From Joins to Aggregates and Optimization Problems:

One idea to rule them all and at their core *factorize* them!

Dan Olteanu (Oxford)

http://www.cs.ox.ac.uk/projects/FDB/

RiSE & LogiCS Summer School, July 4, 2017
Acknowledgements

Most work reported in this tutorial has been done in the context of the FDB project and/or LogicBlox analytics engine by

- the FDB members Zavodny, Schleich, Kara, Ciucanu, and myself
- collaborators Abo Khamis and Ngo (LogicBlox), Nguyen (U. Michigan)

Some of these slides are derived from presentations by

- Kara (on covers and various graphics)
- Ciucanu (on join-at-a-time processing)
- Ngo (on FAQ)
- Abo Khamis (in-db analytics diagrams)

Lastly, Kara and Schleich proofread the slides.

I would like to thank them for their support and sharing their work!
Goal of This Tutorial

Introduction to a systematic approach to in-database computation

- **Joins**
  - Worst-case optimal join algorithms
  - Factorized representations of join results

- **Aggregates**
  - Generalization of join algorithms to aggregates over joins
  - Functional aggregate queries with applications in e.g., DB, logic, probabilistic graphical models, matrix chain computation

- **Optimization Problems**
  - In-database machine learning

- Highlight some open problems we are currently working on
Outline

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Part 4. Open Problems
Join Queries

Basic building blocks in query languages and studied extensively.

However, worst-case optimal join algorithms were only proposed recently. 

[NPRR12, NRR13, V14, OZ15, ANS17]

Likewise for systematic investigation of redundancy in the computation and representation of join results. [OZ12, OZ15, KO17]

This tutorial highlights recent work on worst-case optimal join algorithms under different data representations.
Plan for Tutorial Part 1 on Joins

- Introduction of join queries via examples
- Size bounds for results of join queries
  - Standard (exhaustive) listing representation
  - Factorized (succinct) representations
- Worst-case optimal join algorithms
  - LeapFrog TrieJoin used by LogicBlox for listing representation
  - FDB for factorized representations
Introduction to Join Queries
## Join Example: Itemized Customer Orders

<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>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
</tbody>
</table>

Consider the natural join of the above relations:

<table>
<thead>
<tr>
<th>O(customer, day, dish), D(dish, item), I(item, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Join Example: Listing the Triangles in the Database

<table>
<thead>
<tr>
<th></th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
<th>( R_1(A, B), R_2(A, C), R_3(B, C) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( A )</td>
<td>( A )</td>
<td>( B )</td>
<td>( A )</td>
</tr>
<tr>
<td></td>
<td>( B )</td>
<td>( C )</td>
<td>( C )</td>
<td>( B )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_0 )</td>
<td>( a_0 )</td>
<td>( c_0 )</td>
<td>( a_0 ) ( b_0 ) ( c_0 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( \ldots )</td>
<td>( a_0 )</td>
<td>( \ldots )</td>
<td>( a_0 ) ( b_0 ) ( c_0 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_m )</td>
<td>( a_0 )</td>
<td>( c_m )</td>
<td>( a_0 ) ( b_0 ) ( c_m )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_0 )</td>
<td>( a_1 )</td>
<td>( c_0 )</td>
<td>( a_0 ) ( b_1 ) ( c_0 )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( a_0 ) ( b_0 ) ( c_0 )</td>
</tr>
<tr>
<td>( a_m )</td>
<td>( b_0 )</td>
<td>( a_m )</td>
<td>( c_0 )</td>
<td>( a_0 ) ( b_m ) ( c_0 )</td>
</tr>
<tr>
<td>( b_m )</td>
<td>( c_0 )</td>
<td>( b_m )</td>
<td>( c_0 )</td>
<td>( a_1 ) ( b_0 ) ( c_0 )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td></td>
<td>( \ldots )</td>
<td></td>
<td>( \ldots ) ( b_0 ) ( c_0 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td></td>
<td></td>
<td></td>
<td>( \ldots ) ( b_0 ) ( c_0 )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
<td>( a_1 ) ( b_0 ) ( c_0 )</td>
</tr>
</tbody>
</table>
Join Hypergraphs

We associate with every join query $Q$ a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$

- $\mathcal{V}$ is the set of its variables, $\mathcal{E}$ consists of the sets of variables of relation symbols

Examples:
- Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

- Order query: $O(\text{cust}, \text{day}, \text{dish}), D(\text{dish}, \text{item}), I(\text{item}, \text{price})$
**Definition:** A (hyper)tree decomposition $\mathcal{T}$ of the hypergraph $(\mathcal{V}, \mathcal{E})$ of a query $Q$ is a pair $(\mathcal{T}, \chi)$, where
- $\mathcal{T}$ is a tree
- $\chi$ is a function mapping each node in $\mathcal{T}$ to a subset of $\mathcal{V}$ or bag.

Properties of a tree decomposition $\mathcal{T}$:
- **Coverage:** $\forall e \in \mathcal{E}$, there must be a node $t \in \mathcal{T}$ such that $e \subseteq \chi(t)$.
- **Connectivity:** $\forall v \in \mathcal{V}$, $\{ t \mid t \in \mathcal{T}, v \in \chi(t) \}$ forms a connected subtree.

The hypergraph of the query $R_1(A, B), R_2(B, C), R_3(C, D)$

A tree decomposition
**Hypertree Decompositions**

**Definition:** A (hyper)tree decomposition \( T \) of the hypergraph \((\mathcal{V}, \mathcal{E})\) of a query \( Q \) is a pair \((T, \chi)\), where
\[
\begin{align*}
&\quad T \text{ is a tree} \\
&\quad \chi \text{ is a function mapping each node in } T \text{ to a subset of } \mathcal{V} \text{ called bag.}
\end{align*}
\]

[GLS99]

**Properties of a tree decomposition \( T \):**
\[
\begin{align*}
&\quad \text{Coverage: } \forall e \in \mathcal{E}, \text{ there must be a node } t \in T \text{ such that } e \subseteq \chi(t). \\
&\quad \text{Connectivity: } \forall v \in \mathcal{V}, \{ t \mid t \in T, v \in \chi(t) \} \text{ forms a connected subtree.}
\end{align*}
\]

The hypergraph of the triangle query \( R_1(A, B), R_2(A, C), R_3(B, C) \)

A tree decomposition

![Diagram of a hypergraph and tree decomposition](image)
Variable Orders

**Definition:** A variable order $\Delta$ for a query $Q$ is a pair $(F, key)$, where
- $F$ is a rooted forest with one node per variable in $Q$
- $key$ is a function mapping each variable $A$ to a subset of its ancestor variables in $F$.

Properties of a variable order $\Delta$ for $Q$:
- For each relation symbol, its variables lie along the same root-to-leaf path in $F$. For any such variables $A$ and $B$, $A \in key(B)$ if $A$ is an ancestor of $B$.
- For every child $B$ of $A$, $key(B) \subseteq key(A) \cup \{A\}$.

Possible variable orders for the path query $R_1(A, B), R_2(B, C), R_3(C, D)$:

```
A      key(A) = ∅
   |  
B  key(B) = {A}
   |  
C  key(C) = {B}
   |  
D  key(D) = {C}
```

```
B   key(B) = ∅
   |  
A    key(A) = {B}
   |  
C    key(C) = {B}
   |  
D   key(D) = {C}
```
**Variable Orders**

**Definition:** A variable order $\Delta$ for a query $Q$ is a pair $(F, \text{key})$, where

- $F$ is a rooted forest with one node per variable in $Q$
- $\text{key}$ is a function mapping each variable $A$ to a subset of its ancestor variables in $F$.

Properties of a variable order $\Delta$ for $Q$:

- For each relation symbol, its variables lie along the same root-to-leaf path in $F$. For any such variables $A$ and $B$, $A \in \text{key}(B)$ if $A$ is an ancestor of $B$.
- For every child $B$ of $A$, $\text{key}(B) \subseteq \text{key}(A) \cup \{A\}$.

Possible variable orders for the triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$:

<table>
<thead>
<tr>
<th>A</th>
<th>$\text{key}(A) = \emptyset$</th>
<th>B</th>
<th>$\text{key}(B) = \emptyset$</th>
<th>C</th>
<th>$\text{key}(C) = \emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>$\text{key}(B) = {A}$</td>
<td>A</td>
<td>$\text{key}(A) = {B}$</td>
<td>B</td>
<td>$\text{key}(B) = {C}$</td>
</tr>
<tr>
<td>C</td>
<td>$\text{key}(C) = {A, B}$</td>
<td>C</td>
<td>$\text{key}(C) = {A, B}$</td>
<td>A</td>
<td>$\text{key}(A) = {B, C}$</td>
</tr>
</tbody>
</table>
Tree Decompositions $\iff$ Variable Orders

From variable order $\Delta$ to tree decomposition $T$: [OZ15]

- For each node $A$ in $\Delta$, create a bag $key(A) \cup \{A\}$.
- The bag for $A$ is connected to the bags for its children and parent.
- Optionally, remove redundant bags

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

```
  A  key(A) = {}  ⇒  A  ⇒  A, B, C
  |   
  B  key(B) = \{A\}  ⇒  A, B  ⇒  A, B, C
  |   
  C  key(C) = \{A, B\}
```

```
Tree Decompositions ⇔ Variable Orders

From variable order $\Delta$ to tree decomposition $T$: [OZ15]

- For each node $A$ in $\Delta$, create a bag $key(A) \cup \{A\}$.
- The bag for $A$ is connected to the bags for its children and parent.
- Optionally, remove redundant bags

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

```
A  key(A) = ∅
  /   /
B  key(B) = {A}  ⇒  A, B  ⇒  A, B
  |
C  key(C) = {B}
  |
D  key(D) = {C}
```

16 / 142
Tree Decompositions ⇔ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$:

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update $key$ for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$
Tree Decompositions $\iff$ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$: \cite{OZ15}

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update $key$ for each variable at each step.

Example: Triangle query $R_1(A, B)$, $R_2(A, C)$, $R_3(B, C)$

\[
\begin{align*}
A & \quad key(A) = \emptyset \\
\text{Step 1:} & \\
\text{A is removed from } \mathcal{T} & \\
\text{and inserted into } \Delta
\end{align*}
\]
Tree Decompositions ⇔ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$:

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update $key$ for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

Step 2:

$B$ is removed from $\mathcal{T}$ and inserted into $\Delta$

$A, B, C$ $\Rightarrow$ $B$

$A$ $key(A) = \emptyset$

$B$ $key(B) = \{A\}$
Tree Decompositions ⇔ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$: \[\text{[OZ15]}\]

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update key for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

Step 3: $C$ is removed from $\mathcal{T}$ and inserted into $\Delta$

\[
\begin{align*}
A & \quad \text{key}(A) = \emptyset \\
B & \quad \text{key}(B) = \{A\} \\
C & \quad \text{key}(C) = \{A, B\}
\end{align*}
\]
Tree Decompositions $\iff$ Variable Orders

From tree decomposition $T$ to variable order $\Delta$: [OZ15]

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $T$
- Recurse with $T$ where $A$ is removed from all bags in $T$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update $key$ for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

