LOCAL STRUCTURE AND DETERMINISM IN PROBABILISTIC DATABASES

Theodoros Rekatsinas, Amol Deshpande, Lise Getoor
Motivation

- Probabilistic databases store, manage and query uncertain data

- Numerous applications generate correlated data (e.g., sensor networks, data integration etc.)

Problem

Scaling query evaluation in probabilistic databases with correlated data
Contributions in a nutshell

Challenges in correlated probabilistic DBs:

1. How to represent correlations?
   - Annotated Arithmetic Circuit (AAC)
     - Exploits global and local structure of distribution

2. How to evaluate queries efficiently?
   - Online merging
     - Transform queries into AACs; merge with DB AACs

---

Our Approach

Baselines (red, blue)

<table>
<thead>
<tr>
<th>Evaluation Time (sec)</th>
<th>TPC-H Scale factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^1</td>
<td>0.01</td>
</tr>
<tr>
<td>10^0</td>
<td>0.05</td>
</tr>
<tr>
<td>10^-1</td>
<td>0.1</td>
</tr>
<tr>
<td>10^-2</td>
<td></td>
</tr>
<tr>
<td>10^-3</td>
<td></td>
</tr>
</tbody>
</table>

TPC-H Query B3

VE - indep. DB
VE(ADD) - indep. DB
AAC - indep. DB
VE - cor. DB
VE(ADD) - cor. DB
AAC - cor. DB
Outline

• Background

• Arithmetic Circuits in probabilistic DBs

• Experiments

• Conclusion and future work
Outline

• Background

• Arithmetic Circuits in probabilistic DBs

• Experiments

• Conclusion and future work
Example correlated probabilistic DB

**Database**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>'m'</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>'n'</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>'p'</td>
</tr>
</tbody>
</table>

**Factor graph**

**Query**

*Query:* \( q(c):= s(a,b), t(b,c) \)

*Lineage:* \( \ell = (s1 \land t1) \lor (s2 \land t1) \)

**Query Evaluation \( \cong \) Probabilistic inference**

Inference has exponential complexity
Speeding inference

Compilation: separate inference into two phases
Offline: compile distribution into a data structure
Online: use data structure to answer queries

Goal: Push work to offline phase to optimize online inference time

Exploit local and global structure

• **Global structure**: Model topology (treewidth)
• **Local structure**: Exhibited in model parameters
  • Determinism
  • Context specific independence (CSI)
Determinism

Example Bayesian network [Darwiche, UAI `06]

If Battery Power = Dead, then Lights = OFF

<table>
<thead>
<tr>
<th>Battery Power</th>
<th>Lights</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>0.99</td>
</tr>
<tr>
<td>WEAK</td>
<td>0.20</td>
</tr>
<tr>
<td>DEAD</td>
<td>0</td>
</tr>
</tbody>
</table>
Context specific independence

Example Bayesian network [Darwiche, UAI `06]
Context specific independence

Example Bayesian network [Darwiche, UAI `06]
### Overview of correlation representations

<table>
<thead>
<tr>
<th>Data correlations</th>
<th>Query correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>factor graphs</strong> [Sen et al., `07]</td>
<td>augmented factor graphs (variable elimination)</td>
</tr>
<tr>
<td><strong>pc-tables</strong> [Green et al., `06]</td>
<td>lineage, OBDDs [Olteanu et al., <code>08], D-trees [Olteanu et al., </code>10]</td>
</tr>
<tr>
<td></td>
<td>- OBDDs, D-trees exploit <strong>local structure</strong></td>
</tr>
<tr>
<td><strong>junction trees</strong> [Kanagal et al., `10]</td>
<td>lineage</td>
</tr>
<tr>
<td>- <strong>compilation</strong></td>
<td></td>
</tr>
<tr>
<td>- exploit <strong>global</strong> structure</td>
<td></td>
</tr>
<tr>
<td><strong>arithmetic circuits</strong> [Darwiche, `03],</td>
<td></td>
</tr>
<tr>
<td><strong>AACs</strong> [this paper]</td>
<td></td>
</tr>
<tr>
<td>- <strong>compilation</strong></td>
<td></td>
</tr>
<tr>
<td>- exploit <strong>global</strong> and <strong>local</strong> structure</td>
<td></td>
</tr>
</tbody>
</table>

