$

if $\forall 1 \leq j < i : \lambda(C_j) = \text{false}$ then $Cnt = Cnt + 1$

$P = Cnt/N \times S/2^{|V(\phi)|}$

return $P/\approx Pr(\phi)$

If $N \geq (1/m) \times (4 \ln(2/\delta)/\epsilon^2)$ then $Pr[| P/Pr(\phi) - 1 | > \epsilon] < \delta$.

- Slightly modified algorithms are used in MayBMS, MystiQ, and MCDB.
- Veeeery sloooow in practice: SPROUT query plans are about two orders of magnitude faster that the optimized Monte Carlo.
Approximate Evaluation with **SPROUT**

- Monte Carlo simulations are very powerful and generic
  - only require sampling the formula, no knowledge of its structure

- Why not exploit the structure of the input formula? [O.&HK10]
  - incrementally compile it into an equivalent decomposed form that allows for efficient probability computation
  - in practice, good approximations are obtained after a few decomposition steps
  - Topic of my ICDE10 talk tomorrow :-)}
Thanks!
Literature on Probabilistic Relational Data


Literature on Probabilistic Relational Data