```
A, B
  / \  \
B, C
  /   /
C, D
```
Tree Decompositions ⇔ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$: [OZ15]

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update $key$ for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

Step 1:
$A$ is removed from $\mathcal{T}$
and inserted into $\Delta$

$\Rightarrow$

$A\quad key(A) = \emptyset$

$A, B$

$B, C$

$C, D$
Tree Decompositions ⇔ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$:  

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update key for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

```
Step 2:
B is removed from $\mathcal{T}$
and inserted into $\Delta$
```

```
A, B

B, C

C, D

A

key(A) = \emptyset

B

key(B) = \{A\}
```
Tree Decompositions $\Leftrightarrow$ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$: [OZ15]

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$
- Update $key$ for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

Step 3:

$C$ is removed from $\mathcal{T}$ and inserted into $\Delta$

$A \quad key(A) = \emptyset$

$B \quad key(B) = \{A\}$

$C \quad key(C) = \{B\}$
Tree Decompositions $\iff$ Variable Orders

From tree decomposition $\mathcal{T}$ to variable order $\Delta$: [OZ15]

- Create a node $A$ in $\Delta$ for a variable $A$ in the top bag in $\mathcal{T}$.
- Recurse with $\mathcal{T}$ where $A$ is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of $A$ in $\Delta$.
- Update $key$ for each variable at each step.

Example: Path query $R_1(A, B)$, $R_2(B, C)$, $R_3(C, D)$

Step 4:

$D$ is removed from $\mathcal{T}$ and inserted into $\Delta$.

$A \quad key(A) = \emptyset$

$B \quad key(B) = \{A\}$

$C \quad key(C) = \{B\}$

$D \quad key(D) = \{C\}$
Size Bounds for Listing Representation of Join Results
How Can We Bound the Size of the Join Result?

(Assumption in this tutorial: all relations have size $N$.)

Example: the path query $R_1(A, B)$, $R_2(B, C)$, $R_3(C, D)$

- The result is included in the result of $R_1(A, B)$, $R_3(C, D)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$
  - All variables are "covered" by the relations $R_1$ and $R_3$

- There are databases for which the result size is at least $N^2$
  - Let $R_1 = [N] \times \{1\}$, $R_2 = \{1\} \times [N]$, $R_3 = [N] \times \{1\}$. 
How Can We Bound the Size of the Join Result?

(Assumption in this tutorial: all relations have size $N$.)

Example: the path query $R_1(A, B), R_2(B, C), R_3(C, D)$

- The result is included in the result of $R_1(A, B), R_3(C, D)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$.
  - All variables are ”covered” by the relations $R_1$ and $R_3$.

- There are databases for which the result size is at least $N^2$
  - Let $R_1 = [N] \times \{1\}, R_2 = \{1\} \times [N], R_3 = [N] \times \{1\}$.

Example: the triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

- The result is included in the result of $R_1(A, B), R_3(B, C)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$.
  - All variables are ”covered” by the relations $R_1$ and $R_3$.

- There are databases for which the result size is at least $N$
  - Let $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supset \{(1, 1)\}$. 
Edge Covers and Independent Sets

We can generalize the previous examples as follows:

For the size upper bound:
- Cover all variables by \( k \) relations \( \Rightarrow \) size \( \leq N^k \).
- This is an edge cover of the query hypergraph!

For the size lower bound:
- Set of \( m \) independent variables \( \Rightarrow \) construct database such that size \( \geq N^m \).
- This is an independent set of the query hypergraph!

\[
\max_m = |\text{IndependentSet}(Q)| \leq |\text{EdgeCover}(Q)| = \min_k
\]

\[\text{max}_m \text{ and } \text{min}_k \text{ do not necessarily meet!}\]

Can we further refine this analysis?
The Fractional Edge Cover Number $\rho^*(Q)$

The two bounds meet if we take their fractional versions $[AGM08]$

- *Fractional* edge cover of $Q$ with weight $k \Rightarrow$ size $\leq N^k$.
- *Fractional* independent set with weight $m \Rightarrow$ construct database such that size $\geq N^m$.

By duality of linear programming:

$$\max_m = |\text{FractionalIndependentSet}(Q)| = |\text{FractionalEdgeCover}(Q)| = \min_k$$

- This is the fractional edge cover number $\rho^*(Q)$!
The Fractional Edge Cover Number $\rho^*(Q)$

For a join query $Q(A_1 \cup \cdots \cup A_n) = R_1(A_1), \ldots, R_n(A_n)$, $\rho^*(Q)$ is the cost of an optimal solution to the linear program:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i \in [n]} x_{R_i} \\
\text{subject to} & \quad \sum_{i: R_i \text{ has variable } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} A_j, \\
& \quad x_{R_i} \geq 0 \quad \forall i \in [n].
\end{align*}
\]

- $x_{R_i}$ is the weight of relation $R_i$.
- Each variable has to be covered by relations with sum of weights $\geq 1$.
- In the integer program for the edge cover, $x_{R_i} \in \{0, 1\}$.
Consider the following join query $Q$:

$$R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

- Relations $R, S, U$ cover all variables.
  \[ \text{FractionalEdgeCover}(Q) \leq 3 \]
- Each of the nodes $C, D,$ and $F$ must be covered by separate relations.
  \[ \text{FractionalIndependentSet}(Q) \geq 3 \]

\[ \Rightarrow \rho^*(Q) = 3 \]

\[ \Rightarrow \text{Size} \leq N^3 \text{ and for some inputs is } \Theta(N^3). \]
Example of Fractional Edge Cover Computation (2)

Consider the triangle query \( Q: R_1(A, B), R_2(A, C), R_3(B, C) \).

\[
\begin{align*}
\text{minimize} & \quad x_{R_1} + x_{R_2} + x_{R_3} \\
\text{subject to} & \\
A: & \quad x_{R_1} + x_{R_2} \geq 1 \\
B: & \quad x_{R_1} + x_{R_3} \geq 1 \\
C: & \quad x_{R_2} + x_{R_3} \geq 1 \\
& \quad x_{R_1} \geq 0 \quad x_{R_2} \geq 0 \quad x_{R_3} \geq 0
\end{align*}
\]

Our previous size upper bound was \( N^2 \):

- This is obtained by setting any two of \( x_{R_1}, x_{R_2}, x_{R_3} \) to 1.

What is the fractional edge cover number for the triangle query?
Consider the triangle query $Q$: $R_1(A, B), R_2(A, C), R_3(B, C)$.

$$\begin{align*}
\text{minimize} & \quad x_{R_1} + x_{R_2} + x_{R_3} \\
\text{subject to} & \quad A : \quad x_{R_1} + x_{R_2} \geq 1 \\
& \quad B : \quad x_{R_1} + x_{R_3} \geq 1 \\
& \quad C : \quad x_{R_2} + x_{R_3} \geq 1 \\
& \quad x_{R_1} \geq 0 \quad x_{R_2} \geq 0 \quad x_{R_3} \geq 0
\end{align*}$$

Our previous size upper bound was $N^2$:

- This is obtained by setting any two of $x_{R_1}, x_{R_2}, x_{R_3}$ to 1.

What is the fractional edge cover number for the triangle query?

We can do better: $x_{R_1} = x_{R_2} = x_{R_3} = 1/2$. Then, $\rho^* = 3/2$.

Lower bound reaches $N^{3/2}$ for $R_1 = R_2 = R_3 = [\sqrt{N}] \times [\sqrt{N}]$. 
Example of Fractional Edge Cover Computation (3)

Consider the (4-cycle) join: \( R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1) \).

The linear program for its fractional edge cover number:

\[
\begin{align*}
\text{minimize} & \quad x_R + x_S + x_T + x_W \\
\text{subject to} & \quad x_R + x_W \geq 1 \\
& \quad x_R \geq 0 \quad x_S \geq 0 \quad x_T \geq 0 \quad x_W \geq 0
\end{align*}
\]

Possible solution: \( x_R = x_T = 1 \). Another solution: \( x_S = x_W = 1 \). Then, \( \rho^* = 2 \).

Lower bound reaches \( N^2 \) for \( R = T = [N] \times \{1\} \) and \( S = W = \{1\} \times [N] \).
Size Bounds for Factorized Representations of Join Results
Recall the Itemized Customer Orders Example

Orders (O for short)

<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>

Dish (D for short)

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

Items (I for short)

<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>

Consider the natural join of the above relations:

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

<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>
</tbody>
</table>
Factor Out Common Data Blocks

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

<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>
</tbody>
</table>

... ... ... ... ...

The listing representation of the above query result is:

\[
\langle Elise \rangle \times \langle Monday \rangle \times \langle burger \rangle \times \langle patty \rangle \times \langle 6 \rangle \cup \\
\langle Elise \rangle \times \langle Monday \rangle \times \langle burger \rangle \times \langle onion \rangle \times \langle 2 \rangle \cup \\
\langle Elise \rangle \times \langle Monday \rangle \times \langle burger \rangle \times \langle bun \rangle \times \langle 2 \rangle \cup \\
\langle Elise \rangle \times \langle Friday \rangle \times \langle burger \rangle \times \langle patty \rangle \times \langle 6 \rangle \cup \\
\langle Elise \rangle \times \langle Friday \rangle \times \langle burger \rangle \times \langle onion \rangle \times \langle 2 \rangle \cup \\
\langle Elise \rangle \times \langle Friday \rangle \times \langle burger \rangle \times \langle bun \rangle \times \langle 2 \rangle \cup \ldots
\]

It uses relational product (\(\times\)), union (\(\cup\)), and data (singleton relations).

- The attribute names are not shown to avoid clutter.
This is How A Factorized Join Looks Like!

There are several \textit{algebraically equivalent} factorized representations defined:

\begin{itemize}
  \item by distributivity of product over union and their commutativity;
  \item as groundings of variable orders.
\end{itemize}
.. Now with Further Compression using Caching

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!
.. and Further Compressed using Caching
Which factorization should we choose?

The *size* of a factorization is the number of its values.

Example:

\[ F_1 = \left( \langle 1 \rangle \cup \cdots \cup \langle n \rangle \right) \times \left( \langle 1 \rangle \cup \cdots \cup \langle m \rangle \right) \]
\[ F_2 = \langle 1 \rangle \times \langle 1 \rangle \cup \cdots \cup \langle 1 \rangle \times \langle m \rangle \]
\[ \quad \cup \cdots \cup \]
\[ \langle n \rangle \times \langle 1 \rangle \cup \cdots \cup \langle n \rangle \times \langle m \rangle \].

- \( F_1 \) is factorized, \( F_2 \) is a listing representation
- \( F_1 \equiv F_2 \)
- **BUT** \( |F_1| = m + n \ll |F_2| = m \times n \).

How much space does factorization save over the listing representation?
Size Bounds for Join Results

