From Joins to Aggregates

and Optimization Problems

Dan Olteanu (Oxford & Turing)

`https://fdbresearch.github.io`

Data Science Class
Alan Turing Institute
Jan 29, 2018

Acknowledgements

Some work reported in this tutorial has been done in the context of the FDB project, LogicBlox, and RelationalAI by

- Zavodný, Schleich, Kara, Ciucanu, and myself (Oxford)
- Abo Khamis and Ngo (RelationalAI), Nguyen (U. Michigan)

Some of the following slides are derived from presentations by

- Abo Khamis (in-db analytics diagrams)
- Kara (covers and various graphics)
- Ngo (FAQ)
- Schleich (performance and quizzes)

Lastly, Kara and Schleich proofread the slides.

I would like to thank them for their support and for sharing their work!

Goal of This Tutorial

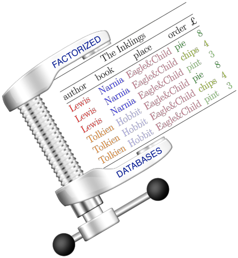
Introduction to a principled, relatively new approach to in-database computation

It starts where mainstream introductory/advanced courses on databases finish.

- Joins
  - Worst-case optimal join algorithms
  - Listing vs. 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
  - New algorithms with low computational complexity
- Optimizations
  - In-database learning of regression and classification models

Quizzes: Test your understanding after class

Outline



Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Join Queries

Basic building blocks in query languages. 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 listing and factorized data representations.

Plan for Part 1 on Joins

- Introduction to join queries via examples
- Size bounds for results of join queries
  - Standard (exhaustive) listing representation
  - Factorized (succinct) representations
- Worst-case optimal join algorithms
  - LFTJ (LeapFrog TrieJoin) used by LogicBlox for listing representation
  - FDB (Factorized Databases) for factorized representations

Introduction to Join Queries

Join Example: Itemized Customer Orders

Orders (O for short)

customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4

Consider the natural join of the above relations:

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

Join Example: Listing the Triangles in the Database

R_1

A	B
a_0	b_0
a_0	$\dots$
a_0	b_m
a_1	b_0
$\dots$	b_0
a_m	b_0

R_2

A	C
a_0	c_0
a_0	$\dots$
a_0	c_m
a_1	c_0
$\dots$	c_0
a_m	c_0

R_3

B	C
b_0	c_0
b_0	$\dots$
b_0	c_m
b_1	c_0
$\dots$	c_0
b_m	c_0

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

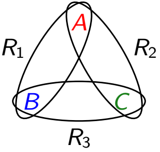
A	B	C
a_0	b_0	c_0
a_0	b_0	$\dots$
a_0	b_0	c_m
a_0	b_1	c_0
a_0	$\dots$	c_0
a_0	b_m	c_0
a_1	b_0	c_0
$\dots$	b_0	c_0
a_1	b_0	c_0

Join Hypergraphs

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

- Each variable in Q corresponds to a node in $\mathcal{V}$
- Each relation symbol in Q corresponds to a (hyper)edge in $\mathcal{E}$

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



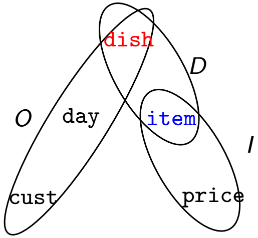
- $\mathcal{V} = \{A, B, C\}$
- $\mathcal{E} = \{\{A, B\}, \{A, C\}, \{B, C\}\}$

Join Hypergraphs

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

- Each variable in Q corresponds to a node in $\mathcal{V}$
- Each relation symbol in Q corresponds to a (hyper)edge in $\mathcal{E}$

Example: Order query $O(\texttt{cust}, \texttt{day}, \texttt{dish}), D(\texttt{dish}, \texttt{item}), I(\texttt{item}, \texttt{price})$



- $\mathcal{V} = \{\texttt{cust}, \texttt{day}, \texttt{dish}, \texttt{item}, \texttt{price}\}$
- $\mathcal{E} = \{\{\texttt{cust}, \texttt{day}, \texttt{dish}\}, \{\texttt{dish}, \texttt{item}\}, \{\texttt{item}, \texttt{price}\}\}$

Hypertree Decompositions

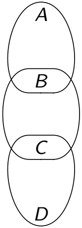
Definition[GLS99]: A (hypertree) decomposition $\mathcal{T}$ of the hypergraph $(\mathcal{V}, \mathcal{E})$ of a query Q is a pair (T, χ), where

- T is a tree
- χ is a function mapping each node in T to a subset of $\mathcal{V}$ called *bag*.

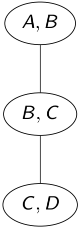
Properties of a decomposition $\mathcal{T}$:

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

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



A hypertree decomposition



Hypertree Decompositions

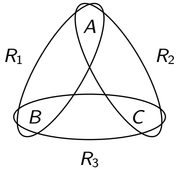
Definition[GLS99]: A (hypertree) decomposition $\mathcal{T}$ of the hypergraph $(\mathcal{V}, \mathcal{E})$ of a query Q is a pair (T, χ), where

- T is a tree
- χ is a function mapping each node in T to a subset of $\mathcal{V}$ called *bag*.

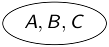
Properties of the decomposition $\mathcal{T}$:

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

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



A hypertree decomposition



Variable Orders

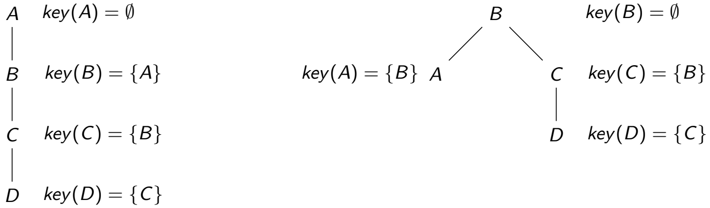
Definition[OZ15]: A *variable order* Δ for a query Q is a pair (F, key), where

- 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 Δ 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 $R_1(A, B), R_2(B, C), R_3(C, D)$:



Variable Orders

Definition[OZ15]: A *variable order* Δ 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 Δ 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 triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$:

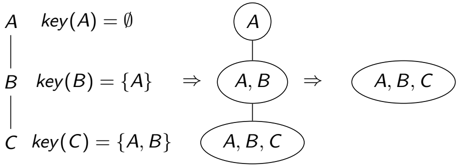
$$\begin{array}{ll} A & key(A) = \emptyset \\ | & \\ B & key(B) = \{A\} \\ | & \\ C & key(C) = \{A, B\} \end{array} \qquad \begin{array}{ll} B & key(B) = \emptyset \\ | & \\ A & key(A) = \{B\} \\ | & \\ C & key(C) = \{A, B\} \end{array} \qquad \begin{array}{ll} C & key(C) = \emptyset \\ | & \\ B & key(B) = \{C\} \\ | & \\ A & key(A) = \{B, C\} \end{array}$$

Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From variable order Δ to hypertree decomposition $\mathcal{T}$: [OZ15]

- For each node A in Δ, 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 $R_1(A, B), R_2(A, C), R_3(B, C)$

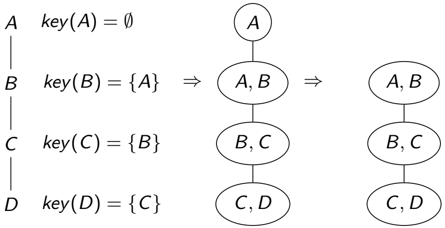


Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From variable order Δ to hypertree decomposition $\mathcal{T}$: [OZ15]

- For each node A in Δ, 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 $R_1(A, B), R_2(B, C), R_3(C, D)$

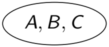


Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ 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 Δ
- Update *key* for each variable at each step.

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



Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ 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 Δ
- Update *key* for each variable at each step.

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

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

Step 1:

A is removed from $\mathcal{T}$

and inserted into Δ

$A, B, C \Rightarrow$

Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

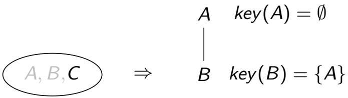
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

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 Δ



Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ 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 Δ
- 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 Δ

A, B, C $\Rightarrow$

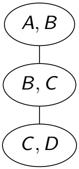
$$
\begin{array}{ll}
A & key(A) = \emptyset \\
| & \\
B & key(B) = \{A\} \\
| & \\
C & key(C) = \{A, B\}
\end{array}
$$

Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ 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 Δ
- Update *key* for each variable at each step.

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



Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

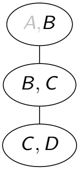
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

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 Δ



$\Rightarrow$

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

Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

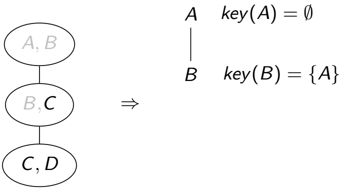
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

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 Δ



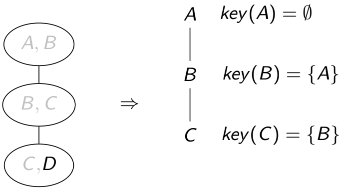
Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ 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 Δ
- 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 Δ



Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

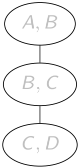
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

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 Δ



$\Rightarrow$

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?

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

- Assumption: All relations have size N.

- 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?

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

- Assumption: All relations have size N.

- 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 $R_1 = [N] \times \{1\}, R_2 = \{1\} \times [N], R_3 = [N] \times \{1\}$.

- Conclusion: Size of the query result is $\Theta(N^2)$ for some inputs

How Can We Bound the Size of the Join Result?

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

- Assumption: All relations have size N.

- 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 $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supset \{(1, 1)\}$

How Can We Bound the Size of the Join Result?

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

- Assumption: All relations have size N.
- 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 $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supset \{(1, 1)\}$
- Conclusion: Size gap between the N^2 upper bound and the N lower bound

Question: Can we close this gap and give tight size bounds?

Edge Covers and Independent Sets

We can generalize the previous examples as follows:

For the size upper bound:

- Cover all nodes (variables) by k edges (relations) $\Rightarrow$ size $\leq N^k$.
- This is an edge cover of the query hypergraph!

For the size lower bound:

- m independent nodes $\Rightarrow$ construct database such that size $\geq N^m$.
- This is an independent set of the query hypergraph!

$$\mathsf{max}_m = |\mathrm{IndependentSet}(Q)| \leq |\mathrm{EdgeCover}(Q)| = \mathsf{min}_k$$

$$\boxed{\mathsf{max}_m \text{ and } \mathsf{min}_k \text{ do not necessarily meet!}}$$

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 $m \Rightarrow \exists$ database with 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 $\rho^*(Q)$!

For query Q and database of size N, the query result has size $O(N^{\rho^*(Q)})$.

The Fractional Edge Cover Number $\rho^*(Q)$

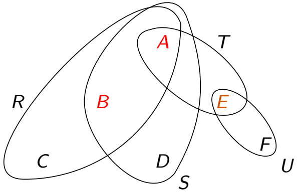
For a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$,
$\rho^*(Q)$ is the cost of an optimal solution to the linear program:

$$
\begin{array}{rll}
\text{minimize} & \sum_{i \in [n]} x_{R_i} & \\
\text{subject to} & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 & \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 & \forall i \in [n].
\end{array}
$$

- x_{R_i} is the weight of edge (relation) R_i in the hypergraph of Q
- Each node (variable) has to be covered by edges with sum of weights ≥ 1
- In the integer program variant for the edge cover, $x_{R_i} \in \{0, 1\}$

Example of Fractional Edge Cover Computation (1)

Consider the join query Q: $R(\mathtt{A}, \mathtt{B}, C), S(\mathbf{A}, \mathbf{B}, D), T(\mathbf{A}, \mathbf{E}), U(\mathbf{E}, F)$.



