The Relational Data Borg is Learning: Part Deux

Dan Olteanu
University of Zurich

VLDB 2020 Keynote
Virtual Tokyo, Sept 1, 2020
Where We Are

Covered so far:

- Relational data is ubiquitous
- **Structure-agnostic learning** is the state of the art
- **Structure-aware learning** can be much faster
- Idea 1: Turn learning into a DB workload challenge

To come: Exploit structure of the data and problem

- Idea 2: Lower the asymptotics
- Idea 3: Lower the constant factors
Idea 2: Exploit Problem Structure to Lower Complexity

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Place</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lewis</td>
<td>Narnia</td>
<td>Eagle &amp; Child</td>
<td>4</td>
</tr>
<tr>
<td>Lewis</td>
<td>Narnia</td>
<td>Eagle &amp; Child</td>
<td>3</td>
</tr>
<tr>
<td>Tolkien</td>
<td>Hobbit</td>
<td>Eagle &amp; Child</td>
<td>4</td>
</tr>
<tr>
<td>Tolkien</td>
<td>Hobbit</td>
<td>Eagle &amp; Child</td>
<td>3</td>
</tr>
</tbody>
</table>

*Note: The diagram shows a clamped list with entries that are factorized into 'Narnia' and 'Eagle & Child', and their corresponding orders.*
Algebraic structure: (semi)rings \((\mathcal{R}, +, \ast, 0, 1)\)

- **Distributivity law** → **Factorisation**
  - Factorised Databases \([\text{VLDB}'12+'13,TODS'15,\text{SIGREC}'16]\)
  - Factorised Machine Learning \([\text{SIGMOD}'16+'19,\text{DEEM}'18,\text{PODS}'18+'19, \text{TODS}'20]\)

- **Additive inverse** → **Uniform treatment of updates**
  - Factorised Incremental Maintenance \([\text{SIGMOD}'18+'20]\)

- **Sum-Product abstraction** → **Same processing for distinct tasks**
  - DB queries, Covariance matrix, PGM inference, Matrix chain multiplication \([\text{SIGMOD}'18+'19]\)
Combinatorial structure: query width and data degree measures

- **Width measure $w$ for FEQ → Low complexity $\tilde{O}(N^w)$**
  
factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  
  worst-case optimal size and time for factorised joins
  
  [ICDT'12+’18, TODS’15, PODS’19, TODS’20]

- **Degree → Adaptive processing depending on high/low degrees**
  
worst-case optimal incremental maintenance
  
  [ICDT’19a, PODS’20]

  evaluation of queries with negated relations of bounded degree
  
  [ICDT’19b]

- **Functional dependencies → Learn simpler, equivalent models**
  
  reparameterisation of polynomial regression models and factorisation machines
  
  [PODS’18, TODS’20]
Factorised Query Evaluation

⇓

Time/Size Improvement
<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Steve</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dish</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>burger</td>
<td>patty</td>
</tr>
<tr>
<td>onion</td>
<td></td>
</tr>
<tr>
<td>bun</td>
<td></td>
</tr>
<tr>
<td>bun</td>
<td></td>
</tr>
<tr>
<td>onion</td>
<td></td>
</tr>
<tr>
<td>sausage</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>sausage</td>
<td>4</td>
</tr>
</tbody>
</table>
A Burgers & Hotdogs Use Case

<table>
<thead>
<tr>
<th>Orders (O for short)</th>
<th>Dish (D for short)</th>
<th>Items (I for short)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
<td>dish</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Steve</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consider the natural join of the above relations:

\[
O(\text{customer, day}, \text{dish}), D(\text{dish, item}), I(\text{item, price})
\]

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
### Burgers & Hotdogs in Relational Algebra

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
</tbody>
</table>

An algebraic encoding uses product (×), union (∪), and values:

```
Elise × Monday × burger × patty × 6 ∪
Elise × Monday × burger × onion × 2 ∪
Elise × Monday × burger × bun × 2 ∪
Elise × Friday × burger × patty × 6 ∪
Elise × Friday × burger × onion × 2 ∪
Elise × Friday × burger × bun × 2 ∪...
```
There are several algebraically equivalent factorised joins defined by distributivity of product over union and their commutativity.
Observation:

- price is under item, which is under dish, but only depends on item,
- so the same price appears under an item regardless of the dish.