Given a join query $Q$, for any database of size $N$, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$.  

[AGM08]
Size Bounds for Join Results

Given a join query \(Q\), for any database of size \(N\), the join result admits

- a listing representation of size \(O(N^{\rho^*(Q)})\). \[AGM08\]

- a factorization \textit{without caching} of size \(O(N^{s(Q)})\). \[OZ12\]
Size Bounds for Join Results

Given a join query $Q$, for any database of size $N$, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. [AGM08]
- a factorization without caching of size $O(N^{s(Q)})$. [OZ12]
- a factorization with caching of size $O(N^{fhtw(Q)})$. [OZ15]
Size Bounds for Join Results

Given a join query $Q$, for any database of size $N$, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. \[\text{[AGM08]}\]

- a factorization without caching of size $O(N^{s(Q)})$. \[\text{[OZ12]}\]

- a factorization with caching of size $O(N^{fhtw(Q)})$. \[\text{[OZ15]}\]

\[
1 \leq fhtw(Q) \leq s(Q) \leq \rho^*(Q) \leq |Q|
\]

- up to $\log |Q|$ 
- up to $|Q|$

- $|Q|$ is the number of relations in $Q$
- $\rho^*(Q)$ is the fractional edge cover number of $Q$
- $s(Q)$ is the factorization width of $Q$
- $fhtw(Q)$ is the fractional hypertree width of $Q$ \[\text{[M10]}\]
Size Bounds for Join Results

Given a join query $Q$, for any database of size $N$, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. \[AGM08\]

- a factorization without caching of size $O(N^{s(Q)})$. \[OZ12\]

- a factorization with caching of size $O(N^{fhtw(Q)})$. \[OZ15\]

These size bounds are asymptotically tight!

- **Best possible size bounds** for factorized representations over variable orders of $Q$ and for listing representation, but not database optimal!

There exists arbitrarily large databases for which

- the listing representation has size $\Omega(N^{\rho^*(Q)})$
- the factorization with (without) caching over any variable order of $Q$ has size $\Omega(N^{s(Q)})$ ($\Omega(N^{fhtw(Q)})$).
Example: The Factorization Width $s$

The structure of the factorization over the above variable order $\Delta$:

$$\bigcup_{a \in A} \left( \langle a \rangle \times \bigcup_{b \in B} \left( \langle b \rangle \times \left( \bigcup_{c \in C} \langle c \rangle \right) \times \left( \bigcup_{d \in D} \langle d \rangle \right) \right) \times \bigcup_{e \in E} \left( \langle e \rangle \times \left( \bigcup_{f \in F} \langle f \rangle \right) \right) \right)$$

The number of values for a variable is dictated by the number of valid tuples of values for its ancestors in $\Delta$:

- One value $\langle f \rangle$ for each tuple $(a, e, f)$ in the join result.

Size of factorization = sum of sizes of results of subqueries along paths.
Example: The Factorization Width $s$

- The factorization width for $\Delta$ is the largest $\rho^*$ over subqueries defined by root-to-leaf paths in $\Delta$
- $s(Q)$ is the minimum factorization width over all variable orders of $Q$

In our example:

- Path $A\rightarrow E\rightarrow F$ has fractional edge cover number 2.
  - $\Rightarrow$ The number of $F$-values is $\leq N^2$, but can be $\sim N^2$.
- All other root-to-leaf paths have fractional edge cover number 1.
  - $\Rightarrow$ The number of other values is $\leq N$.

$s(Q) = 2$ \quad $\Rightarrow$ Factorization size is $\Theta(N^2)$

Recall that $\rho^*(Q) = 3$ \quad $\Rightarrow$ Listing representation size is $\Theta(N^3)$
Example: The Fractional Hypertree Width \( fhtw \)

Idea: Avoid repeating identical expressions, store them once and use pointers.

\[
\begin{align*}
\text{key}(A) &= \emptyset \\
\text{key}(B) &= \{A\} \\
\text{key}(C) &= \{A, B\} \\
\text{key}(D) &= \{A, B\} \\
\text{key}(E) &= \{A\} \\
\text{key}(F) &= \{E\}
\end{align*}
\]

\[
\bigcup_{a \in A} [\langle a \rangle \times \cdots \times \bigcup_{e \in E} (\langle e \rangle \times (\bigcup_{f \in F} \langle f \rangle))]
\]

Observation:

- Variable \( F \) only depends on \( E \) and not on \( A \): \( \text{key}(F) = \{E\} \)
- A value \( \langle e \rangle \) maps to the same union \( \bigcup_{(e,f) \in U} \langle f \rangle \) regardless of its pairings with \( A \)-values.

\[\Rightarrow \text{Define } U_e = \bigcup_{(e,f) \in U} \langle f \rangle \text{ for each value } \langle e \rangle \text{ and use } U_e \text{ instead of the union } \bigcup_{(e,f) \in U} \langle f \rangle.\]
Example: The Fractional Hypertree Width $fhtw$

Idea: Avoid repeating identical expressions, store them once and use pointers.

A factorization with definitions would be:

$$
\bigcup_{a \in A} \left[ a \right] \times \cdots \times \bigcup_{e \in E} \left( e \times U_e \right); \quad \left\{ U_e = \bigcup_{(e, f) \in U} \langle f \rangle \right\}
$$

- $fhtw$ for $\Delta$ is the largest $\rho^*(Q')$ over subqueries $Q'$ defined by the variables $key(X) \cup \{X\}$ for each variable $X$ in $\Delta$
- $fhtw(Q)$ is the minimum $fhtw$ over all variable orders of $Q$

In our example: $fhtw(Q) = 1 < s(Q) = 2 < \rho^*(Q) = 3$. 
Relational Counterpart of Factorized Representation
Covers: Relational Counterparts of Factorizations

- Factorized representations are not relational :(  
  - This makes it difficult to integrate them into relational data systems

- Covers of Query Results [KO17]
  - Relations that are lossless representations of query results, yet are as succinct as factorized representations
  - For a join query $Q$ and any database of size $N$, a cover has size $O(N^{fhtw(Q)})$ and can be computed in time $O(N^{fhtw(Q)} \log N)$

- How to get a cover?
  - Construct a hypertree decomposition of the query
  - Project query result onto the bags of the hypertree decomposition
  - Construct on these projections the hypergraph of the query result
  - Take a minimal edge cover of this hypergraph
Recall the Itemized Customer Orders Example

<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><strong>customer</strong></td>
<td><strong>dish</strong></td>
<td><strong>item</strong></td>
</tr>
<tr>
<td>Elise</td>
<td>burger</td>
<td>patty</td>
</tr>
<tr>
<td>Elise</td>
<td>burger</td>
<td>onion</td>
</tr>
<tr>
<td>Steve</td>
<td>hotdog</td>
<td>bun</td>
</tr>
<tr>
<td>Joe</td>
<td>hotdog</td>
<td>bun</td>
</tr>
<tr>
<td></td>
<td>hotdog</td>
<td>sausage</td>
</tr>
</tbody>
</table>

| customer, day, dish | dish, item       | item, price       |

<table>
<thead>
<tr>
<th>customer, day, dish</th>
<th>dish, item</th>
<th>item, price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday, burger</td>
<td>patty, 6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday, burger</td>
<td>onion, 2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday, burger</td>
<td>bun, 2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday, burger</td>
<td>patty, 6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday, burger</td>
<td>onion, 2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday, burger</td>
<td>bun, 2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday, burger</td>
<td>patty, 6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday, burger</td>
<td>onion, 2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday, burger</td>
<td>bun, 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

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

O(customer, day, dish), D(dish, item), I(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>
</tbody>
</table>
The Hypergraph of the Query Result

customer, day, dish

- dish, item
- item, price

O(customer, day, dish), D(dish, item), I(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>
</tbody>
</table>

...
The Hypergraph of the Query Result

O(customer, day, dish), D(dish, item), I(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>
</tbody>
</table>

...
The Hypergraph of the Query Result

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

O(customer, day, dish), D(dish, item), I(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>
</tbody>
</table>
The Hypergraph of the Query Result

O(customer, day, dish), D(dish, item), I(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>
</tbody>
</table>

customer, day, dish

dish, item

item, price
The Hypergraph of the Query Result

O(customer, day, dish), D(dish, item), I(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>
</tbody>
</table>

...
The Hypergraph of the Query Result

<table>
<thead>
<tr>
<th>customer, day, dish</th>
<th>O(customer, day, dish), D(dish, item), I(item, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

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

<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>
</tbody>
</table>

customer, day, dish

- dish, item
- item, price
The Hypergraph of the Query Result

| O(customer, day, dish), D(dish, item), I(item, price) |
|-----------------------------------|----------|---------|---------|----------|
| customer | day    | dish   | item   | price   |
| 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       |
| ...      | ...    | ...    | ...    | ...     |
A Minimal Edge Cover of the Hypergraph

Elise Monday burger

Elise Friday burger

| O(customer, day, dish), D(dish, item), I(item, price) |
|---------------|-----------|------------|--------|-------|
| customer      | day       | dish       | item   | price |
| 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     |
| ...            | ...       | ...        | ...    | ...   |
A Cover of (a part of) the Query Result

<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>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>

...
Compression by Factorization in Practice
Compression Contest: Factorized vs. Zipped Relations

Result of query Orders \Join Dish \Join Items

- **Tabular** = listing representation in CSV text format
- **Gzip** (compression level 6) outputs binary format
- **Factorized** representation in text format (each digit takes one character)

Observations:

- **Gzip** does not exploit distant repetitions!
- **Factorizations** can be arbitrarily more succinct than gzipped relations.
- **Gzipping factorizations** improves the compression by 3x.
Retailer dataset used for LogicBlox analytics

- Relations: Inventory (84M), Sales (1.5M), Clearance (368K), Promotions (183K), Census (1K), Location (1K).

- Compression factors (caching not used):