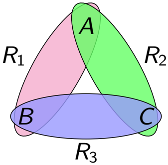
- The three edges R, S, U to cover all nodes.
  $\mathrm{FractionalEdgeCover}(Q) \leq 3$
- Each node C, D, and F must be covered by a distinct edge.
  $\mathrm{FractionalIndependentSet}(Q) \geq 3$

$\Rightarrow \rho^*(Q) = 3$

$\Rightarrow$ Size $\leq N^3$ and for some inputs is $\Theta(N^3)$.

Example of Fractional Edge Cover Computation (2)

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



$$
\begin{array}{lccccccc}
\multicolumn{8}{l}{\textit{minimize } x_{R_1} + x_{R_2} + x_{R_3}} \\
\multicolumn{8}{l}{\textit{subject to}} \\
A: & x_{R_1} & + & x_{R_2} & & & \geq 1 \\
B: & x_{R_1} & & & + & x_{R_3} & \geq 1 \\
C: & & & x_{R_2} & + & x_{R_3} & \geq 1 \\
 & x_{R_1} \geq 0 & & x_{R_2} \geq 0 & & x_{R_3} \geq 0 &
\end{array}
$$

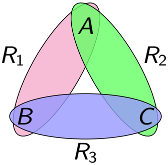
Our previous size upper bound was N^2:

- 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?

Example of Fractional Edge Cover Computation (2)

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



$$
\begin{array}{llllllll}
\multicolumn{8}{l}{\textit{minimize } x_{R_1} + x_{R_2} + x_{R_3}} \\
\multicolumn{8}{l}{\textit{subject to}} \\
A: & x_{R_1} & + & x_{R_2} & & & \geq 1 \\
B: & x_{R_1} & & & + & x_{R_3} & \geq 1 \\
C: & & & x_{R_2} & + & x_{R_3} & \geq 1 \\
 & x_{R_1} \geq 0 & & x_{R_2} \geq 0 & & x_{R_3} \geq 0 &
\end{array}
$$

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?

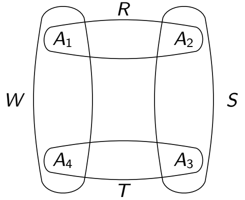
We can do better: $x_{R_1} = x_{R_2} = x_{R_3} = 1/2$. Then, $\rho^* = 3/2$.

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:



$$
\begin{array}{lccccccccl}
\multicolumn{10}{l}{\textit{minimize } x_R + x_S + x_T + x_W} \\
\multicolumn{10}{l}{\textit{subject to}} \\
A_1: & x_R & & & & & + & x_W & \geq 1 \\
A_2: & x_R & + & x_S & & & & & \geq 1 \\
A_3: & & & x_S & + & x_T & & & \geq 1 \\
A_4: & & & & & x_T & + & x_W & \geq 1 \\
 & x_R \geq 0 & & x_S \geq 0 & & x_T \geq 0 & & x_W \geq 0 &
\end{array}
$$

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 $S = W = \{1\} \times [N]$.

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Recall the linear program for computing the fractional edge cover number $\rho^*(Q)$ of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{array}{lll}
\text{minimize} & \sum_{i \in [n]} x_{R_i} & \\
\text{subject to} & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 & \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\
 & x_{R_i} \geq 0 & \forall i \in [n].
\end{array}
$$

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Add relation sizes into the linear program that computes the result size of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{aligned}
\text{minimize} \quad & N^{\sum_{i \in [n]} x_{R_i}} \\
\text{subject to} \quad & \textstyle\sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 \qquad\qquad\qquad\qquad \forall i \in [n].
\end{aligned}
$$

Assumption: All relations have the same size N.

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Add relation sizes into the linear program that computes the result size of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{aligned}
\text{minimize} \quad & \textstyle\prod_{i\in[n]} N^{x_i} \\
\text{subject to} \quad & \textstyle\sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j\in[n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 \qquad\qquad\qquad\qquad \forall i \in [n].
\end{aligned}
$$

Assumption: All relations have the same size N.

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Add relation sizes into the linear program that computes the result size of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{array}{rll}
\text{minimize} & \prod_{i \in [n]} N_i^{x_i} & \\
\text{subject to} & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 & \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 & \forall i \in [n].
\end{array}
$$

Assumption: Relation R_i has size N_i, $\forall i \in [n]$.

Size Bounds for Factorized Representations of Join Results

Recall the Itemized Customer Orders Example

Orders (O for short)

customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4

Consider the natural join of the above relations:

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

Factor Out Common Data Blocks

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

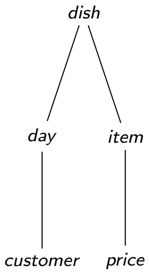
The listing representation of the above query result is:

$$
\begin{array}{lllllllllll}
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{burger} \rangle & \times & \langle \text{patty} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{burger} \rangle & \times & \langle \text{onion} \rangle & \times & \langle 2 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{burger} \rangle & \times & \langle \text{bun} \rangle & \times & \langle 2 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{burger} \rangle & \times & \langle \text{patty} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{burger} \rangle & \times & \langle \text{onion} \rangle & \times & \langle 2 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{burger} \rangle & \times & \langle \text{bun} \rangle & \times & \langle 2 \rangle & \cup \ldots
\end{array}
$$

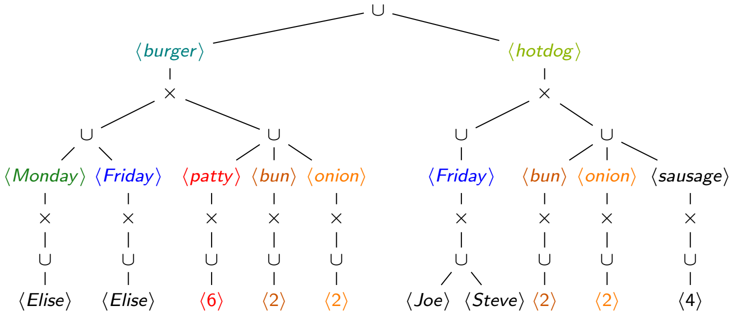
It uses relational product ($\times$), union ($\cup$), and data (singleton relations).

- The attribute names are not shown to avoid clutter.

This is How A Factorized Join Looks Like!



Var order

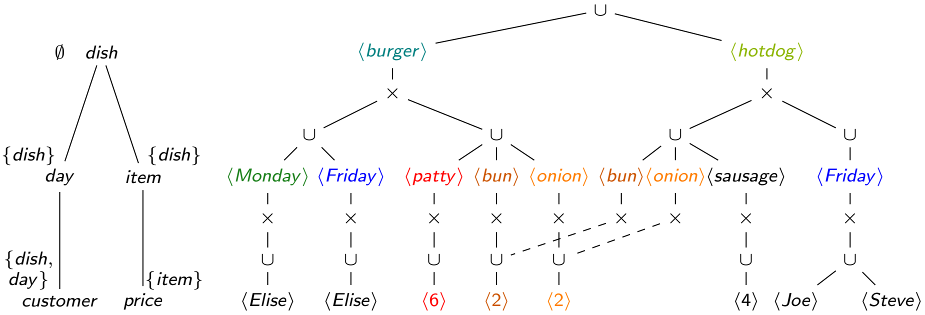


Factorized representation of the join result

There are several *algebraically equivalent* factorized representations defined:

- by distributivity of product over union and their commutativity;
- as groundings of variable orders.

.. Now with Further Compression using Caching

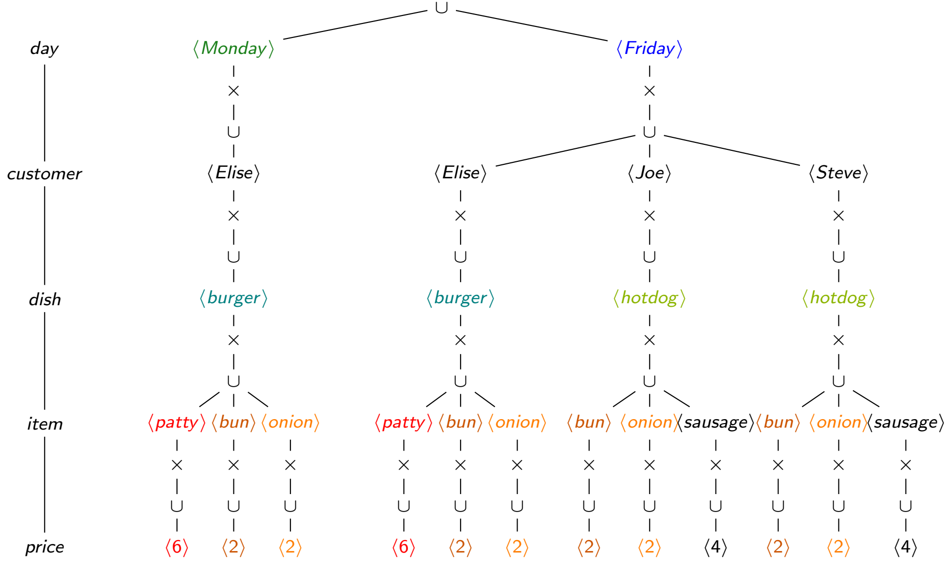


Observation:

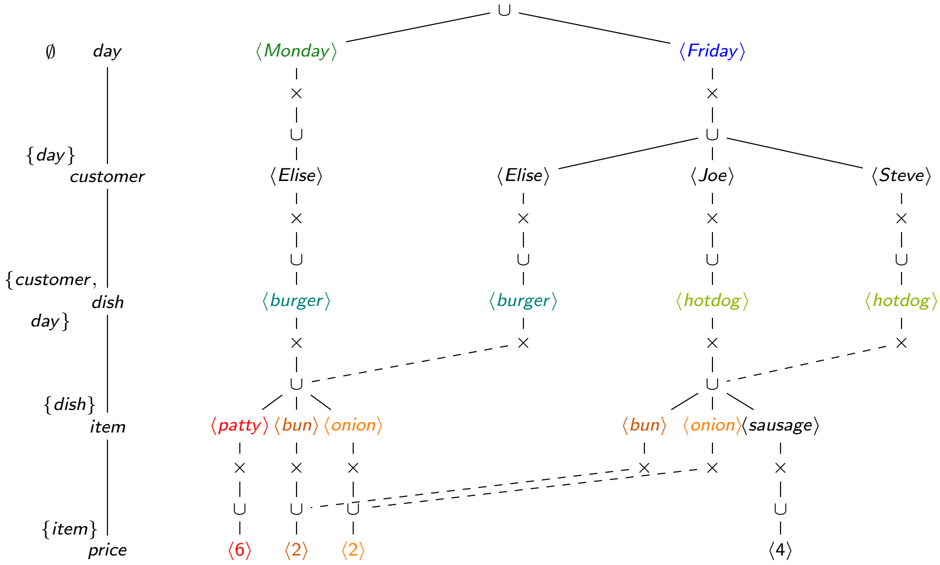
- `price` is under `item`, which is under `dish`, but only *depends* on `item`,
- .. so the same `price` appears under an `item` *regardless* of the `dish`.