Idea: Cache price for a specific item and avoid repetition!
COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto \ast$. 
Factorised Aggregate Computation

COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$. 
SUM(price) GROUP BY dish computed in one pass over the factorisation:

- All values except for dish & price $\mapsto 1$,
- $\cup \mapsto +, \times \mapsto \ast$. 
SUM(price) GROUP BY dish computed in one pass over the factorisation:

- All values except for dish & price $\mapsto 1$,
- $\cup \mapsto +, \times \mapsto *$. 
Sum-Product Ring Abstraction

⇓

Sharing Aggregate Computation
Ring for computing \( \text{SUM}(1) \), \( \text{SUM}(\text{price}) \), \( \text{SUM}(\text{price}) \) \ GROUP BY \text{dish}: 

- Elements = triples, one per aggregate 
- Sum (+) and product (*) now defined over triples 
  They enable shared computation across the aggregates
Shared Computation of Several Aggregates (2/2)

Ring for computing \( \text{SUM}(1), \text{SUM}(\text{price}), \text{SUM}(\text{price}) \) GROUP BY dish:

- Elements = triples, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates
Ring generalisation for the entire covariance matrix

Ring \((\mathcal{R}, +, *, 0, 1)\) over triples of aggregates \((c, s, Q) \in \mathcal{R}:\)

\[
\begin{pmatrix}
\sum(1), & \sum(x_i), & \sum(x_i \cdot x_j)
\end{pmatrix}
\]

\[
(c_1, s_1, Q_1) + (c_2, s_2, Q_2) = (c_1 + c_2, s_1 + s_2, Q_1 + Q_2)
\]

\[
(c_1, s_1, Q_1) \cdot (c_2, s_2, Q_2) = (c_1 \cdot c_2, c_2 \cdot s_1 + c_1 \cdot s_2,
\]

\[
\begin{align*}
c_2 \cdot Q_1 + c_1 \cdot Q_2 + s_1 s_2^T + s_2 s_1^T
\end{align*}
\]

\[
0 = (0, 0_{n \times 1}, 0_{n \times n})
\]

\[
1 = (1, 0_{n \times 1}, 0_{n \times n})
\]

- \(\sum(1)\) reused for all \(\sum(x_i)\) and \(\sum(x_i \cdot x_j)\)
- \(\sum(x_i)\) reused for all \(\sum(x_i \cdot x_j)\)
Idea 3: Lower the Constant Factors

- $12x$
- $3x$
- $2x$
1. **Specialisation** for workload and data
   - Generate code specific to the query batch and dataset
   - Improve cache locality for hot data path

2. **Sharing low-level data access**
   - Aggregates decomposed into views over join tree
   - Share data access across views with different output schemas

3. **Parallelisation**: multi-core (SIMD & distribution to come)
   - Task and domain parallelism

[DEEM’18, SIGMOD’19, CGO’20]
IFAQ: Iterative Functional Aggregate Queries

One DSL to Express both DB and ML Workloads! [CGO’20]

- Building blocks: Functional Aggregate Queries [PODS’16]
  - Formalism that expresses computation in databases, linear algebra, AI, logic
  - Relations are dictionaries
  - Sum-product computation over dictionaries
  - Conditionals using Kronecker delta

- Iteration constructs for
  - Stateful computation over collection elements
  - Constructing nested dictionaries
Transformation Steps for IFAQ Expressions

IFAQ Expression → Loop Scheduling → Factorisation → Static Memoisation → Code Motion

High-Level Optimisations

IFAQ Expression

Loop Scheduling

Factorisation

Static Memoisation

Code Motion

Loop Unrolling → Static Field Access → Aggregate Extraction → Aggregate Pushdown → Aggregate Fusion

Schema Specialisation

Loop Unrolling

Static Field Access

Aggregate Extraction

Aggregate Pushdown

Aggregate Fusion

Aggregate Optimisations

Trie Conversion → Code Motion → Factorisation → Data Layout → C++ Code

Trie Conversion

Code Motion

Factorisation

Data Layout

C++ Code

Trie Conversion

High-Level Optimisations

ICAQ Expression
Relative Speedup of Code Optimisations

Added optimisations for covariance matrix computation:

specialisation → + sharing → + parallelisation

AWS d2.xlarge (4 vCPUs, 32GB)
Conclusion
1. Turn the learning problem into a database problem

2. Exploit the problem structure to lower the complexity

3. Specialise and optimise the code to lower the constant factors

Q.E.D.
Relational Data Borg’s Call to Arms

We need more sustained work on theory and systems for

Structure-aware Approaches to Data Analytics