  - 26.61x for natural join of Inventory, Census, Location.
  - 159.59x for natural join of Inventory, Sales, Clearance, Promotions
Factorization Gains in Practice (2/3)

LastFM public dataset


- Compression factors:
  - \(143.54x\) for joining two copies of UserArtists and UserFriends
  
  With caching: \(982.86x\)

  - \(253.34x\) when also joining on TaggedArtists

  - \(2.53x/\ 3.04x/\ 924.46x\) for triangle/4-clique/bowtie query on UserFriends

  - \(9213.51x/\ 552Kx/\ \geq 86Mx\) for versions of triangle/4-clique/bowtie queries with copies for UserArtists for each UserFriend copy
Factorization Gains in Practice (3/3)

Twitter public dataset

- Relation: Follower-Followee (1M)

- Compression factors:
  - 2.69x for triangle query
  - 3.48x for 4-clique query
  - 4918.73x for bowtie query
Worst-Case Optimal Join Algorithms
Given a join query $Q$, for any database of size $N$, the join result can be computed in time

- $O(N^{\rho^*(Q)} \log N)$ as listing representation \[\text{[NPRR12,V14]}\]
- $O(N^s(Q) \log N)$ as factorization \textit{without caching} \[\text{[OZ15]}\]
- $O(N^{fhtw(Q)} \log N)$ as factorization \textit{with caching} \[\text{[OZ15]}\]

These upper bounds essentially follow the succinctness gap. They are:

- worst-case optimal (modulo $\log N$) within the given representation model
- with respect to data complexity
  - additional quadratic factor in the number of variables and linear factor in the number of relations in $Q$
Example: Computing the Factorized Join Result with FDB

Our join: \( O(\text{customer, day, dish}) \), \( D(\text{dish, item}) \), \( I(\text{item, price}) \)

can be grounded to a factorized representation as follows:

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

This evaluation follows the variable order given below:

dish

   |
  day

   |
  item

   |
customer

   |
price
Example: Computing the Factorized Join Result with FDB

\[ \bigcup_{\text{O}(-, \cdot, \text{dish})} \langle \text{dish} \rangle \times \bigcup_{\text{O}(-, \text{day}, \text{dish})} \langle \text{day} \rangle \times \bigcup_{\text{O}(\text{customer}, \text{day}, \text{dish})} \langle \text{customer} \rangle \times \bigcup_{\text{I}(\text{item}, \text{price})} \langle \text{price} \rangle \times \bigcup_{\text{D}(\text{dish}, \cdot, \cdot)} \langle \text{dish} \rangle \]

- Relations are sorted following any topological order of the variable order.
- The intersection of relations O and D on dish takes time \( O(N_{\min} \log(N_{\max}/N_{\min})) \), where \( N_m = m(|\pi_{\text{dish}} O|, |\pi_{\text{dish}} D|) \).
- The remaining operations are lookups in the relations, where we first fix the dish value and then the day and item values.
LeapFrog TrieJoin

- Much acclaimed worst-case optimal join algorithm used by LogicBlox [V14]
- Computes a listing representation of the join result
  ⇒ It does not exploit factorization
- Glorified multi-way sort-merge join with an efficient list intersection

LeapFrog TrieJoin (LogicBlox) is a special case of our algorithm, where

- the input variable order $\Delta$ is a path, and
- for each variable $A$, $key(A)$ consists of all ancestors of $A$ in $\Delta$. 
Outline

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Part 4. Open Problems
Aggregates

Essential operators in any realistic query language and database-supported application.

Much DB research into optimization of aggregates is rather ad-hoc.

Aggregates over joins can express a host of problems in Computer Science, and it is only recently that this connection has been made explicit \[\text{ANR16}\]

This tutorial highlights recent theoretical work on aggregate computation with lowest computational complexity to date \[\text{BKOZ13,ANR16}\]

In Part 3, we will see how this development supports state-of-the-art machine learning inside the database \[\text{SOC16,ANNOS17}\]
Plan for Tutorial Part 2

- We will first show how aggregates can be computed over factorized joins. [BKOZ13]

- We will then show how to factorize the computation of aggregates using optimized relational queries.

- We will then present a beautiful generalization of aggregates over joins called Functional Aggregate Queries (FAQs). [ANR16]
  
  - Captures different semirings, e.g., sum-product, max-product, Boolean
  - Captures a large class of problems across Computer Science
  - FAQ computation is factorized and has the computational complexity of aggregates over factorized joins
Examples: Aggregates over Factorized Joins
Example 1: COUNT Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

- **COUNT(\(\ast\))**:  
  - values \(\mapsto\) 1,  
  - \(\cup\) \(\mapsto\) +,  
  - \(\times\) \(\mapsto\) \(*\).
Example 1: COUNT Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

- **COUNT(*)**:
  - values $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto \ast$. 
Example 2: SumProd Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

- **SUM(dish * price):**
  - Assume there is a function \( f \) that turns dish into reals.
  - All values except for dish & price \( \mapsto 1 \),
  - \( \cup \mapsto + \),
  - \( \times \mapsto \ast \).
Example 2: SumProd Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

- **SUM(dish * price):**
  - Assume there is a function $f$ that turns dish into reals.
  - All values except for dish & price $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto \ast$. 

$$20f(\langle \text{burger} \rangle) + 16f(\langle \text{hotdog} \rangle)$$
Example: Factorized Aggregate Computation

The 4-path query $Q_4$ on a graph with the edge relation $E$ ($E_i$’s are copies of $E$):

$V_1(A), E_1(A, B), E_2(B, C), E_3(C, D), E_4(D, E), V_2(E)$
Example: Factorized Aggregate Computation

The 4-path query \( Q_4 \) on a graph with the edge relation \( E \) (\( E_i \)'s are copies of \( E \)):

\[
V_1(A), E_1(A, B), E_2(B, C), E_3(C, D), E_4(D, E), V_2(E)
\]

Recall sizes for factorized results of path queries

- \( \rho^*(Q_4) = 3 \Rightarrow \text{listing representation has size } O(|E|^3) \).
- \( s(Q_4) = 2 \Rightarrow \text{factorization without caching has size } O(|E|^2) \).
- \( fhtw(Q_4) = 1 \Rightarrow \text{factorization with caching has size } O(|E|) \).
- For the \( n \)-path query \( Q_n \), \( s(Q_n) = \log_2 n \) and \( fhtw(Q_n) = 1 \).
Example: Factorized Aggregate Computation

We would like to compute \( \text{COUNT}(Q_4) \):

- in (pseudo)linear time
- using optimized queries that are derived from the variable order of \( Q_4 \)
- without materializing the factorized result of the path query

Convention:

- View the relations as functions mapping tuples to numbers.
- The functions for input relations map their tuples to 1.
Example: Factorized Aggregate Computation

We next show how to compute \( \text{COUNT}(Q_4) \).
Example: Factorized Aggregate Computation

We next show how to compute $\text{COUNT}(Q_4)$.

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a)
\]
Example: Factorized Aggregate Computation

We next show how to compute \( \text{COUNT}(Q_4) \).

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \quad U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)
\]
We next show how to compute \( \text{COUNT}(Q_4) \).

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \\
U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b) \\
U_3(d) = \sum_{e \in \text{Dom}(E)} V_2(e) \cdot E_4(d, e)
\]
We next show how to compute $\text{COUNT}(Q_4)$.

$$U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a)$$

$$U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)$$

$$U_3(d) = \sum_{e \in \text{Dom}(E)} V_2(e) \cdot E_4(d, e)$$

$$U_4(c) = \sum_{d \in \text{Dom}(D)} E_3(c, d) \cdot U_3(d)$$
Example: Factorized Aggregate Computation

We next show how to compute $\text{COUNT}(Q_4)$.

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \\
U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b) \\
U_3(d) = \sum_{e \in \text{Dom}(E)} V_2(e) \cdot E_4(d, e) \\
U_4(c) = \sum_{d \in \text{Dom}(D)} E_3(c, d) \cdot U_3(d) \\
U_5 = \sum_{c \in \text{Dom}(C)} U_2(c) \cdot U_4(c)
\]
Functional Aggregate Queries
**Functional Aggregate Query**

FAQ generalizes factorized aggregate computation to a host of problems.

We use the following notation:

- \( X_i \) denote variables, \( i \in [n] = \{1, \ldots, n\} \),
- \( x_i \) are values in discrete domain \( \text{Dom}(X_i) \),
- \( x = (x_1, \ldots, x_n) \in \text{Dom}(X_1) \times \cdots \times \text{Dom}(X_n) \),
- For any \( S \subseteq [n] \),
  \[
  x_S = (x_i)_{i \in S} \in \prod_{i \in S} \text{Dom}(X_i)
  \]
- e.g. \( x_{\{2,5,8\}} = (x_2, x_5, x_8) \in \text{Dom}(X_2) \times \text{Dom}(X_5) \times \text{Dom}(X_8) \)
**Functional Aggregate Query: The Problem**

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

All functions have the same range \( D \)
**Functional Aggregate Query: The Input**

- **n variables** $X_1, \ldots, X_n$
- **a multi-hypergraph** $\mathcal{H} = (\mathcal{V}, \mathcal{E})$
  - Each vertex is a variable (notation overload: $\mathcal{V} = [n]$)
  - To each hyperedge $S \in \mathcal{E}$ there corresponds a *factor* $\psi_S$

$$
\psi_S : \prod_{i \in S} \text{Dom}(X_i) \to D
$$

All functions have the same range $D$

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

$n = 4$
**Functional Aggregate Query: The Input**

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4,3} \psi_{1,3,4} \psi_{1,4} \]

- **n variables** \( X_1, \ldots, X_n \)
- **a multi-hypergraph** \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \)
  - Each vertex is a variable (notation overload: \( \mathcal{V} = [n] \))
  - To each hyperedge \( S \in \mathcal{E} \) there corresponds a factor \( \psi_S \)

\[ \psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D \]

All functions have the same range \( D \)

\[ V = \{1, 2, 3, 4\} \]

\[ E = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\} \]
Functional Aggregate Query: The Input

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

- **n variables** \( X_1, \ldots, X_n \)
- a multi-hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \)
  - Each vertex is a variable (notation overload: \( \mathcal{V} = [n] \))
  - To each hyperedge \( S \in \mathcal{E} \) there corresponds a factor \( \psi_S \)
    \[