Idea: *Cache* `price` for a specific `item` and avoid repetition!

Same Data, Different Factorization



.. and Further Compressed using Caching



Which factorization should we choose?

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

Example:

$$
\begin{aligned}
F_1 =& \big(\langle 1\rangle \cup \cdots \cup \langle n\rangle\big) \times \big(\langle 1\rangle \cup \cdots \cup \langle m\rangle\big) \\
F_2 =& \langle 1\rangle \times \langle 1\rangle \cup \cdots \cup \langle 1\rangle \times \langle m\rangle \\
& \qquad\qquad \cup \cdots \cup \\
& \quad \langle n\rangle \times \langle 1\rangle \cup \cdots \cup \langle n\rangle \times \langle m\rangle.
\end{aligned}
$$

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

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

$$\boxed{1 \leq fhtw(Q) \underbrace{\leq}_{\text{up to } \log|Q|} s(Q) \underbrace{\leq}_{\text{up to } |Q|} \rho^*(Q) \leq |Q|}$$

- $|Q|$ 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 [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]

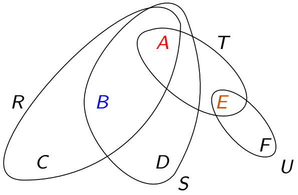
- a factorization *with caching* of size $O(N^{fhtw(Q)})$. [OZ15]

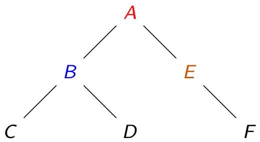
These size bounds are asymptotically tight!

- **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 $\Omega(N^{s(Q)})$ and $\Omega(N^{fhtw(Q)})$ respectively.

Example: The Factorization Width s





The structure of the factorization over the above variable order Δ:

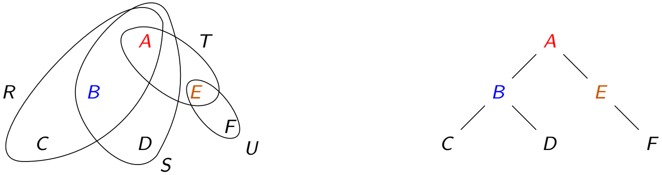
$$\bigcup_{a\in\mathbf{A}}\Big(\langle a\rangle \times \bigcup_{b\in\mathbf{B}}\Big(\langle b\rangle \times \big(\bigcup_{c\in C}\langle c\rangle\big) \times \big(\bigcup_{d\in D}\langle d\rangle\big)\Big) \times \bigcup_{e\in\mathbf{E}}\Big(\langle e\rangle \times \big(\bigcup_{f\in F}\langle f\rangle\big)\Big)\Big)$$

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

- 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 Δ is the largest ρ^* over subqueries defined by root-to-leaf paths in Δ
- $s(Q)$ is the minimum factorization width over all variable orders of Q

In our example:

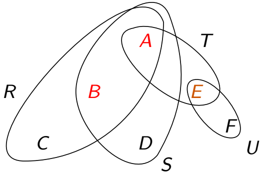
- Path A–E–F has fractional edge cover number 2.
  $\Rightarrow$ The number of F-values is $\leq N^2$, but can be $\sim N^2$.
- All other root-to-leaf paths have fractional edge cover number 1.
  $\Rightarrow$ The number of other values is $\leq N$.

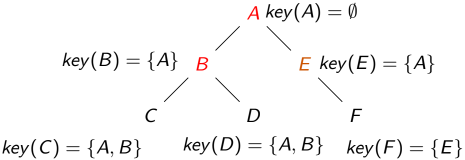
$s(Q) = 2$ $\Rightarrow$ Factorization size is $O(N^2)$

Recall that $\rho^*(Q) = 3$ $\Rightarrow$ Listing representation size is $O(N^3)$

Example: The Fractional Hypertree Width *fhtw*

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





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

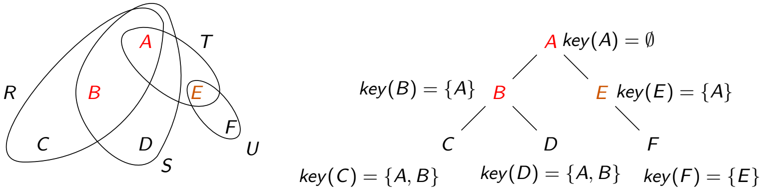
Observation:

- Variable F only depends on **E** and not on **A**: $key(F) = \{\mathbf{E}\}$
- 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.

  $\Rightarrow$ Define $U_e = \bigcup_{(e,f) \in U} \langle f \rangle$ for each value $\langle e \rangle$ and use U_e 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 caching would be:

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

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

In our example: $fhtw(Q) = 1 < s(Q) = 2 < \rho^*(Q) = 3$.

Alternative Characterizations of *fhtw*

The fractional hypertree width *fhtw* has been originally defined for hypertree decompositions. [M10]

- Given a join query Q.
- Let $\mathbf{T}$ be the set of hypertree decompositions of the hypergraph of Q.

$$\boxed{\mathit{fhtw}(Q) = \min_{(T,\chi)\in\mathbf{T}} \max_{n\in T} \rho^*(Q_{\chi(n)})}$$

Alternative Characterizations of *fhtw*

The fractional hypertree width *fhtw* has been originally defined for hypertree decompositions. [M10]

- Given a join query Q.
- Let $\mathbf{T}$ be the set of hypertree decompositions of the hypergraph of Q.

$$\boxed{\mathit{fhtw}(Q) = \min_{(T,\chi)\in\mathbf{T}} \max_{n\in T} \rho^*(Q_{\chi(n)})}$$

Alternative characterization of the fractional hypertree width *fhtw* using the mapping between hypertree decompositions and variable orders [OZ15]

- Given a join query Q.
- Let $\mathbf{VO}$ be the set of variable orders of Q.

$$\boxed{\mathit{fhtw}(Q) = \min_{(F,\mathit{key})\in\mathbf{VO}} \max_{v\in F} \rho^*(Q_{\mathit{key}(v)\cup\{v\}})}$$

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 $\widetilde{O}(N^{fhtw(Q)})$
- 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

Orders (O for short)

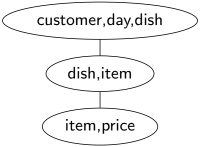
customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4



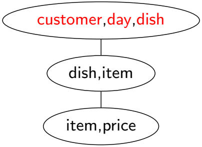
O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