*[this paper]*
Example arithmetic circuit

Factor graph

<table>
<thead>
<tr>
<th>A</th>
<th>( f_a(.) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( f_{ab}(.) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MLF \((a_0 = \text{false}, a_1 = \text{true})\):

\[
\lambda_{a_0} \lambda_{b_0} \theta_{a_0} \theta_{b_0a_0} + \lambda_{a_0} \lambda_{b_1} \theta_{a_0} \theta_{b_1a_0} + \\
\lambda_{a_1} \lambda_{b_0} \theta_{a_1} \theta_{b_0a_1} + \lambda_{a_1} \lambda_{b_1} \theta_{a_1} \theta_{b_1a_1}
\]

Factor graph
Querying an arithmetic circuit

Arithmetic circuit

\[ \lambda a_0 
\]

\[ + \]

\[ * \]

\[ \lambda a_1 \]

\[ \lambda a_0 \]

\[ 0.6 \]

\[ + \]

\[ * \]

\[ \lambda a_1 \]

\[ 0.4 \]

\[ * \]

\[ * \]

\[ \lambda b_0 \]

\[ \lambda b_1 \]

\[ 0.3 \]

\[ \lambda b_0 \]

\[ 0.7 \]

\[ \lambda b_1 \]

Probability of evidence query

Pr(A = false, B = false)

- Set \( \lambda a_0 = 1, \lambda a_1 = 0, \lambda b_0 = 1, \lambda b_1 = 0 \)

- Evaluate the circuit to get
  \[ \text{Pr}(A = \text{false}, B = \text{false}) = 0.18 \]
Outline

• Background

• Arithmetic Circuits in probabilistic DBs

• Experiments

• Conclusion and future work
Overview of our Framework

**Problem**
No online updates

**Solution (bad)**
Repeat entire compilation process (expensive)

**Solution (good)**
- **Annotated Arithmetic Circuits (AACs):** Add variable annotations to summation nodes, value annotations to edges
- **Merge AACs** during query evaluation

Evaluate Probability
Arithmetic Circuits
Overview of our Framework

Offline Pre-processing
- DB Factor Graph
- Compile
- DB Annotated Arithmetic Circuits

Data

Online Query Evaluation
- Query
- Lineage Formulas
- Result Tuple Annotated Arithmetic Circuit
- Merge Annotated Arithmetic Circuits
- Final AC
- Evaluate Probability

For each result tuple

Convert to Annotated Arithmetic Circuits
Example – Annotated Arithmetic Circuit

<table>
<thead>
<tr>
<th>A</th>
<th>( f_a(.) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( f_{ab}(.) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \begin{align*} + A & + B + B \\ \lambda a_0 & \ast \ast \lambda a_1 \\ 0.6 & + B 0.4 \\ 0 & 1 0 & 1 \\ \ast & \ast \ast & \ast \\ \lambda b_0 & 0.3 0.7 0 \\ \lambda b_1 & 
\]
Overview of our Framework

Offline Pre-processing

- DB Factor Graph
- Data
- Compile
- DB Annotated Arithmetic Circuits

Online Query Evaluation

- Query
- Lineage Formulas
- Convert to Annotated Arithmetic Circuits
- For each result tuple
- Result Tuple Annotated Arithmetic Circuit
- Merge Annotated Arithmetic Circuits
- Final AC
- Evaluate Probability
Compiling Factor Graphs

- Extend compilation technique based on VE [Chavira et al., `07]
  - Uses Algebraic Decision Diagrams (ADDs)
  - Add variable annotations to summation nodes

- VE requires a variable ordering
  - Output AACs follow the reverse VE ordering

- Collection of disjoint ordered sets of variables
  - One for each disconnected part of the factor graph
**Example**

**Database**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>‘m’</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>‘n’</td>
<td>1</td>
</tr>
</tbody>
</table>

**T**

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>‘p’</td>
</tr>
</tbody>
</table>

**Factor graph**

VE Ordering:

\[ O = s_1, s_2, t_1 \]