    \psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D
    \]

\( \mathcal{V} = \{1, 2, 3, 4\} \)
\( \mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\} \)
**Functional Aggregate Query: The Input**

- **n variables** $X_1, \ldots, X_n$
- a multi-*hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$
  - Each vertex is a variable (notation overload: $\mathcal{V} = [n]$)
  - To each hyperedge $S \in \mathcal{E}$ there corresponds a *factor* $\psi_S$

$$\psi_S : \prod_{i \in S} \text{Dom}(X_i) \to \mathcal{D}$$

- All functions have the same range $\mathcal{D}$

\[ \mathcal{V} = \{1, 2, 3, 4\} \]
\[ \mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\} \]
**Functional Aggregate Query: The Input**

- **n variables** $X_1, \ldots, X_n$
- **a multi-hypergraph** $\mathcal{H} = (\mathcal{V}, \mathcal{E})$
  - Each vertex is a variable (notation overload: $\mathcal{V} = [n]$)
  - To each hyperedge $S \in \mathcal{E}$ there corresponds a factor $\psi_S$

$$\psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D$$

- $\mathcal{V} = \{1, 2, 3, 4\}$
- $\mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\}$

---

$$\varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4}$$

---

All functions have the same range $D$
**Functional Aggregate Query: The Input**

\[
\varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4}
\]

- **n variables** \(X_1, \ldots, X_n\)
- **a multi-hypergraph** \(\mathcal{H} = (\mathcal{V}, \mathcal{E})\)
  - Each vertex is a variable (notation overload: \(\mathcal{V} = [n]\))
  - To each hyperedge \(S \in \mathcal{E}\) there corresponds a *factor* \(\psi_S\)
    \[
    \psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D
    \]
    \(\mathbb{R}_+, \{\text{true, false}\}, \{0, 1\}, 2^\mathcal{U}, \text{etc.}\)
- **a set** \(F \subseteq \mathcal{V}\) of *free variables* (wlog, \(F = [f] = \{1, \ldots, f\}\))
Functional Aggregate Query: The Output

\[ \varphi(x_3) = \sum x_1 \prod x_2 \max x_4 \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

- Compute the function \( \varphi : \prod_{i \in F} \text{Dom}(X_i) \rightarrow D \).
Functional Aggregate Query: The Output

Compute the function $\varphi : \prod_{i \in F} \text{Dom}(X_i) \rightarrow D$.

- $\varphi$ defined by the FAQ-expression

$$
\varphi(x_{[f]}) = \bigoplus_{x_{f+1} \in \text{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \text{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \text{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(x_S)
$$
Functional Aggregate Query: The Output

Compute the function \( \varphi : \prod_{i \in F} \text{Dom}(X_i) \to D \).

\( \varphi \) defined by the FAQ-expression

\[
\varphi(x_{[f]}) = \bigoplus_{x_{f+1} \in \text{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \text{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \text{Dom}(X_n)}^{(n)} \otimes \psi_S(x_S)
\]

For each \( \bigoplus^{(i)} \)
Functional Aggregate Query: The Output

- Compute the function $\varphi : \prod_{i \in F} \text{Dom}(X_i) \rightarrow D$.
- $\varphi$ defined by the FAQ-expression

$$\varphi(x) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4}$$

- For each $\bigoplus^{(i)}$
  - Either $(D, \bigoplus^{(i)}, \otimes)$ is a commutative semiring
**Functional Aggregate Query: The Output**

\[ \varphi(x_3) = \sum x_1 \prod x_2 \max x_4 \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

- All functions have the same range \( D \)

- Compute the function \( \varphi : \prod_{i \in F} \text{Dom}(X_i) \to D \).

- \( \varphi \) defined by the **FAQ-expression**

\[
\varphi(x[f]) = \bigoplus_{x_{f+1} \in \text{Dom}(X_{f+1})}^{f+1} \cdots \bigoplus_{x_{n-1} \in \text{Dom}(X_{n-1})}^{n-1} \bigoplus_{x_{n} \in \text{Dom}(X_{n})}^{n} \otimes \psi_{S}(x_S)
\]

- For each \( \bigoplus^{(i)} \)
  - Either \( (D, \bigoplus^{(i)}, \otimes) \) is a **commutative semiring**
  - Or \( \bigoplus^{(i)} = \otimes \)
Semirings

- $(D, \oplus, \otimes)$ is a **commutative semiring** when
  
  **Additive identity** \( 0 \in D : 0 \oplus e = e \oplus 0 = e \)
  
  **Multiplicative identity** \( 1 \in D : 1 \otimes e = e \otimes 1 = e \)
  
  **Annihilation by 0** \( 0 \otimes e = e \otimes 0 = 0 \)
  
  **Distributive law** \( a \otimes b \oplus a \otimes c = a \otimes (b \oplus c) \)
(D, ⊕, ⊗) is a commutative semiring when

**Additive identity**  \( 0 \in D : 0 \oplus e = e \oplus 0 = e \)

**Multiplicative identity**  \( 1 \in D : 1 \otimes e = e \otimes 1 = e \)

**Annihilation by 0**  \( 0 \otimes e = e \otimes 0 = 0 \)

**Distributive law**  \( a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c \)

Common examples (there are many more!)

- **Boolean**  \( (\{true, false\}, \lor, \land) \)
- **sum-product**  \( (\mathbb{R}, +, \times) \)
- **max-product**  \( (\mathbb{R}_+, \max, \times) \)
- **set**  \( (2^U, \cup, \cap) \)
Problem (SumProduct)

*Given a commutative semiring \((D, \oplus, \otimes)\), compute the function*

\[
\varphi(x_1, \ldots, x_f) = \bigoplus_{x_{f+1}} \bigoplus_{x_{f+2}} \cdots \bigoplus_{x_n} \bigotimes_{S \in \mathcal{E}} \psi_S(x_S)
\]
Problem (SumProduct)

*Given a commutative semiring* \((D, \oplus, \otimes)\), *compute the function*

\[
\varphi(x_1, \ldots, x_f) = \bigoplus_{x_{f+1}} \bigoplus_{x_{f+2}} \cdots \bigoplus_{x_n} \bigotimes_{S \in E} \psi_S(x_S)
\]

- **SumProduct**
  - Rina Dechter (Artificial Intelligence 1999 and earlier)

- **Marginalize a Product Function**
Many examples for SumProduct

- $\left(\{\text{true, false}\}, \lor, \land\right)$
  - Constraint satisfaction problems
  - Boolean conjunctive query evaluation
  - SAT
  - $k$-colorability
  - etc.

- $\left(U, \cup, \cap\right)$
  - Conjunctive query evaluation

- $\left(\mathbb{R}, +, \times\right)$
  - Permanent
  - DFT
  - Inference in graphical models
  - #CSP
  - Aggregates in DB

- $\left(\mathbb{R}_+, \max, \times\right)$
  - MAP queries in graphical models
Outline

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Part 4. Open Problems
In-Database Optimization aka In-Database Analytics

Why in-database analytics?

1. Bring analytics close to data
   \[ \Rightarrow \text{Save non-trivial export/import time} \]

2. Large chunks of analytics code can be rewritten into SumProduct FAQs
   \[ \Rightarrow \text{Use scalable systems and low complexity for query processing} \]

Hot topic in the current DB research & industry landscape:

- Very recent tutorials and research agenda \([\text{A17}, \text{KBY17}, \text{PRWZ17}]\)
- This tutorial highlights our recent work \([\text{SOC16}, \text{ANNOS17}]\)
In-database vs. Out-of-database Learning

- $h$ and $g$ are functions over features and respectively model parameters
- $\theta^*$ are the parameters of the learned model
Our Approach to In-Database Analytics

Unified in-database analytics solution for a host of optimization problems.

Used by LogicBlox retail-planning and forecasting applications

- Typical databases have weekly sales, promotions, and products

- Training dataset = Result of a feature extraction query over the database

- Task = Train parameterized model to predict, e.g., additional demand generated for a product due to promotion

- Training algorithm = First-order optimization algorithm, e.g., batch or stochastic gradient descent, whose convergence rates are dimension-free

Commercial database management system LogicBlox

- One platform for OLAP and OLTP, descriptive and predictive analytics

- Datalog++ as programming language for applications
Plan for Tutorial Part 3

- We will first introduce the main technical ideas via an example
  - Train a linear regression model using batch gradient descent
  - Express gradient computation as database queries
  - Re-parameterize the model under functional dependencies

- We will then discuss a generalization
  - Polynomial regression, factorization machines, classification

- We will conclude with complexity analysis of in-database analytics
  - Model training faster than computing the input to external ML library!
Typical Retail Example
Typical Retail Example

- Database $I = (R_1, R_2, R_3, R_4, R_5)$
- Feature selection query $Q$:

$$Q(sku, store, color, city, country, unitsSold) \leftarrow$$

$$R_1(sku, store, day, unitsSold), R_2(sku, color),$$
$$R_3(day, quarter), R_4(store, city), R_5(city, country).$$

- Free variables
  - Categorical (qualitative): $F = \{sku, store, color, city, country\}$.
  - Continuous (quantitative): $unitsSold$.
- Bounded variables
  - Categorical (qualitative): $B = \{day, quarter\}$

We learn the ridge linear regression model $\langle \theta, x \rangle = \sum_{f \in F} \langle \theta_f, x_f \rangle$ over $D = Q(I)$ with feature vector $x$ and response $y_{unitsSold}$.

The parameters $\theta$ are obtained by minimizing the square loss function:

$$J(\theta) = \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y_{unitsSold})^2 + \|\theta\|_2^2$$
Recap: One-hot encoding of categorical variables

- **Continuous** variables are mapped to scalars
  - $y_{\text{unitsSold}} \in \mathbb{R}.$

- **Categorical** variables are mapped to indicator vectors
  - Say variable country has categories vietnam and england.
  - The variable country is then mapped to an indicator vector
    \[ x_{\text{country}} = [x_{\text{vietnam}}, x_{\text{england}}]^\top \in (\{0, 1\}^2)^\top. \]
  - $x_{\text{country}} = [0, 1]^\top$ for a tuple with country = ‘‘england’’

One-hot encoding leads to very wide training datasets and many 0-values.
Recap: Role of the Least Square Loss Function

Goal: Describe a linear relationship \( \text{fun}(x) = \theta_1 x + \theta_0 \) between variables \( x \) and \( y = \text{fun}(x) \), so we can estimate new \( y \) values given new \( x \) values.

- We are given \( n \) (black) data points \((x_i, y_i)_{i \in [n]}\)
- We would like to find a (red) regression line \( \text{fun}(x) \) such that the (green) error \( \sum_{i \in [n]} (\text{fun}(x_i) - y_i)^2 \) is minimized
- The role of the \( \ell_2 \)-regularization \( \|\theta\|_2^2 = \theta_0^2 + \theta_1^2 \) is to avoid over/under-fitting. It gives preference to functions \( \text{fun} \) with smaller norms.
From Optimization to SumProduct FAQ Queries

We can solve $\theta^* := \arg \min_{\theta} J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