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

The Hypergraph of the Query Result

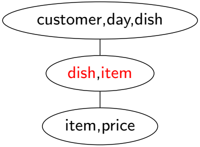
burger patty

Elise Monday burger

burger onion

Elise Friday burger

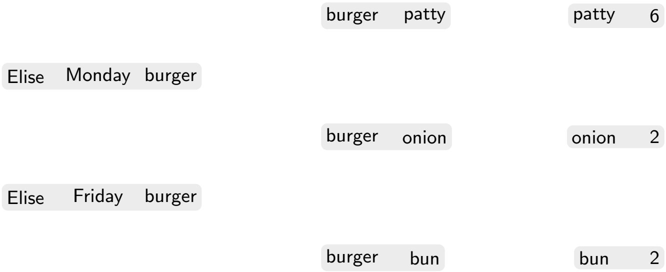
burger bun

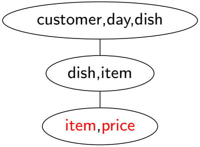


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

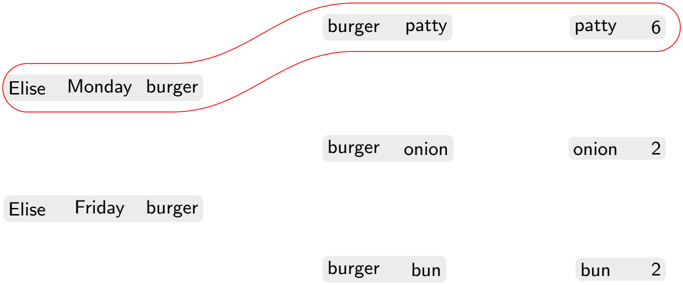


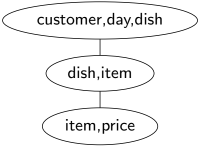


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

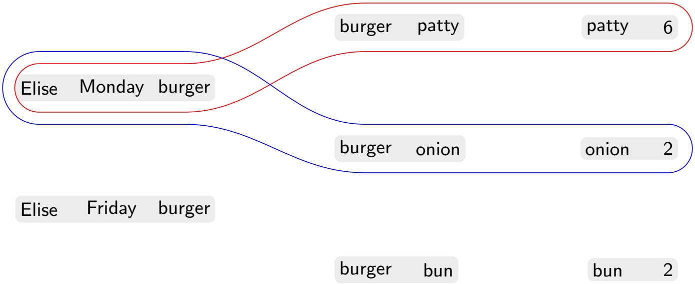


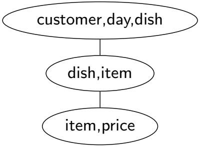


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

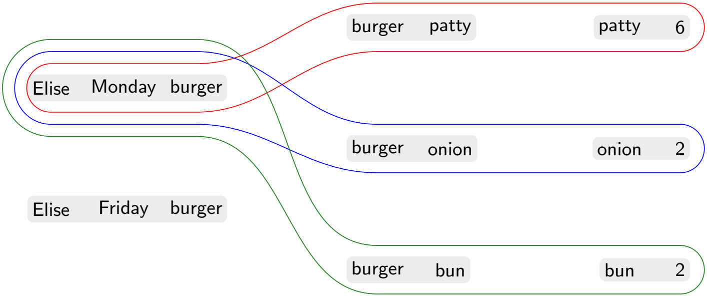


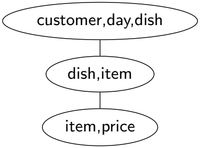


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

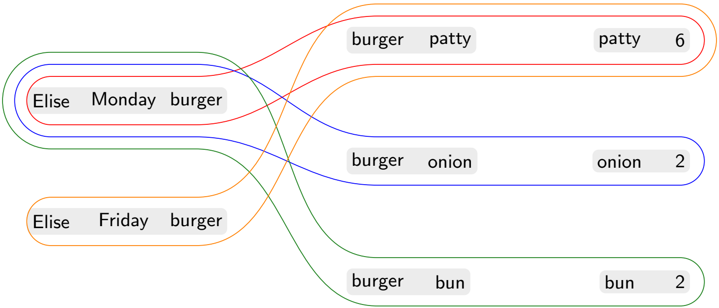


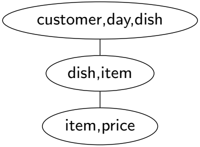


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

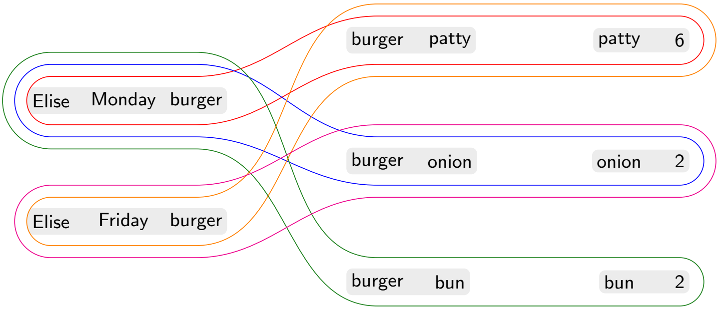


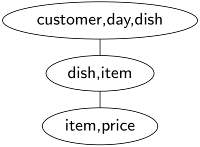


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

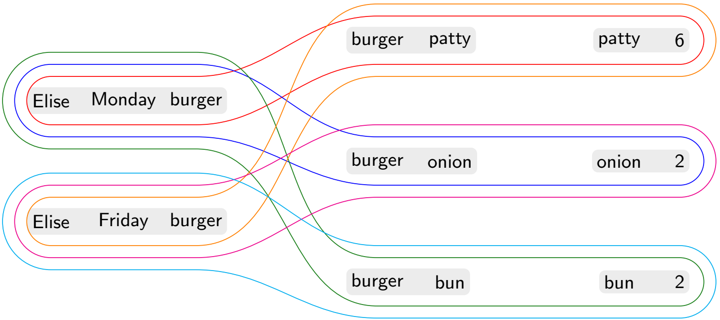


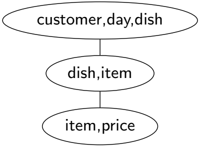


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

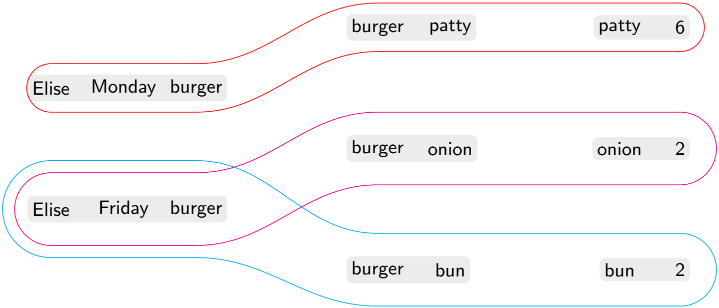


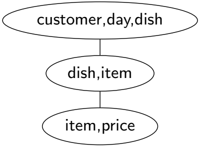


O(customer, day, dish), D(dish, item), I(item, price)

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





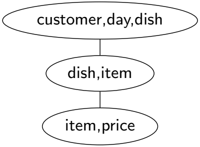
O(customer, day, dish), D(dish, item), I(item, price)

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

O(customer, day, dish), D(dish, item), I(item, price)

customer	day	dish	item	price
Elise	Monday	burger	patty	6
Elise	Friday	burger	onion	2
Elise	Friday	burger	bun	2

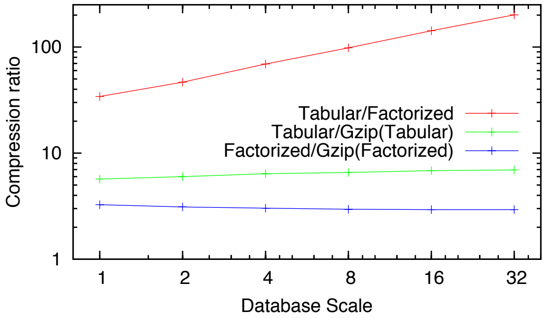


O(customer, day, dish), D(dish, item), I(item, price)

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

Compression by Factorization in Practice

Compression Contest: Factorized vs. Zipped Relations



Result of query $\text{Orders} \bowtie \text{Dish} \bowtie \text{Items}$ [BKOZ13]

- 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.

Factorization Gains in Practice (1/3)

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

- Relations: UserArtists (93K), UserFriends (25K), TaggedArtists (186K).
- 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

How Fast Can We Compute Join Results?

Given a join query Q, for any database of size N, the join result can be computed in time

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

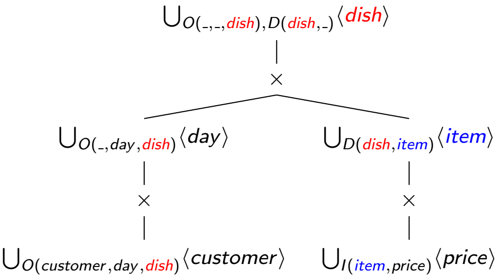
These upper bounds essentially follow the succinctness gap. They are:

- 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

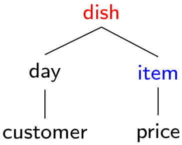
Example: Computing the Factorized Join Result with FDB

Our join: O(customer, day, dish), D(dish, item), I(item, price)

can be grounded to a factorized representation as follows:



This computation follows the variable order given below:



Example: Computing the Factorized Join Result with FDB

$$\bigcup_{O(\_,\_,dish),D(dish,\_)} \langle dish \rangle$$

$$\times$$

$$\bigcup_{O(\_,day,dish)} \langle day \rangle \qquad\qquad \bigcup_{D(dish,item)} \langle item \rangle$$

$$\times \qquad\qquad\qquad\qquad \times$$

$$\bigcup_{O(customer,day,dish)} \langle customer \rangle \qquad \bigcup_{I(item,price)} \langle price \rangle$$

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

LeapFrog TrieJoin Algorithm

- Much acclaimed worst-case optimal join algorithm used by LogicBlox [V14]
- Computes a listing representation of the join result

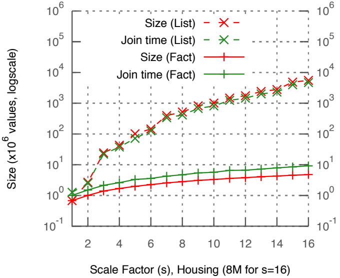
  $\Rightarrow$ It does not exploit factorization
- Glorified multi-way sort-merge join with an efficient list intersection
- Several generalizations, e.g., PANDA [NRR13,ANS17]

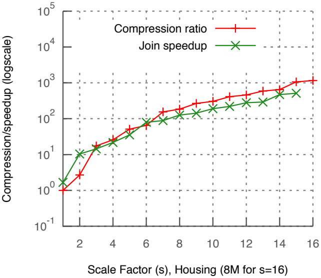
LeapFrog TrieJoin is a special case of FDB, where

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

Experiment: Factorized vs. Listing Computation

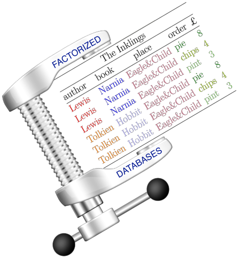
		Retailer (3B)	LastFM (5.8M)
Join	Factorization	169M	316K
Size	Listing	3.6B	591M
(values)	Compression	21.4×	1870.7×
Join	FDB	30	10
Time	PostgreSQL	217	61
(sec)	Speedup	7×	6.1×





Both FDB and PostgreSQL list the records in the results of the join queries.

Outline



Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Aggregates

Important operators in database query languages and essential for applications.

Natural generalization of aggregates over joins can express a host of problems across Computer Science. [ANR16]

We highlight recent work on aggregate computation with lowest known computational complexity. This extends the work from Part 1. [BKOZ13,ANR16]

Part 3 later discusses an extension of this work to state-of-the-art machine learning inside the database. [SOC16,ANNOS17]

Plan for Part 2 on Aggregates

- Computation of aggregates over factorized joins using the FDB algorithm [BKOZ13]

- Factorized computation of aggregates using optimized relational queries. [SOC16,OS16]

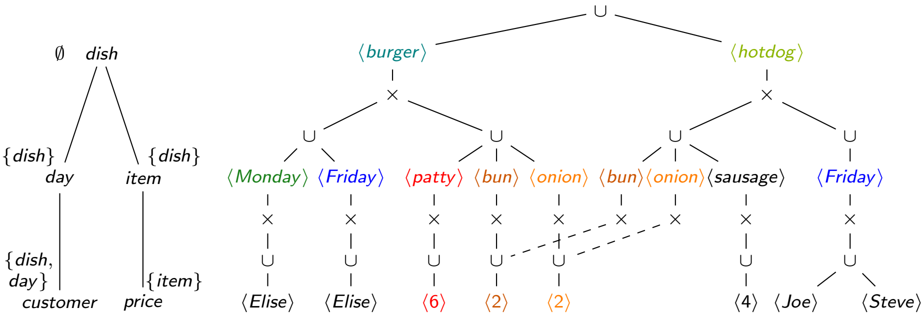
- Functional Aggregate Queries (FAQs) [ANR16]

  - Generalize aggregate-join queries to many semirings, e.g., sum-product, max-product, Boolean
  - FAQ computation is factorized and has the computational complexity of aggregates over factorized joins

- FAQ computation using the InsideOut algorithm [ANR16]

Examples: Aggregates over Factorized Joins

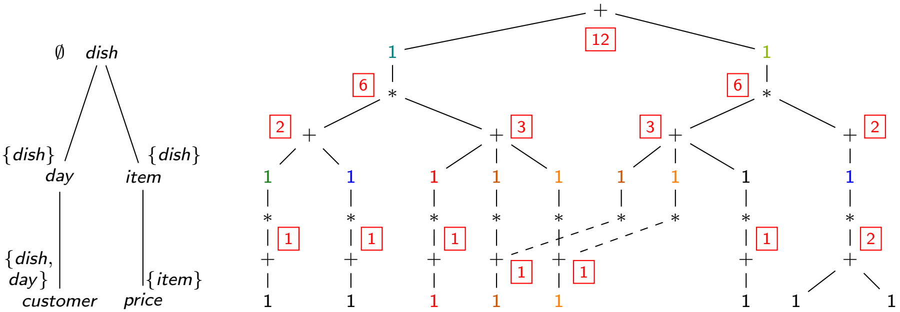
Example 1: COUNT Aggregate over Factorized Join



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

- `COUNT(*)`:
  - values $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto *$.

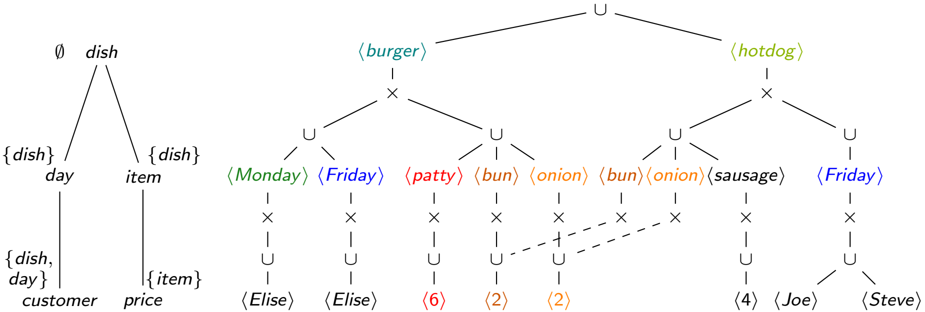
Example 1: COUNT Aggregate over Factorized Join



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

- COUNT(*):
  - values $\mapsto$ 1,
  - $\cup \mapsto +$,
  - $\times \mapsto *$.