Partial Orderings:

\[ O_1 = s_1, O_2 = s_2, t_1 \]
Overview of our Framework

Offline Pre-processing

DB Factor Graph

Data

Compile

DB Annotated Arithmetic Circuits

Online Query Evaluation

Query

Lineage Formulas

Convert to Annotated Arithmetic Circuits

For each result tuple

Result Tuple Annotated Arithmetic Circuit

Merge Annotated Arithmetic Circuits

Final AC

Evaluate Probability

Compile

DB Annotated Arithmetic Circuits
Compiling Lineage

- Use Ordered Binary Decision Diagrams [Olteanu et al., 2008]
  - Lineage consists of only deterministic (AND/OR) factors
  - More efficient compilation

Compilation algorithm (sketch)
1. Choose a total variable ordering
2. Compile lineage formula to an OBDD
3. Construct an AAC from an OBDD
Getting a total ordering

• OBDD construction requires a total variable ordering
  • Finding an optimal variable ordering is NP-hard

Designing a heuristic (ideas):

1. Connected variables in lineage must appear together [Ebendt et al., 2005]
2. More influential variables must appear sooner in the ordering [Ebendt et al., 2005]
3. Total ordering must respect partial ordering constraints
   • Necessary for merging
4. Variables in a constraint must be kept together
   • Facilitates caching during merging
Variable Ordering under Partial Order Constraints

Algorithm (Goal: Find total order of variables)
1. Score variables by their position in the partial orderings
2. Group random variables that are connected in the lineage
3. Use scores to get a total variable ordering
Overview of our Framework

Offline Pre-processing

DB Factor Graph

Data

Compile

DB Annotated Arithmetic Circuits

Online Query Evaluation

Query

Lineage Formulas

Result Tuple Annotated Arithmetic Circuit

Merge Annotated Arithmetic Circuits

Final AC

Evaluate Probability

For each result tuple

Convert to Annotated Arithmetic Circuits
Merging Annotated Arithmetic Circuits

Algorithm:
- Traverse the query and DB AACs simultaneously
- At each step merge exactly two AACs (rooted at summation nodes)
- If the result can be computed immediately, done
- Else: choose one root and recurse on its sub-circuits
  - Choose the root with the variable that appears first in the total ordering

Complexity: $O\left(m \sum_{i \in A} s_i \right)$

$m$: size of lineage-AAC, $A$: set of relevant database AACs, $s_i$: size of i-th AAC in $A$
Outline

• Background

• Arithmetic Circuits in probabilistic DBs

• Experiments

• Conclusion and future work
Experimental Evaluation

- Experimental evaluation using the TPC-H benchmark
  - Tuple independence
  - Schema level correlations
  - Multiple scale factors (0.001, 0.005, 0.01, 0.05, 0.1)

- Evaluation on queries with both safe and not-safe evaluation plans
  - Boolean versions of TPC-H queries Q2, Q3, Q5, Q6, Q8 and Q16 without top-level aggregates

- Comparison to:
  - Variable elimination with Conditional Probability Tables [Sen et al., `07]
  - Variable elimination with Algebraic Decision Diagrams
    - Captures determinism and context specific independence
Compilation Time

Database Compilation Time

Compilation Time (sec)

TPC-H Scale factor

Independent DB

Correlated DB
Evaluation Time

TPC-H Query B3

TPC-H Query B8

Evaluation Time (sec)

TPC-H Scale factor

Missing points: Evaluation time exceeded 100 sec
Evaluation Time

Longer lineage formulas

TPC-H Query B6

TPC-H Query B16

Evaluation Time (sec)

TPC-H Scale factor

Missing points: Evaluation time exceeded 100 sec
Micro-benchmarking

TPC-H Query B3 evaluation breakdown

- Merging Phase
- Lineage-to-AAC
- Variable Ordering
- Lineage Creation

TPC-H Query B16 evaluation breakdown

- Merging Phase
- Lineage-to-AAC
- Variable Ordering
- Lineage Creation

- Longer lineage formula
- Not optimized (blue)
Outline

• Background

• Arithmetic Circuits in probabilistic DBs

• Experiments

• Conclusion and future work
Conclusion and future work

Conclusion
• Exploit compilation, global and local structure for efficient exact query evaluation
  • At least one order of magnitude improvement
• Introduced Annotated Arithmetic Circuits
  • New merging algorithm to support updates
  • Of independent interest to probabilistic reasoning

Open problems
• Lifted Inference in Arithmetic Circuits
  • isomorphic parts of an arithmetic circuit with respect to operations and θ variables?
• Approximate inference in Arithmetic Circuits
  • How to incorporate approximations into the framework?