$$\theta := \theta - \alpha \cdot \nabla J(\theta).$$
We can solve $\theta^* := \arg\min_{\theta} J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

$$\theta := \theta - \alpha \cdot \nabla J(\theta).$$

Define the matrix $\Sigma = (\sigma_{ij})_{i,j \in |F|}$, the vector $c = (c_i)_{i \in |F|}$, and the scalar $s_Y$:

$$\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top \quad c_i = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot x_i \quad s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$
We can solve $\theta^* := \arg\min_\theta J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

$$\theta := \theta - \alpha \cdot \nabla J(\theta).$$

Define the matrix $\Sigma = (\sigma_{ij})_{i,j \in |F|}$, the vector $c = (c_i)_{i \in |F|}$, and the scalar $s_Y$:

$$\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top \quad \quad c_i = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot x_i \quad \quad s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$

Then,

$$J(\theta) = \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \|\theta\|_2^2$$

$$= \frac{1}{2} \theta^\top \Sigma \theta - \langle \theta, c \rangle + \frac{s_Y}{2} + \frac{\lambda}{2} \|\theta\|_2^2$$

$$\nabla J(\theta) = \Sigma \theta - c + \lambda \theta$$
Expressing $\Sigma$, $c$, $s_Y$ as SumProduct FAQ Queries

FAQ queries for $\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top$ (w/o factor $\frac{1}{|D|}$):

- $x_i$, $x_j$ continuous $\Rightarrow$ FAQ query with no free variable

\[ \psi_{ij} = \sum_{f \in F: a_f \in \text{Dom}(x_f)} \sum_{b \in B: a_b \in \text{Dom}(x_b)} a_i \cdot a_j \cdot \prod_{k \in [5]} 1_{R_k(a_{S(R_k)})} \]

- $x_i$ categorical, $x_j$ continuous $\Rightarrow$ FAQ query with one free variable

\[ \psi_{ij}[a_i] = \sum_{f \in F - \{i\}: a_f \in \text{Dom}(x_f)} \sum_{b \in B: a_b \in \text{Dom}(x_b)} a_j \cdot \prod_{k \in [5]} 1_{R_k(a_{S(R_k)})} \]

- $x_i$, $x_j$ categorical $\Rightarrow$ FAQ query with two free variables

\[ \psi_{ij}[a_i, a_j] = \sum_{f \in F - \{i, j\}: a_f \in \text{Dom}(x_f)} \sum_{b \in B: a_b \in \text{Dom}(x_b)} \prod_{k \in [5]} 1_{R_k(a_{S(R_k)})} \]

$S(R_k)$ is the set of variables of $R_k$; $a_{S(R_k)}$ is a tuple in relation $R_k$; $1_E$ is the Kronecker delta that evaluates to 1 (0) whenever the event $E$ (not) holds.
Expressing $\Sigma$, $c$, $s_Y$ as SQL Queries

SQL queries for $\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top$ (w/o factor $\frac{1}{|D|}$):

- $x_i$, $x_j$ continuous $\Rightarrow$ SQL query with no group-by attribute

$$SELECT \text{SUM}(x_i x_j) \text{ FROM } D;$$

- $x_i$ categorical, $x_j$ continuous $\Rightarrow$ SQL query with one group-by attribute

$$SELECT x_i, \text{SUM}(x_j) \text{ FROM } D \text{ GROUP BY } x_i;$$

- $x_i$, $x_j$ categorical $\Rightarrow$ SQL query with two free variables

$$SELECT x_i, x_j, \text{SUM}(1) \text{ FROM } D \text{ GROUP BY } x_i, x_j;$$

- $\Sigma$, $c$, $s_Y$ are all aggregates that can be computed inside the database!

- We avoid one-hot/sparse encoding of the input data.
Consider the functional dependency $\text{city} \rightarrow \text{country}$

- There is one country for each city.

Assume we have:

- Vietnam, England as categories for country
- Saigon, Hanoi, Oxford, Leeds, Bristol as categories for city

The one-hot encoding enforces the following identities:

- $x_{\text{vietnam}} = x_{\text{saigon}} + x_{\text{hanoi}}$
  
  That is: If country is Vietnam, then city is either Saigon or Hanoi
  
  if $x_{\text{vietnam}} = 1$ then either $x_{\text{saigon}} = 1$ or $x_{\text{hanoi}} = 1$

- $x_{\text{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}$
  
  That is: If country is England, then city is either Oxford, Leeds, or Bristol
  
  if $x_{\text{england}} = 1$ then either $x_{\text{oxford}} = 1$ or $x_{\text{leeds}} = 1$ or $x_{\text{bristol}} = 1$
Dimensionality Reduction with Functional Dependencies

- **Identities due to one-hot encoding**
  
  \[ x_{\text{vietnam}} = x_{\text{saigon}} + x_{\text{hanoi}} \]
  
  \[ x_{\text{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}} \]

- **Encode \( x_{\text{country}} \) as \( x_{\text{country}} = Rx_{\text{city}} \), where**

  \[
  R = \begin{bmatrix}
  1 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 1 \\
  \end{bmatrix}
  \]

  For instance, if \( \text{city} \) is \( \text{saigon} \), i.e., \( x_{\text{city}} = [1, 0, 0, 0, 0]^T \), then \( \text{country} \) is \( \text{vietnam} \), i.e., \( x_{\text{country}} = Rx_{\text{city}} = [1, 0]^T \).

  \[
  \begin{bmatrix}
  1 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 1 \\
  \end{bmatrix}
  \begin{bmatrix}
  1 \\
  0 \\
  0 \\
  0 \\
  \end{bmatrix}
  =
  \begin{bmatrix}
  1 \\
  0 \\
  \end{bmatrix}
  \]
Dimensionality Reduction with Functional Dependencies

- Functional dependency: city → country
- \(x_{\text{country}} = Rx_{\text{city}}\)
- Replace all occurrences of \(x_{\text{country}}\) by \(Rx_{\text{city}}\):

\[
\sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, x_{\text{country}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle = \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, Rx_{\text{city}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle = \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \left( R^\top \theta_{\text{country}} + \theta_{\text{city}}, x_{\text{city}} \right) \gamma_{\text{city}}
\]

We avoid computing aggregates over \(x_{\text{country}}\).
We reparameterize the problem and ignore parameters \(\theta_{\text{country}}\).

What about the penalty term in the loss function?
Dimensionality Reduction with Functional Dependencies

- Functional dependency: \( \text{city} \rightarrow \text{country} \)
- \( x_{\text{country}} = Rx_{\text{city}} \)
- Replace all occurrences of \( x_{\text{country}} \) by \( Rx_{\text{city}} \):

\[
\sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, x_{\text{country}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle
= \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, Rx_{\text{city}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle
= \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \left( R^\top \theta_{\text{country}} + \theta_{\text{city}}, x_{\text{city}} \right)
\]

- We avoid computing aggregates over \( x_{\text{country}} \).
- We reparameterize the problem and ignore parameters \( \theta_{\text{country}} \).
- What about the penalty term in the loss function?
Dimensionality Reduction with Functional Dependencies

- **Functional dependency**: city → country

- \( x_{\text{country}} = Rx_{\text{city}} \)

- \( \gamma_{\text{city}} = R^T \theta_{\text{country}} + \theta_{\text{city}} \)

- Rewrite the penalty term

\[
\|\theta\|_2^2 = \sum_{j \neq \text{city}} \|\theta_j\|_2^2 + \|\gamma_{\text{city}} - R^T \theta_{\text{country}}\|_2^2 + \|\theta_{\text{country}}\|_2^2
\]

- "Optimize out" \( \theta_{\text{country}} \) by expressing it in terms of \( \gamma_{\text{city}} \):

\[
\theta_{\text{country}} = (I_{\text{country}} + RR^T)^{-1} R \gamma_{\text{city}} = R(I_{\text{city}} + R^T R)^{-1} \gamma_{\text{city}}
\]

\( I_{\text{country}} \) is the order-\( N_{\text{country}} \) identity matrix and similarly for \( I_{\text{city}} \).

- The penalty term becomes

\[
\|\theta\|_2^2 = \sum_{j \neq \text{city}} \|\theta_j\|_2^2 + \langle (I_{\text{city}} + R^T R)^{-1} \gamma_{\text{city}}, \gamma_{\text{city}} \rangle
\]
The General Picture
General Problem Formulation

A typical machine learning task is to solve $\theta^* := \arg \min_{\theta} J(\theta)$, where

$$J(\theta) := \sum_{(x,y) \in D} \mathcal{L} \left( \langle g(\theta), h(x) \rangle, y \right) + \Omega(\theta).$$

- $\theta = (\theta_1, \ldots, \theta_p) \in \mathbb{R}^p$ are parameters
- functions $g : \mathbb{R}^p \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ for $n$ numeric features, $m > 0$
  - $g = (g_j)_{j \in [m]}$ is a vector of multivariate polynomials
  - $h = (h_j)_{j \in [m]}$ is a vector of multivariate monomials
- $\mathcal{L}$ is a loss function, $\Omega$ is the regularizer
- $D$ is the training dataset with features $x$ and response $y$.

Example problems: ridge linear regression, degree-$d$ polynomial regression, degree-$d$ factorization machines; logistic regression, SVM; PCA.
Special Case: Ridge Linear Regression

General problem formulation:

\[
J(\theta) := \sum_{(x, y) \in D} \mathcal{L}(\langle g(\theta), h(x) \rangle, y) + \Omega(\theta).
\]

Under

- square loss \( \mathcal{L} \), \( \ell_2 \)-regularization,
- data points \( x = (x_0, x_1, \ldots, x_n, y) \),
- \( p = n + 1 \) parameters \( \theta = (\theta_0, \ldots, \theta_n) \),
- \( x_0 = 1 \) corresponds to the bias parameter \( \theta_0 \)
- \( g \) and \( h \) identity functions \( g(\theta) = \theta \) and \( h(x) = x \),

we obtain the following formulation for ridge linear regression:

\[
J(\theta) := \frac{1}{2|D|} \sum_{(x, y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \| \theta \|_2^2.
\]
Special Case: Degree-$d$ Polynomial Regression

General problem formulation:

$$J(\theta) := \sum_{(x, y) \in D} L \left( \langle g(\theta), h(x) \rangle, y \right) + \Omega(\theta).$$

Under

- square loss $L$, $\ell_2$-regularization,
- data points $x = (x_0, x_1, \ldots, x_n, y),$
- $p = m = 1 + n + n^2 + \cdots + n^d$ parameters $\theta = (\theta_a),$ where $a = (a_1, \ldots, a_n)$ is a tuple of non-negative integers such that $\|a\|_1 \leq d.$
- $g(\theta) = \theta,$
- the components of $h$ are given by $h_a(x) = \prod_{i=1}^{n} x_i^{a_i}.$

we obtain the following formulation for polynomial regression:

$$J(\theta) := \frac{1}{2|D|} \sum_{(x, y) \in D} \left( \langle g(\theta), h(x) \rangle - y \right)^2 + \frac{\lambda}{2} \|\theta\|_2^2.$$
Special Case: Factorization Machines

Under

- square loss $\mathcal{L}$, $\ell_2$-regularization,
- data points $\mathbf{x} = (x_0, x_1, \ldots, x_n, y)$,
- $p = m = 1 + n + r \cdot n$ parameters and $m = 1 + n + \binom{n}{2}$ features

we obtain the following formulation for degree-2 rank-$r$ factorization machines:

$$ J(\theta) := \frac{1}{2|D|} \sum_{(x,y) \in D} \left( \sum_{i=0}^{n} \theta_i x_i + \sum_{\{i,j\} \in \binom{[n]}{2}} \theta_i^{(\ell)} \theta_j^{(\ell)} x_i x_j - y \right)^2 + \frac{\lambda}{2} \|\theta\|_2^2. $$