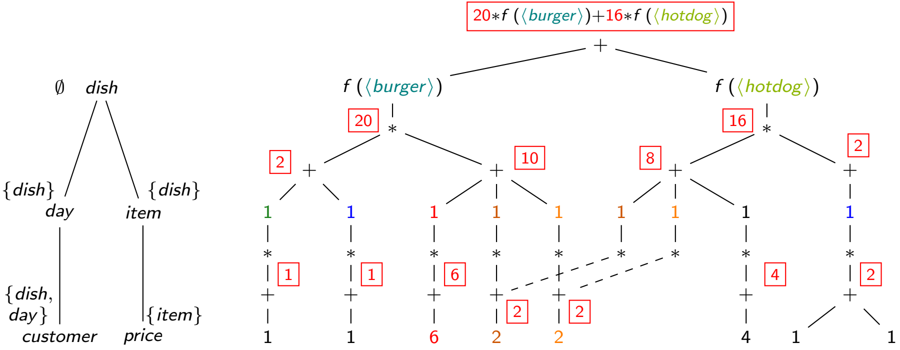
Example 2: SumProd Aggregate over Factorized Join



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 or indicator vectors.
  - All values except for `dish` & `price` $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto *$.

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` $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto *$.

Computing Aggregates over Factorized Joins using FDB

Given an aggregate-join query Q [BKOZ13]

- Construct a variable order Δ where **the group-by (free) variables are above the other (bound) variables of Q**
  - **A new width w measure that is at least *fhtw***
- Compute the factorized join over Δ
  - **The complexity now depends on the width w**
- Finally compute the aggregates in one pass over the factorized join.

Computing Aggregates over Factorized Joins using FDB

Given an aggregate-join query Q [BKOZ13]

- Construct a variable order Δ where **the group-by (free) variables are above the other (bound) variables of Q**
  - **A new width w measure that is at least $fhtw$**
- Compute the factorized join over Δ
  - **The complexity now depends on the width w**
- Finally compute the aggregates in one pass over the factorized join.

Is it necessary to first compute the factorized join?

Computing Aggregates over Factorized Joins using FDB

Given an aggregate-join query Q [BKOZ13]

- Construct a variable order Δ where **the group-by (free) variables are above the other (bound) variables of** Q
  - **A new width** w **measure that is at least** *fhtw*
- Compute the factorized join over Δ
  - **The complexity now depends on the width** w
- Finally compute the aggregates in one pass over the factorized join.

Is it necessary to first compute the factorized join?

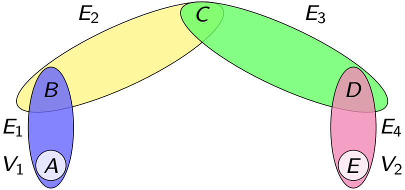
Aggregates can be computed without materializing the factorized join

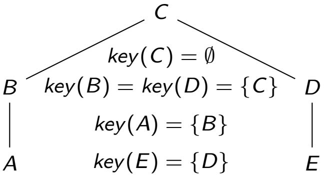
- The factorized join becomes the *trace* of the aggregate computation
- This is the *factorized computation* of the query Q.

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):

$$V_1(A), E_1(A, B), E_2(B, C), E_3(C, D), E_4(D, E), V_2(E)$$

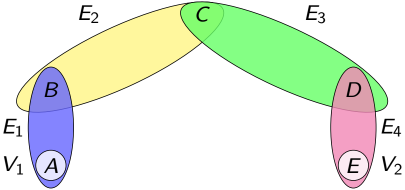


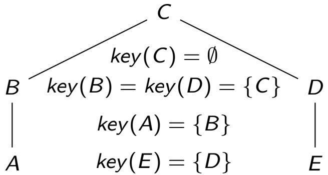


Example: Factorized Aggregate Computation

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

- $\rho^*(Q_4) = 3 \Rightarrow$ listing representation has size $O(|E|^3)$.
- $fhtw(Q_4) = 1 \Rightarrow$ factorization with caching has size $O(|E|)$.

Example: Factorized Aggregate Computation

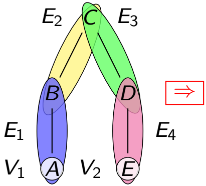
We would like to compute COUNT(Q_4):

- in $O(|E|)$ time (no free variables, so use best variable order for Q_4)
- using optimized queries that are derived from the variable order of Q_4
- without materializing the factorized result of the path query

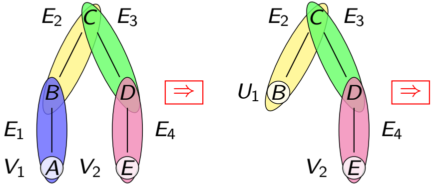
Convention:

- View the relations as functions mapping tuples to numbers.
- The functions for input relations map their tuples to 1.

Example: Factorized Computation of COUNT(Q_4)

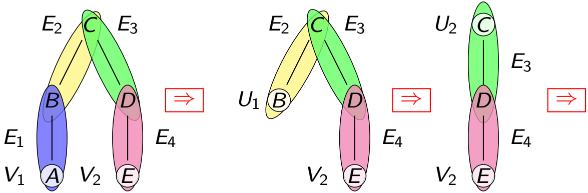


Example: Factorized Computation of COUNT(Q_4)



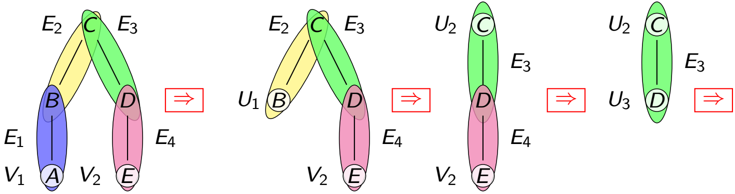
$$U_1(b) = \sum_{a \in \mathrm{Dom}(A)} V_1(a) \cdot E_1(b, a)$$

Example: Factorized Computation of COUNT(Q_4)



$$U_1(b) = \sum_{a \in \mathrm{Dom}(A)} V_1(a) \cdot E_1(b, a) \qquad U_2(c) = \sum_{b \in \mathrm{Dom}(B)} E_2(c, b) \cdot U_1(b)$$

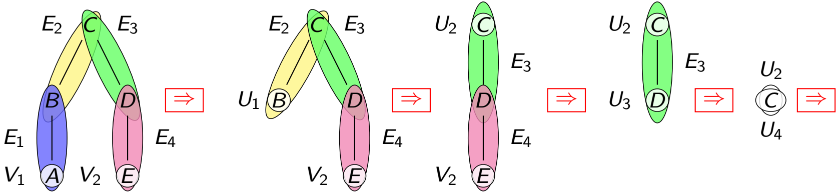
Example: Factorized Computation of COUNT(Q_4)



$$U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \qquad U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)$$

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

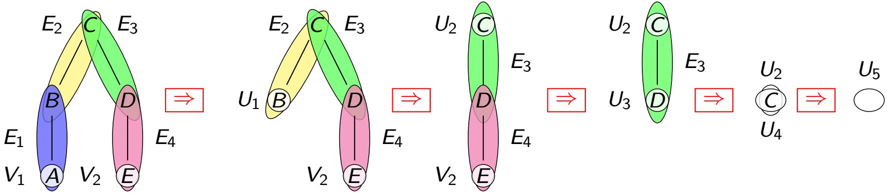
Example: Factorized Computation of COUNT(Q_4)



$$U_1(b) = \sum_{a \in \mathrm{Dom}(A)} V_1(a) \cdot E_1(b, a) \qquad U_2(c) = \sum_{b \in \mathrm{Dom}(B)} E_2(c, b) \cdot U_1(b)$$

$$U_3(d) = \sum_{e \in \mathrm{Dom}(E)} V_2(e) \cdot E_4(d, e) \qquad U_4(c) = \sum_{d \in \mathrm{Dom}(D)} E_3(c, d) \cdot U_3(d)$$

Example: Factorized Computation of COUNT(Q_4)

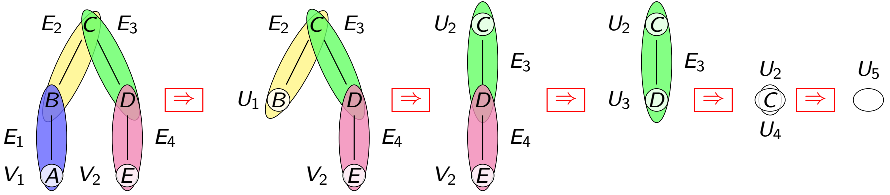


$$U_1(b) = \sum_{a \in \mathrm{Dom}(A)} V_1(a) \cdot E_1(b, a) \qquad U_2(c) = \sum_{b \in \mathrm{Dom}(B)} E_2(c, b) \cdot U_1(b)$$

$$U_3(d) = \sum_{e \in \mathrm{Dom}(E)} V_2(e) \cdot E_4(d, e) \qquad U_4(c) = \sum_{d \in \mathrm{Dom}(D)} E_3(c, d) \cdot U_3(d)$$

$$U_5 = \sum_{c \in \mathrm{Dom}(C)} U_2(c) \cdot U_4(c)$$

Example: Factorized Computation of COUNT(Q_4)



This computation strategy corresponds to the following query rewriting:

$$\sum_{a\in\mathrm{Dom}(A)}\sum_{b\in\mathrm{Dom}(B)}\sum_{c\in\mathrm{Dom}(C)}\sum_{d\in\mathrm{Dom}(D)}\sum_{e\in\mathrm{Dom}(E)} V_1(a)\cdot E_1(b,a)\cdot E_2(c,b)\cdot E_3(c,d)\cdot E_4(d,e)\cdot V_2(e)$$

$$=$$

$$\sum_{c\in\mathrm{Dom}(C)}\Big(\sum_{b\in\mathrm{Dom}(B)} E_2(c,b)\cdot\big(\sum_{a\in\mathrm{Dom}(A)} V_1(a)\cdot E_1(b,a)\big)\Big)\cdot$$
$$\Big(\sum_{d\in\mathrm{Dom}(D)} E_3(c,d)\cdot\big(\sum_{e\in\mathrm{Dom}(E)} E_4(d,e)\cdot V_2(e)\big)\Big)$$

Is Factorized Aggregate Computation Practical?

Experiments published in several papers, here a quick glimpse from [ANNOS17]

Retailer dataset (records)	excerpt (17M)	full (86M)
PostgreSQL computing the join	50.63 sec	216.56 sec
FDB computing both the join and the aggregates	25.51 sec	380.31 sec
Number of aggregates (scalar+group-by)	595+2,418	595+145k
FDB computing both the join and the aggregates	132.43 sec	1,819.80 sec
Number of aggregates (scalar+group-by)	158k+742k	158k+37M

In this experiment:

- FDB only used one core of a commodity machine
- For both PostgreSQL and FDB, the dataset was entirely in memory
- The aggregates represent gradients (or parts thereof) used for learning degree 1 and 2 polynomial regression models

Functional Aggregate Queries

Functional Aggregate Query

FAQ generalizes factorized aggregate computation to a host of problems.