where

$$ h_S(\mathbf{x}) = \prod_{i \in S} x_i, \text{ for } S \subseteq [n], |S| \leq 2 $$

$$ g_S(\theta) = \begin{cases} 
\theta_0 & \text{when } |S| = 0 \\
\theta_i & \text{when } S = \{i\} \\
\sum_{\ell=1}^{r} \theta_i^{(\ell)} \theta_j^{(\ell)} & \text{when } S = \{i,j\}.
\end{cases} $$
Special Case: Classification methods

Examples: support vector machines, logistic regression, Adaboost

- Typically, the regularizer is $\frac{\lambda}{2}\|\theta\|_2^2$

- The response is now binary: $y \in \{\pm 1\}$

- The loss function $\mathcal{L}(\gamma, y)$, where $\gamma := \langle g(\theta), h(x) \rangle$, takes the form:
  - $\mathcal{L}(\gamma, y) = \max\{1 - y\gamma, 0\}$ for support vector machines (SVM),
  - $\mathcal{L}(\gamma, y) = \log(1 + e^{-y\gamma})$ for logistic regression, and
  - $\mathcal{L}(\gamma, y) = e^{-y\gamma}$ for Adaboost.
Batch Gradient Descent (BGD)

Repeatedly update $\theta$ in the direction of the gradient until convergence

$\theta \leftarrow$ a random point;

while not converged yet do

$\alpha \leftarrow$ next step size;

d $\leftarrow \nabla J(\theta);

while $(J(\theta - \alpha \cdot d) \geq J(\theta) - \frac{\alpha}{2} \cdot \|d\|_2^2)$ do $\alpha \leftarrow \alpha/2;$ // line search

$\theta \leftarrow \theta - \alpha \cdot d;$

end

BGD needs:

- Computation of the gradient vector $\nabla J(\theta)$
  - Its data-dependent component is computed once for all iterations

- Point evaluation $J(\theta)$
  - A few times per iteration to adjust $\alpha$ using line search
Compute Parameters $\theta$ using BGD

Immediate extension of the linear regression case discussed before.

Define the matrix $\Sigma = (\sigma_{ij})_{i,j \in [m]}$, the vector $c = (c_i)_{i \in [m]}$, and the scalar $s_Y$ by

$$\Sigma = \frac{1}{|D|} \sum_{(x,y) \in D} h(x)h(x)^\top$$

$$c = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot h(x)$$

$$s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$ 

Under square loss $\mathcal{L}$ and $\ell_2$-regularization:

$$J(\theta) = \frac{1}{2} g(\theta)^\top \Sigma g(\theta) - \langle g(\theta), c \rangle + \frac{s_Y}{2} + \frac{\lambda}{2} \|\theta\|^2$$

$$\nabla J(\theta) = \frac{\partial g(\theta)^\top}{\partial \theta} \Sigma g(\theta) - \frac{\partial g(\theta)^\top}{\partial \theta} c + \lambda \theta$$
### Summing Up

**Insight #1:**
- \( \Sigma, c, s \_Y \) are queries that can be computed **inside the database**!
  - They can take **much less time** than computing the feature extraction query.

**Insight #2:**
- The training dataset has repeating data blocks as it satisfies the join dependencies given by the feature extraction query.
  - A **factorized** training dataset avoids this redundancy.

**Insight #3:**
- The training dataset has many functional dependencies in practice.
  - First learn a **smaller, reparameterized model** whose features functionally determine the left-out features, then map it back to the original model with both functionally determining and determined parameters.
Zoom-in: **In-database** vs. **Out-of-database** Learning

**Feature extraction** query $R_1 \times \ldots \times R_k$

DB

Queries:

- $\sigma_{11}$
- $\ldots$
- $\sigma_{ij}$
- $\ldots$
- $c_1$
- $\ldots$

Model reformulation

Query optimizer

Factorized query evaluation

Cost $\leq N_{\text{faqw}} \ll |D|$
Complexity Analysis
Complexity Analysis: The General Case

Complexity of learning models falls back to factorized computation of aggregates over joins \[\text{[BKOZ13,OZ15,SOC16,ANR16]}\]

Let:

- \((V, E) = \text{hypergraph of } Q\)
- \(N = \max_{R \in I} |R|\)
- \(|\sigma_{ij}| = \text{size of the sparse representation of the } \sigma_{ij} \text{ tensor}\)
- \(faqw(i, j) = \text{FAQ-width of the query that expresses } \sigma_{ij} \text{ over } Q\)

The tensors \(\sigma_{ij}\) and \(c_j\) can be sparsely represented by queries with group-by variables and can be computed in time

\[
O \left( |V|^2 \cdot |E| \cdot \sum_{i,j \in [m]} (N^{faqw(i,j)} + |\sigma_{ij}|) \cdot \log N \right).
\]
Complexity Analysis: Continuous Features Only

Complexity in the general case:

$$O \left( |\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [m]} (N_{faqw(i,j)} + |\sigma_{ij}|) \cdot \log N \right).$$

Complexity in case all features are continuous:

$$O(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot m^2 \cdot N^{fhtw} \cdot \log N).$$

In this case, $faqw(i,j)$ becomes the fractional hypertree width $fhtw$ of $Q$. 
Complexity Analysis: Comparison with State of the Art

Let:
- \( d \) = degree of polynomial regression model
- \( c \) = max number of variables in any monomial in \( h \); \( c \leq d \)
- \( \rho^* \) = fractional edge cover number of query \( Q \)

Comparison against state of the art:
- \( \text{faqw}(i, j) \leq \text{fhtw} + c - 1 \) and \( |\sigma_{ij}| \leq \min\{|D|, N^c\} \).
- For any query \( Q \) with \( \rho^* > \text{fhtw} + c - 1 \), there are infinitely many database instances of size \( N \) for which
  \[
  \lim_{N \to \infty} \frac{|D|}{\sum_{i,j \in [m]} \left( N^{\text{faqw}(i, j)} + |\sigma_{ij}| \right) \log N} = \infty.
  \]
- Computing \( \sigma_{ij} \) for degree-\( d \) polynomial regression takes
  \[
  O(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot m^2 \cdot N^{\text{fhtw} + 2d} \log N).
  \]
  under one-hot encoding of categorical variables.
Outline

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Part 4. Open Problems
Adaptive Join Processing
Adaptive Join Processing

Idea: Use different variable orders (hypertree decompositions) for different databases and even partitions of the input relations. [ANS17]

Consider the (4-cycle) join: \( R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1) \).

There are two (hyper)tree decompositions \( T_1 \) and \( T_2 \):

\[
\begin{array}{c}
A_1 \\ R
\end{array} \quad \begin{array}{c}
A_2 \\ S
\end{array} \quad \begin{array}{c}
A_3 \\ W
\end{array} \quad \begin{array}{c}
A_4 \\ T
\end{array}
\]

Each bag gets covered by two relations \( \Rightarrow \) each bag has \( \rho^* = 2 \Rightarrow fhtw = 2! \)

- For \( T_1 \): \( R \) and \( S \) cover \( \{A_1, A_2, A_3\} \), \( T \) and \( W \) cover \( \{A_1, A_3, A_4\} \)

Usual construction for showing lower bounds, where all relations have size \( N \):

- Take \( R = [N] \times \{1\} \) and \( S = \{1\} \times [N] \), then \( |R \Join S| = N^2 \). 


Adaptive Join Processing

Idea: Use different variable orders (hypertree decompositions) for different databases and even partitions of the input relations. \[\text{[ANS17]}\]

Consider the (4-cycle) join: \(R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)\).

There are two (hyper)tree decompositions \(T_1\) and \(T_2\):

Each bag gets covered by two relations \(\Rightarrow\) each bag has \(\rho^* = 2 \Rightarrow fhtw = 2!\)
- For \(T_1\): \(R\) and \(S\) cover \(\{A_1, A_2, A_3\}\), \(T\) and \(W\) cover \(\{A_1, A_3, A_4\}\)

Usual construction for showing lower bounds, where all relations have size \(N\):
- Take \(R = [N] \times \{1\}\) and \(S = \{1\} \times [N]\), then \(|R \Join S| = N^2\).
- These \(R\) and \(S\) do not yield \(N^2\) for \(T_2\)!: \(R\) and \(W\) cover \(\{A_4, A_1, A_2\}\) with \(|R \Join W| = N\), while \(S\) and \(T\) cover \(\{A_2, A_3, A_4\}\) with \(|S \Join T| = N\).
Adaptive Join Processing

**Definition:** A value $x$ for $X$ is heavy in $U$ if $|\sigma_{X=x}(U)| \leq \sqrt{|U|}$.

Let $R = R_\ell \cup R_h$, where $R_h = \{(a, b) \mid (a, b) \in R, a \text{ is heavy}\}$.

Then, $|\pi_{A_1}(R_h)| \leq \sqrt{N}$ and $\forall a \in \text{Dom}(A_1) : |\sigma_{A_1=a}(R_\ell)| \leq \sqrt{N}$.

Similarly, $T = T_\ell \cup T_h$, where $T_h = \{(a, b) \mid (a, b) \in T, a \text{ is heavy}\}$.

We split the query $Q$ into a union of disjoint, more specialized queries:

\[
Q = Q_1; Q_2; Q_3
\]

\[
Q_1 = R_h(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)
\]

\[
Q_2 = R_\ell(A_1, A_2), S(A_2, A_3), T_h(A_3, A_4), W(A_4, A_1)
\]

\[
Q_3 = R_\ell(A_1, A_2), S(A_2, A_3), T_\ell(A_3, A_4), W(A_4, A_1).
\]
Adaptive Join Processing

**Definition:** A value $x$ for $X$ is heavy in $U$ if $|\sigma_{x=\cdot}(U)| \leq \sqrt{|U|}$.

Let $R = R_\ell \cup R_h$, where $R_h = \{(a, b) \mid (a, b) \in R, a \text{ is heavy}\}$.

Then, $|\pi_{A_1}(R_h)| \leq \sqrt{N}$ and $\forall a \in \text{Dom}(A_1): |\sigma_{A_1=a}(R_\ell)| \leq \sqrt{N}$.

Similarly, $T = T_\ell \cup T_h$, where $T_h = \{(a, b) \mid (a, b) \in T, a \text{ is heavy}\}$.

We split the query $Q$ into a union of disjoint, more specialized queries:

$$Q = Q_1; Q_2; Q_3$$

$$Q_1 = R_h(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

$$Q_2 = R_\ell(A_1, A_2), S(A_2, A_3), T_h(A_3, A_4), W(A_4, A_1)$$

$$Q_3 = R_\ell(A_1, A_2), S(A_2, A_3), T_\ell(A_3, A_4), W(A_4, A_1).$$

Each of $Q_1, Q_2, Q_3$ has $fhtw = 3/2$, hence $Q$ has $fhtw = 3/2$. 
Adaptive Join Processing

\[ Q_1 = R_h(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1) \]

We use decomposition \( T_1 \) for \( Q_1 \):

- We use \( |\pi_{A_1}(R_h) \bowtie S| \leq N^{3/2} \) to cover the bag \( \{A_1, A_2, A_3\} \)
- We use \( |\pi_{A_1}(R_h) \bowtie T| \leq N^{3/2} \) to cover the bag \( \{A_1, A_3, A_4\} \)

\[ Q_2 = R_\ell(A_1, A_2), S(A_2, A_3), T_h(A_3, A_4), W(A_4, A_1) \]

We use decomposition \( T_1 \) for \( Q_2 \):

- We use \( |\pi_{A_3}(T_h) \bowtie R_\ell| \leq N^{3/2} \) to cover the bag \( \{A_1, A_2, A_3\} \)
- We use \( |\pi_{A_3}(T_h) \bowtie W| \leq N^{3/2} \) to cover the bag \( \{A_1, A_3, A_4\} \)

\[ Q_3 = R_\ell(A_1, A_2), S(A_2, A_3), T_\ell(A_3, A_4), W(A_4, A_1). \]

We use decomposition \( T_2 \) for \( Q_3 \):

- We use \( \sum_{(a_1, a_4) \in W} |\{a_2 \mid (a_1, a_2) \in R_\ell\}| \leq \sum_{(a_1, a_4) \in W} \sqrt{N} \leq N^{3/2} \) to cover the bag \( \{A_1, A_2, A_4\} \)
- We use \( \sum_{(a_2, a_3) \in S} |\{a_4 \mid (a_3, a_4) \in T_\ell\}| \leq \sum_{(a_2, a_3) \in S} \sqrt{N} \leq N^{3/2} \) to cover the bag \( \{A_2, A_3, A_4\} \)
One-join-at-a-time vs All-joins-at-once
One-join-at-a-time vs All-joins-at-once

- Standard query evaluation considers one join at a time, yet...
- Worst-case optimal join algorithms for *listing representation* have to consider all joins at once. \[\text{[NRR13]}\]

> Intermediate join results may be larger than the final query result!

**Triangle query Q**

\[R_1(A, B), R_2(A, C), R_3(B, C)\]

For databases \(D\) with \(|R_i| = N(\forall i \in [3])\): \(\rho^*(Q) = 3/2\) and \(|Q(D)| \in \Theta(N^{3/2})\).

Yet there are databases, for which the join of any two relations has size \(\Omega(N^2)\).
Why traditional join-at-a-time plans are suboptimal

Query $Q$

$R_1(A, B), R_2(A, C), R_3(B, C)$

**Database**

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
<td>$C$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_0$</td>
<td>$a_0$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_0$</td>
<td>$a_0$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_m$</td>
<td>$a_0$</td>
<td>$c_m$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_0$</td>
<td>$a_1$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>...</td>
<td>$b_0$</td>
<td>...</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$a_m$</td>
<td>$b_0$</td>
<td>$a_m$</td>
<td>$c_0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B$</th>
<th>$C$</th>
<th>$B$</th>
<th>$C$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$a_0$</td>
</tr>
<tr>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$a_0$</td>
</tr>
<tr>
<td>$b_0$</td>
<td>$c_m$</td>
<td>$b_0$</td>
<td>$c_m$</td>
<td>$a_0$</td>
</tr>
<tr>
<td>$b_1$</td>
<td>$c_0$</td>
<td>$b_1$</td>
<td>$c_0$</td>
<td>$a_0$</td>
</tr>
<tr>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$a_1$</td>
</tr>
<tr>
<td>...</td>
<td>$b_0$</td>
<td>...</td>
<td>$b_0$</td>
<td>...</td>
</tr>
<tr>
<td>$b_m$</td>
<td>$c_0$</td>
<td>$b_m$</td>
<td>$c_0$</td>
<td>...</td>
</tr>
<tr>
<td>$c_0$</td>
<td>$c_0$</td>
<td>$c_0$</td>
<td>$c_0$</td>
<td>...</td>
</tr>
</tbody>
</table>
Why traditional join-at-a-time plans are suboptimal

Query $Q$

$R_1(A, B), R_2(A, C), R_3(B, C)$

Database

Query plans
Why traditional join-at-a-time plans are suboptimal

\[ R_1(A, B), R_2(A, C), R_3(B, C) \]

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline
a_0 & b_0 \\
a_0 & \ldots \\
a_0 & b_m \\
a_1 & b_0 \\
\vdots & b_0 \\
a_m & b_0 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
A & C \\
\hline
a_0 & c_0 \\
a_0 & \ldots \\
a_0 & c_m \\
a_1 & c_0 \\
\vdots & c_0 \\
a_m & c_0 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
B & C \\
\hline
b_0 & c_0 \\
b_0 & \ldots \\
b_0 & c_m \\
b_1 & c_0 \\
\vdots & c_0 \\
b_m & c_0 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
a_0 & b_0 & c_0 \\
a_0 & b_0 & \ldots \\
a_0 & b_0 & c_m \\
a_0 & b_1 & c_0 \\
a_0 & \ldots & c_0 \\
a_0 & b_m & c_0 \\
a_1 & b_0 & c_0 \\
\vdots & b_0 & c_0 \\
a_1 & b_0 & c_0 \\
\hline
\end{array}
\]

\[ 2m + 1 \quad 2m + 1 \quad 2m + 1 \quad 6m + 3 \]
Why traditional join-at-a-time plans are suboptimal

An example of intermediate result of $O(m^2)$ tuples:

The joins $R_1 \Join R_3$ and $R_2 \Join R_3$ also lead to $O(m^2)$ intermediate results.
One-join-at-a-time using Factorized Representations

We can attain worst-case optimality for join-at-a-time processing by factorizing the intermediate results. [CO15]

\[ R_1 \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>(a_0)</td>
<td>(b_0)</td>
<td></td>
</tr>
<tr>
<td>(a_0)</td>
<td>(\ldots)</td>
<td></td>
</tr>
<tr>
<td>(a_0)</td>
<td>(b_m)</td>
<td></td>
</tr>
<tr>
<td>(a_1)</td>
<td>(b_0)</td>
<td></td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(b_0)</td>
<td></td>
</tr>
<tr>
<td>(a_m)</td>
<td>(b_0)</td>
<td></td>
</tr>
</tbody>
</table>
One-join-at-a-time using Factorized Representations

We can attain worst-case optimality for join-at-a-time processing by factorizing the intermediate results. \[\text{[CO15]}\]

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline
a_0 & b_0 \\
a_0 & \ldots \\
a_0 & b_m \\
\hline
a_1 & b_0 \\
\ldots & b_0 \\
a_m & b_0 \\
\hline
\end{array}
\]
One-join-at-a-time using Factorized Representations

We can attain worst-case optimality for join-at-a-time processing by factorizing the intermediate results. [CO15]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₀</td>
<td>b₀</td>
<td></td>
</tr>
<tr>
<td>a₀</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a₀</td>
<td>bₘ</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>b₀</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>b₀</td>
<td></td>
</tr>
<tr>
<td>aₘ</td>
<td>b₀</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₀</td>
<td>c₀</td>
<td></td>
</tr>
<tr>
<td>a₀</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a₀</td>
<td>cₘ</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>c₀</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>c₀</td>
<td></td>
</tr>
<tr>
<td>aₘ</td>
<td>c₀</td>
<td></td>
</tr>
</tbody>
</table>

$$R_1 \cup A \times B \cup a_0 \cdots b_m \cup b_0$$

$$R_2 \cup A \times C \cup a_0 \cdots c_m \cup c_0$$
One-join-at-a-time using Factorized Representations

We can attain worst-case optimality for join-at-a-time processing by factorizing the intermediate results. [CO15]

<table>
<thead>
<tr>
<th></th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( B )</td>
<td>( A )</td>
<td>( C )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_0 )</td>
<td>( a_0 )</td>
<td>( c_0 )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( \ldots )</td>
<td>( a_0 )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( a_0 )</td>
<td>( b_m )</td>
<td>( a_0 )</td>
<td>( c_m )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_0 )</td>
<td>( a_1 )</td>
<td>( c_0 )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( b_0 )</td>
<td>( \ldots )</td>
<td>( c_0 )</td>
</tr>
<tr>
<td>( a_m )</td>
<td>( b_0 )</td>
<td>( a_m )</td>
<td>( c_0 )</td>
</tr>
</tbody>
</table>
One-join-at-a-time using Factorized Representations

Relation $R_1(A, B)$.  

Relation $R_2(A, C)$.  

Relation $R_3(B, C)$.  

After joining on $A$.  

After joining on $B$.  

After joining on $C$.  

$m + 1$ values  

$m + 2$ values  

$m + 3$ values
One-join-at-a-time using Factorized Representations

Relation $R_1(A, B)$.

Relation $R_2(A, C)$.

Relation $R_3(B, C)$.

After joining on $A$

$m + 1$ values
One-join-at-a-time using Factorized Representations

Relation $R_1(A, B)$.

Relation $R_2(A, C)$.

Relation $R_3(B, C)$.

After joining on $A$

$m + 1$ values

After joining on $B$

$3m + 2$ values
One-join-at-a-time using Factorized Representations

Relation $R_1(A, B)$.

Relation $R_2(A, C)$.

Relation $R_3(B, C)$.

After joining on $A$:
$m + 1$ values

After joining on $B$:
$3m + 2$ values

After joining on $C$:
$6m + 3$ values
Open Problems for In-Database Optimization
Beyond the Current Setting

The presented framework works for semi-ring training expressions

- Gradient computation and point evaluation are FAQ queries in the sum-product semi-ring
- Necessary for factorization
- Beyond BGD: quasi-Newton, stochastic/coordinate GD

Further loss functions beyond square loss?

Further regularization functions beyond $\ell_2$-norm?

Further ML problems typically used by LogicBlox

- Decision/boosted trees, clustering, deep neural nets

Is the framework also applicable to these problems?
Effect of Functional Dependencies on Convergence

Rule: Eliminate features that are functionally determined.

It reduces

- Computational complexity
- Model complexity
- Generalization error, i.e., how accurately an algorithm is able to predict outcome values for previously unseen data.

Can it lead to faster BGD convergence?

How to approach non-linear dependencies?

- Statistical effectiveness of above rule is difficult to gauge
- Keep redundant variables if they help construct simpler regression/classification functions

FD-aware optimizations beyond the $\ell_2$-norm in the penalty term?
Foundations for In-Database Linear Algebra

- Current research topic in the database systems community
- Missing: Expressiveness and complexity analysis

For FD-aware model optimization, we need in-database support for

- Tensor, Kronecker, Khatri-Rao products
- Special cases of matrix inversion with rank-1 update, e.g., Sherman-Morrison-Woodbury

\[ \mathbf{M}_S = \mathbf{I} + \sum_{s \in S} \mathbf{R}_s^\top \mathbf{R}_s \]

as needed in the penalty term of the rewritten loss function:

\[ \langle (\mathbf{I}_{\text{city}} + \mathbf{R}^\top \mathbf{R})^{-1} \gamma_{\text{city}}, \gamma_{\text{city}} \rangle \]

- Cholesky decomposition of matrices

Such linear algebra expressions can be phrased as queries in our framework.
Multi-Output Functional Aggregate Queries

- Common ML tasks can be expressed as programs in languages with functional aggregates and fixpoint.
- These programs have many inter-related aggregates, e.g., $\sigma_{ij}$ and $c$.

Needed: Systematic investigation of algorithms and complexity for computing large sets of inter-related aggregates.

- Expected: Lower complexity than computing them independently.
Thank you!
References on Join Computation


References on Join Computation

Veldhuizen. In ICDT 2014.

OZ15 Size Bounds for Factorised Representations of Query Results.
http://dl.acm.org/citation.cfm?doid=2656335

CO15 Worst-Case Optimal Join At A Time.

ANS17 What do Shannon-type inequalities, submodular width, and disjunctive Datalog have to do with one another?
https://arxiv.org/abs/1612.02503

KO17 Covers of Query Results.
References on Aggregate Computation

**BKOZ13** Aggregation and Ordering in Factorised Databases.
https://arxiv.org/abs/1307.0441

**ANR16** FAQ: Questions Asked Frequently.
https://arxiv.org/abs/1504.04044
References on In-Database Analytics

SOC16  Learning Linear Regression Models over Factorized Joins.
       http://dl.acm.org/citation.cfm?doid=2882903.2882939

A17   Research Directions for Principles of Data Management (Dagstuhl
       Perspectives Workshop 16151).

ANNOS17  In-Database Learning with Sparse Tensors.
        https://arxiv.org/abs/1703.04780

KBY17  Data Management in Machine Learning: Challenges, Techniques, and
       Systems.
       Kumar, Boehm, Yang. In SIGMOD 2017, Tutorial.
       https://www.youtube.com/watch?v=U8J0Dd_Z5wo

NO17  Incremental Maintenance of Regression Models over Joins.
       https://arxiv.org/abs/1703.07484

PRWZ17  Data Management Challenges in Production Machine Learning.
        http://dl.acm.org/citation.cfm?doid=3035918.3054782