We use the following notation ($i \in [n] = \{1, \ldots, n\}$):

- X_i are variables,
- x_i are values in discrete domain $\mathrm{Dom}(X_i)$
- $\mathbf{x} = (x_1, \ldots, x_n) \in \mathrm{Dom}(X_1) \times \cdots \times \mathrm{Dom}(X_n)$
- For any $S \subseteq [n]$,

$$\begin{aligned} \mathbf{x}_S &= (x_i)_{i \in S} \in \prod_{i \in S} \mathrm{Dom}(X_i) \\ \text{e.g. } \mathbf{x}_{\{2,5,8\}} &= (x_2, x_5, x_8) \in \mathrm{Dom}(X_2) \times \mathrm{Dom}(X_5) \times \mathrm{Dom}(X_8) \end{aligned}$$

Functional **A**ggregate **Q**uery: The Problem

$\psi_{14}(X_1, X_4)$

$\psi_{13}(X_1, X_3)$

$\psi_{23}(X_2, X_3)$

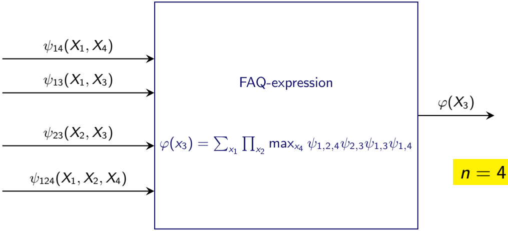
$\psi_{124}(X_1, X_2, X_4)$

FAQ-expression

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

$\varphi(X_3)$

Functional Aggregate Query: The Input



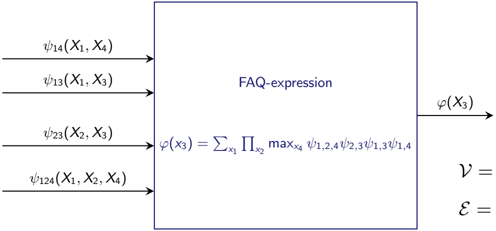
All functions have the same range **D**

$n = 4$

- *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* ψ_S

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

Functional Aggregate Query: The Input



All functions have the same range **D**

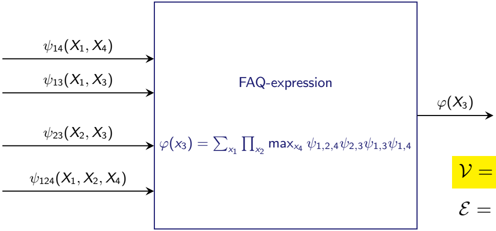
$$\mathcal{V} = \{1, 2, 3, 4\}$$

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

- *n variables* $X_1, \dots, X_n$
- 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* ψ_S

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

Functional Aggregate Query: The Input



All functions have the same range **D**

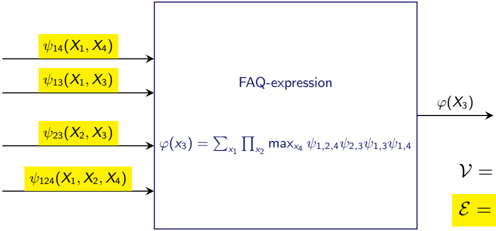
$\mathcal{V} = \{1, 2, 3, 4\}$

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

- *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* ψ_S

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

Functional Aggregate Query: The Input



All functions have the same range **D**

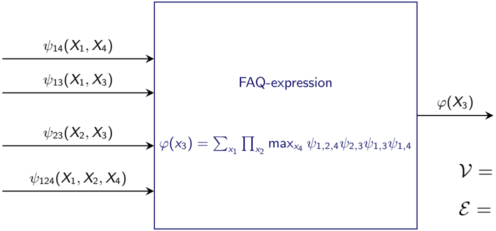
$\mathcal{V} = \{1, 2, 3, 4\}$

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

- *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* ψ_S

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

Functional Aggregate Query: The Input



All functions have the same range **D**

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

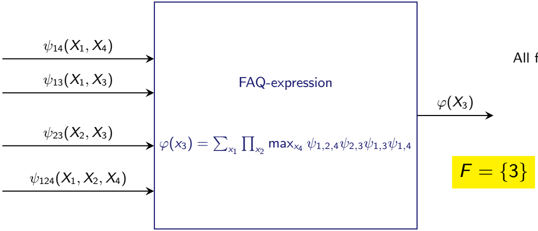
$$\mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\}$$

- *n variables* $X_1, \dots, X_n$
- 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* ψ_S

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

$\mathbf{R}_+$, $\{\text{true}, \text{false}\}$, $\{0, 1\}$, $2^{\mathcal{U}}$, etc.

Functional Aggregate Query: The Input



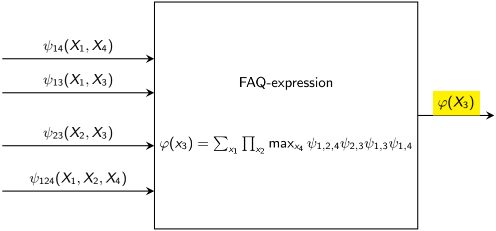
All functions have the same range **D**

- *n variables* $X_1, \dots, X_n$
- 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* ψ_S

$$\psi_S : \prod_{i \in S} \mathrm{Dom}(X_i) \to \underset{\substack{\uparrow \\ \mathbf{R}_+,\ \{\text{true}, \text{false}\},\ \{0,1\},\ 2^{\mathcal{U}},\ \text{etc.}}}{\mathbf{D}}$$

- a set $F \subseteq \mathcal{V}$ of *free variables* (wlog, $F = [f] = \{1, \dots, f\}$)

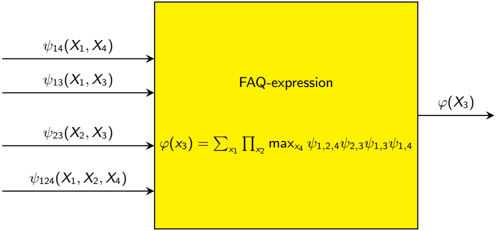
Functional Aggregate Query: The Output



All functions have the same range **D**

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

Functional Aggregate Query: The Output

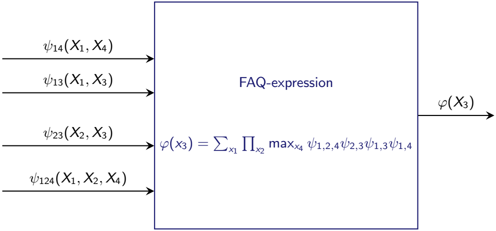


All functions have the same range **D**

- Compute the function $\varphi : \prod_{i \in F} \mathrm{Dom}(X_i) \to \mathbf{D}$.
- φ defined by the *FAQ-expression*

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus_{x_{f+1} \in \mathrm{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \mathrm{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \mathrm{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

Functional **A**ggregate **Q**uery: The Output



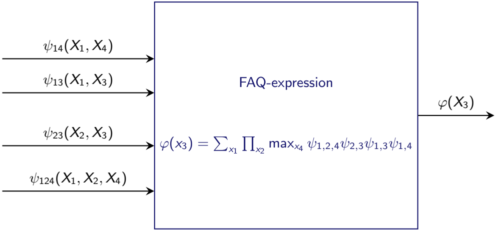
All functions have the same range **D**

- Compute the function $\varphi : \prod_{i \in F} \mathrm{Dom}(X_i) \to \mathbf{D}$.
- φ defined by the *FAQ-expression*

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus_{x_{f+1} \in \mathrm{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \mathrm{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \mathrm{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

- For each $\bigoplus^{(i)}$

Functional Aggregate Query: The Output



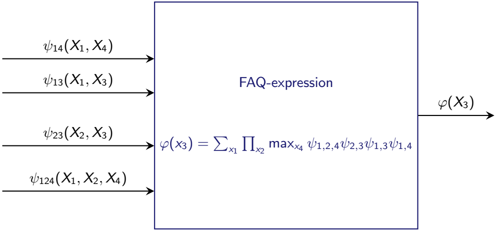
All functions have the same range **D**

- Compute the function $\varphi : \prod_{i \in F} \mathrm{Dom}(X_i) \to \mathbf{D}$.
- φ defined by the *FAQ-expression*

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus_{x_{f+1} \in \mathrm{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \mathrm{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \mathrm{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

- For each $\bigoplus^{(i)}$
  - Either $\left(\mathbf{D}, \bigoplus^{(i)}, \bigotimes\right)$ is a *commutative semiring*

Functional Aggregate Query: The Output



All functions have the same range **D**

- Compute the function $\varphi : \prod_{i \in F} \mathrm{Dom}(X_i) \to \mathbf{D}$.
- φ defined by the *FAQ-expression*

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus_{x_{f+1} \in \mathrm{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \mathrm{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \mathrm{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

- For each $\bigoplus^{(i)}$
  - Either $\left(\mathbf{D}, \bigoplus^{(i)}, \bigotimes\right)$ is a *commutative semiring*
  - Or $\bigoplus^{(i)} = \bigotimes$

Semirings

- $(\mathbf{D}, \oplus, \otimes)$ is a commutative semiring when

Additive identity	$\mathbf{0} \in \mathbf{D} : \mathbf{0} \oplus e = e \oplus \mathbf{0} = e$
Multiplicative identity	$\mathbf{1} \in \mathbf{D} : \mathbf{1} \otimes e = e \otimes \mathbf{1} = e$
Annihilation by $\mathbf{0}$	$\mathbf{0} \otimes e = e \otimes \mathbf{0} = \mathbf{0}$
Distributive law	$a \otimes b \oplus a \otimes c = a \otimes (b \oplus c)$

Semirings

- $(\mathbf{D}, \oplus, \otimes)$ is a commutative semiring when

Additive identity	$\mathbf{0} \in \mathbf{D} : \mathbf{0} \oplus e = e \oplus \mathbf{0} = e$
Multiplicative identity	$\mathbf{1} \in \mathbf{D} : \mathbf{1} \otimes e = e \otimes \mathbf{1} = e$
Annihilation by $\mathbf{0}$	$\mathbf{0} \otimes e = e \otimes \mathbf{0} = \mathbf{0}$
Distributive law	$a \otimes b \oplus a \otimes c = a \otimes (b \oplus c)$

- Common examples (there are many more!)

Boolean	$(\{\text{true}, \text{false}\}, \vee, \wedge)$
sum-product	$(\mathbb{R}, +, \times)$
max-product	$(\mathbb{R}_+, \max, \times)$
set	$(2^U, \cup, \cap)$

SumProduct $\subset$ FAQ

Problem (SumProduct)

Given a commutative semiring $(\mathbf{D}, \oplus, \otimes)$*, compute the function*

$$\varphi(x_1, \dots, x_f) = \bigoplus_{x_{f+1}} \bigoplus_{x_{f+2}} \cdots \bigoplus_{x_n} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

SumProduct $\subset$ FAQ

Problem (SumProduct)

Given a commutative semiring $(\mathbf{D}, \oplus, \otimes)$, *compute the function*

$$\varphi(x_1, \dots, x_f) = \bigoplus_{x_{f+1}} \bigoplus_{x_{f+2}} \cdots \bigoplus_{x_n} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

- SumProduct
  - Rina Dechter (Artificial Intelligence 1999 and earlier)
- $\equiv$ Marginalize a Product Function
  - Aji and McEliece (IEEE Trans. Inform. Theory 2000)

Many examples for SumProduct

- $(\{\text{true}, \text{false}\}, \vee, \wedge)$
  - Constraint satisfaction problems
  - Boolean conjunctive query evaluation
  - SAT
  - *k*-colorability
  - etc.
- $(U, \cup, \cap)$
  - Conjunctive query evaluation
- $(\mathbb{R}, +, \times)$
  - Permanent
  - DFT
  - Inference in probabilistic graphical models
  - #CSP
  - Matrix chain multiplication
  - Aggregates in DB
- $(\mathbf{R}_+, \max, \times)$
  - MAP queries in probabilistic graphical models

SumProduct Example 1: Boolean Query Evaluation

Boolean Conjunctive Queries:

- Boolean query Φ with set $\mathit{rels}(\Phi)$ of relation symbols
- Each relation symbol $R \in \mathit{rels}(\Phi)$ has variables $\mathit{vars}(R)$

$$\Phi = \exists X_1 \ldots \exists X_n : \bigwedge_{R \in \mathit{rels}(\Phi)} R(\mathit{vars}(R))$$

FAQ encoding:

$$\phi = \bigvee_{\mathbf{x}} \bigwedge_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S), \text{ where}$$

- Φ has the hypergraph $(\mathcal{V}, \mathcal{E})$ with
- $\mathcal{V} = \bigcup_{R \in \mathit{rels}(\Phi)} \mathit{vars}(R)$ and $\mathcal{E} = \{\mathit{vars}(R) \mid R \in \mathit{rels}(\Phi)\}$
- For each $S \in \mathcal{E}$, there is a factor ψ_S such that $\psi_S(\mathbf{x}_S) = (\mathbf{x}_S \in R)$

SumProduct Example 2: Matrix Chain Multiplication

$\boxed{\text{Compute the product } \mathbf{A} = \mathbf{A}_1 \cdots \mathbf{A}_n \text{ of } n \text{ matrices}}$

- Each matrix $\mathbf{A}_i$ is over field $\mathbb{F}$ and has dimensions $p_i \times p_{i+1}$

FAQ encoding:

- We use $n+1$ variables $X_1, \dots, X_{n+1}$ with domains $\mathrm{Dom}(X_i) = [p_i]$
- Each matrix $\mathbf{A}_i$ can be viewed as a function of two variables:

$$\psi_{i,i+1} : \mathrm{Dom}(X_i) \times \mathrm{Dom}(X_{i+1}) \to \mathbb{F}, \text{ where } \psi_{i,i+1}(x, y) = (\mathbf{A}_i)_{xy}$$

The problem is now to compute the FAQ expression

$$\phi(x_1, x_{n+1}) = \sum_{x_2 \in \mathrm{Dom}(X_2)} \cdots \sum_{x_n \in \mathrm{Dom}(X_n)} \prod_{i \in [n]} \psi_{i,i+1}(x_i, x_{i+1}).$$

SumProduct Example 3: Queries in Graphical Models

- Discrete undirected graphical model represented by a hypergraph $(\mathcal{V}, \mathcal{E})$
- $\mathcal{V} = \{X_1, \ldots, X_n\}$ consists of n discrete random variables
- There is a factor $\psi_S : \prod_{i \in S} \mathrm{Dom}(X_i) \to \mathbb{R}_+$ for each edge $S \in \mathcal{E}$

FAQ expression to compute the marginal Maximum A Posteriori estimates:

$$\phi(x_1, \ldots, x_f) = \max_{x_{f+1} \in \mathrm{Dom}(X_{f+1})} \cdots \max_{x_n \in \mathrm{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

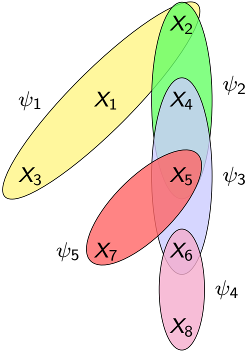
FAQ expression to compute the marginal distribution of variables $X_1, \ldots, X_f$:

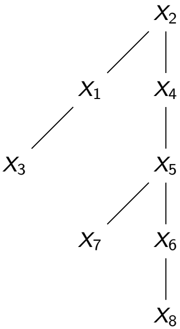
$$\phi(x_1, \ldots, x_f) = \sum_{x_{f+1} \in \mathrm{Dom}(X_{f+1})} \cdots \sum_{x_n \in \mathrm{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$$

For conditional distributions $p(\mathbf{x}_A \mid \mathbf{x}_B)$, the variables $\mathbf{X}_B$ are set to values $\mathbf{x}_B$.

Example 1: FAQ Computation using InsideOut

$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$





$key(X_2) = \emptyset$

$key(X_1) = \{X_2\}$

$key(X_3) = \{X_1, X_2\}$

$key(X_4) = \{X_2\}$

$key(X_5) = \{X_2, X_4\}$

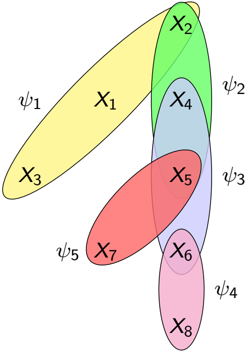
$key(X_6) = \{X_4, X_5\}$

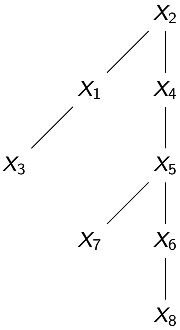
$key(X_8) = \{X_6\}$

$key(X_7) = \{X_5\}$

Example 1: FAQ Computation using InsideOut

$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$





$key(X_2) = \emptyset$

$key(X_1) = \{X_2\}$

$key(X_3) = \{X_1, X_2\}$

$key(X_4) = \{X_2\}$

$key(X_5) = \{X_2, X_4\}$

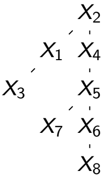
$key(X_6) = \{X_4, X_5\}$

$key(X_8) = \{X_6\}$

$key(X_7) = \{X_5\}$

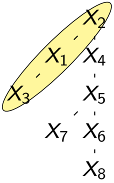
- $\rho^*(\varphi) = 4$, $s(\varphi) = 2$, $\mathit{fhtw}(\varphi) = 1$. The above variable order Δ has the free variables x_1, x_2, x_4 on top of the others and $\mathit{fhtw}(\Delta) = 1$.
- The query result has size: $O(N)$ when factorized; $O(N^2)$ when listed

Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

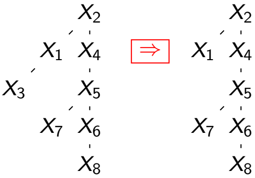
Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \underbrace{\left(\sum_{x_3} \psi_1(x_1, x_2, x_3) \right)}_{\psi_6(x_1, x_2)} \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

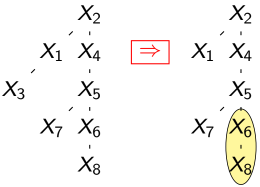
Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

Example 1: FAQ Computation using InsideOut

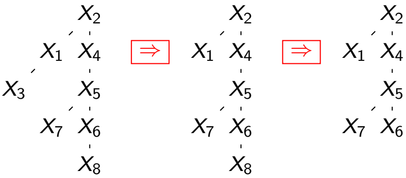


$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \underbrace{\left(\sum_{x_8} \psi_4(x_6, x_8) \right)}_{\psi_7(x_6)} \cdot \psi_5(x_5, x_7)$$

Example 1: FAQ Computation using InsideOut

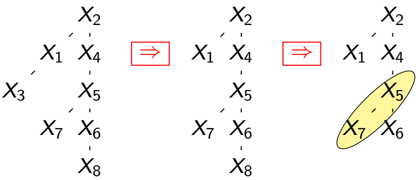


$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

Example 1: FAQ Computation using InsideOut



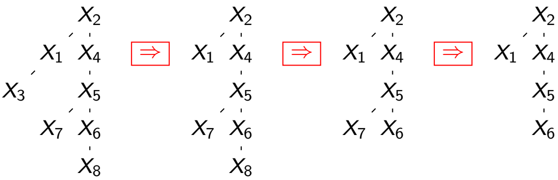
$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \underbrace{\left(\sum_{x_7} \psi_5(x_5, x_7) \right)}_{\psi_8(x_5)}$$

Example 1: FAQ Computation using InsideOut



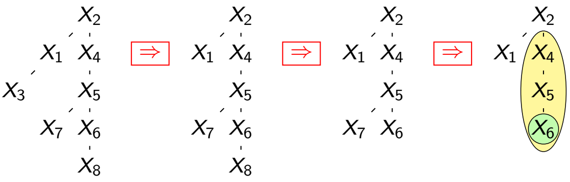
$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

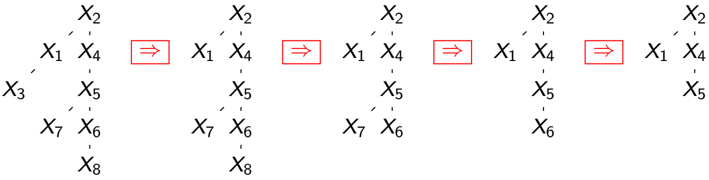
$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \underbrace{\left(\sum_{x_6} \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \right)}_{\psi_9(x_4, x_5)} \cdot \psi_8(x_5)$$

Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

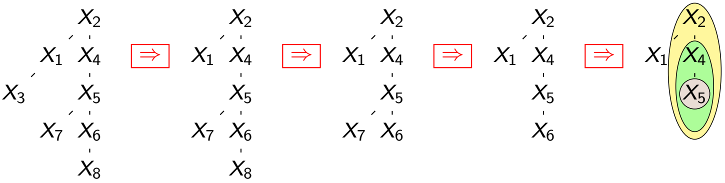
$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

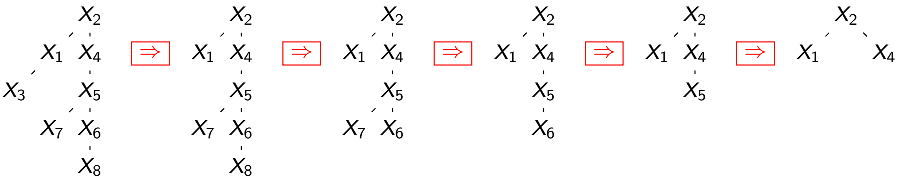
$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \psi_6(x_1, x_2) \cdot \underbrace{\left(\sum_{x_5} \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \right)}_{\psi_{10}(x_2, x_4)}$$

Example 1: FAQ Computation using InsideOut



$$\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \quad \widetilde{O}(N)$$

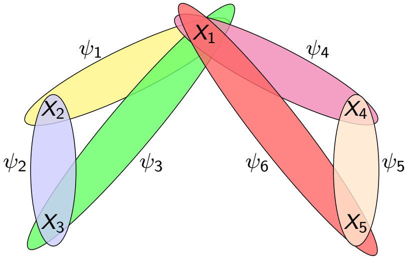
$$\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

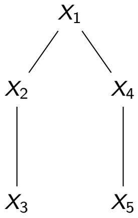
$$\varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \quad \widetilde{O}(N)$$

$$\varphi(x_1, x_2, x_4) = \psi_6(x_1, x_2) \cdot \psi_{10}(x_2, x_4) \quad \widetilde{O}(N)$$

Example 2: FAQ Computation with Indicator Projections

$$\varphi(x_1) = \sum_{x_2,x_3,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_2(x_2,x_3)\cdot\psi_3(x_3,x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)$$





$key(X_1) = \emptyset$

$key(X_2) = \{X_1\}$

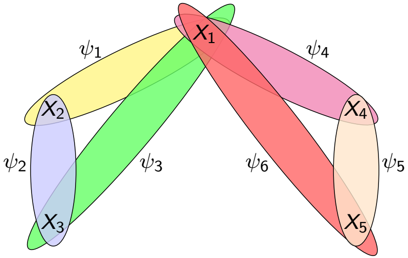
$key(X_3) = \{X_1, X_2\}$

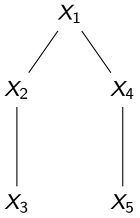
$key(X_4) = \{X_1\}$

$key(X_5) = \{X_1, X_4\}$

Example 2: FAQ Computation with Indicator Projections

$$\varphi(x_1) = \sum_{x_2,x_3,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_2(x_2,x_3)\cdot\psi_3(x_3,x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)$$

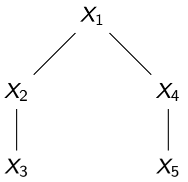




$$key(X_1) = \emptyset$$
$$key(X_2) = \{X_1\}$$
$$key(X_3) = \{X_1, X_2\}$$
$$key(X_4) = \{X_1\}$$
$$key(X_5) = \{X_1, X_4\}$$

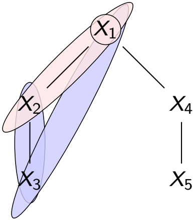
- $\rho^*(\varphi) = 2.5$, $s(\varphi) = 1.5$, $\mathit{fhtw}(\varphi) = 1.5$. The above variable order Δ has the free variable x_1 on top of the others and $\mathit{fhtw}(\Delta) = 1.5$.
- The (unary) query result has size $O(N)$ when factorized or listed.

Example 2: FAQ Computation with Indicator Projections



$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

Example 2: FAQ Computation with Indicator Projections

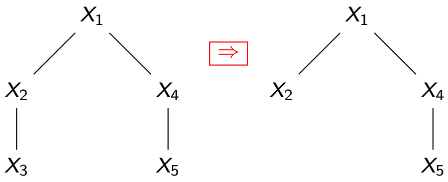


$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \underbrace{\left(\sum_{x_3} \psi_1'(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4'(x_1) \cdot \psi_6'(x_1) \right)}_{\psi_7(x_1, x_2)} \cdot$$

$$\psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

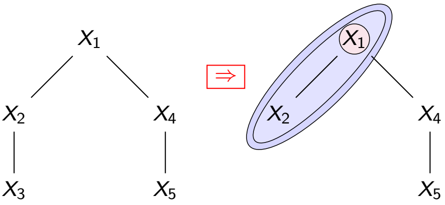
Example 2: FAQ Computation with Indicator Projections



$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

$$\varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N^{1.5})$$

Example 2: FAQ Computation with Indicator Projections



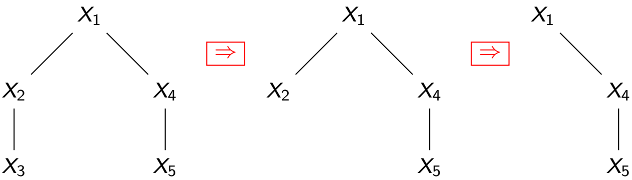
$$\varphi(x_1) = \sum_{x_2,x_3,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_2(x_2,x_3)\cdot\psi_3(x_3,x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)$$

$$\varphi(x_1) = \sum_{x_2,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_7(x_1,x_2)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \sum_{x_4,x_5} \underbrace{\left(\sum_{x_2} \psi_1(x_1,x_2)\cdot\psi_7(x_1,x_2)\cdot\psi'_4(x_1)\cdot\psi'_6(x_1)\right)}_{\psi_8(x_1)}\cdot$$

$$\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)$$

Example 2: FAQ Computation with Indicator Projections

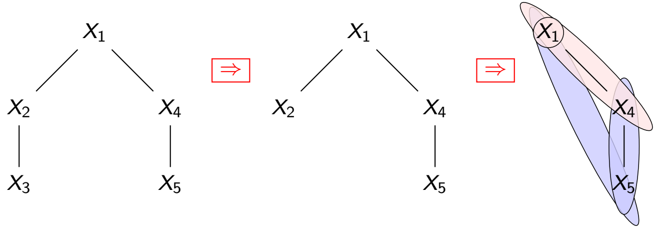


$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

$$\varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N)$$

Example 2: FAQ Computation with Indicator Projections



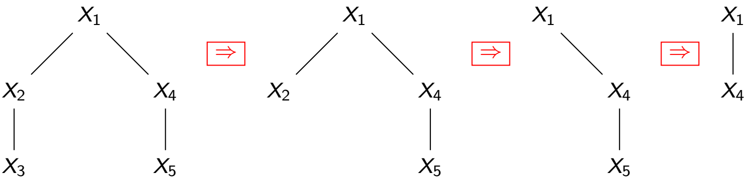
$$\varphi(x_1) = \sum_{x_2,x_3,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_2(x_2,x_3)\cdot\psi_3(x_3,x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)$$

$$\varphi(x_1) = \sum_{x_2,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_7(x_1,x_2)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \sum_{x_4,x_5} \psi_8(x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1) \quad \widetilde{O}(N)$$

$$\varphi(x_1) = \sum_{x_4} \psi_8(x_1)\cdot\psi_4(x_1,x_4)\cdot\underbrace{\left(\sum_{x_5} \psi_8'(x_1)\cdot\psi_4'(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)\right)}_{\psi_9(x_1,x_4)}$$

Example 2: FAQ Computation with Indicator Projections



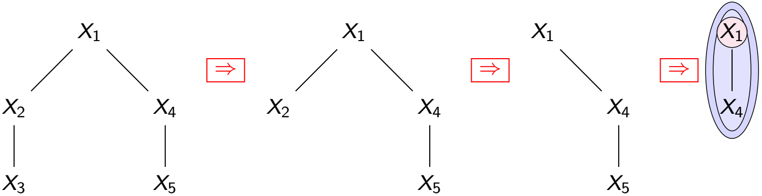
$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

$$\varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N)$$

$$\varphi(x_1) = \sum_{x_4} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_9(x_1, x_4) \quad \widetilde{O}(N^{1.5})$$

Example 2: FAQ Computation with Indicator Projections



$$\varphi(x_1) = \sum_{x_2,x_3,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_2(x_2,x_3)\cdot\psi_3(x_3,x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1)$$

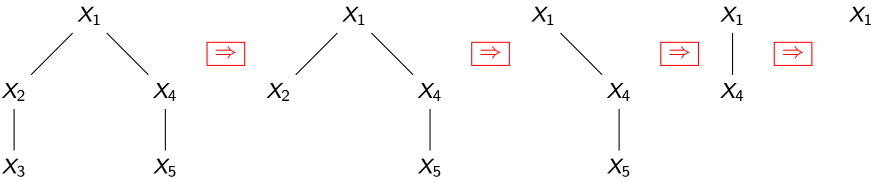
$$\varphi(x_1) = \sum_{x_2,x_4,x_5} \psi_1(x_1,x_2)\cdot\psi_7(x_1,x_2)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \sum_{x_4,x_5} \psi_8(x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_5(x_4,x_5)\cdot\psi_6(x_5,x_1) \quad \widetilde{O}(N)$$

$$\varphi(x_1) = \sum_{x_4} \psi_8(x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_9(x_1,x_4) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \psi_8(x_1)\cdot\underbrace{\left(\sum_{x_4} \psi'_8(x_1)\cdot\psi_4(x_1,x_4)\cdot\psi_9(x_1,x_4)\right)}_{\psi_{10}(x_1)}$$

Example 2: FAQ Computation with Indicator Projections



$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

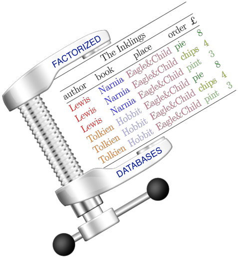
$$\varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \widetilde{O}(N)$$

$$\varphi(x_1) = \sum_{x_4} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_9(x_1, x_4) \quad \widetilde{O}(N^{1.5})$$

$$\varphi(x_1) = \psi_8(x_1) \cdot \psi_{10}(x_1) \quad \widetilde{O}(N)$$

Outline



Part 1. Joins

Part 2. Aggregates

Part 3. Optimization

Optimization Inside the Database

Why solving optimization problems aka *analytics* inside the database?

1. Bring analytics close to data
   ⇒ Save non-trivial export/import time

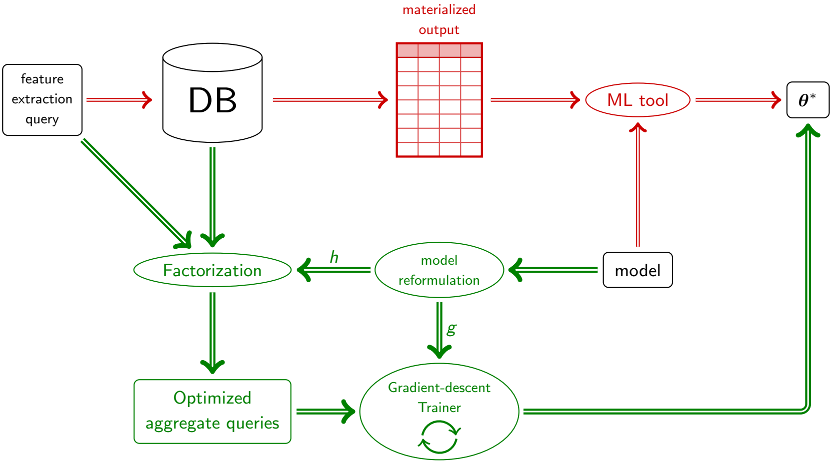
2. Large chunks of analytics code can be rewritten into SumProduct FAQs

   ⇒ Use scalable/factorized query processing

Hot topic in the current DB research & industry landscape:

- Very recent tutorials and research agenda [A17,KBY17,PRWZ17]
- This tutorial highlights our recent work [SOC16,ANNOS17]

In-database vs. Out-of-database Analytics



- h and g are functions over features and respectively model parameters
- θ^* are the parameters of the learned model

Plan for Part 3 on Optimization

- 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 & experimental analysis
  - Model training faster than computing the input to external ML library!

In-Database Analytics Approach in This Tutorial

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

Deployed in industrial 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

Retail Example

Simplified Retail Example

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

$$Q(\texttt{sku}, \texttt{store}, \texttt{color}, \texttt{city}, \texttt{country}, \mathit{unitsSold}) = \\ R_1(\texttt{sku}, \texttt{store}, \texttt{day}, \mathit{unitsSold}), R_2(\texttt{sku}, \texttt{color}), \\ R_3(\texttt{day}, \texttt{quarter}), R_4(\text{store}, \text{city}), R_5(\texttt{city}, \texttt{country}).$$

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

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

- The parameters $\boldsymbol{\theta}$ are obtained by minimizing the square loss function:

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

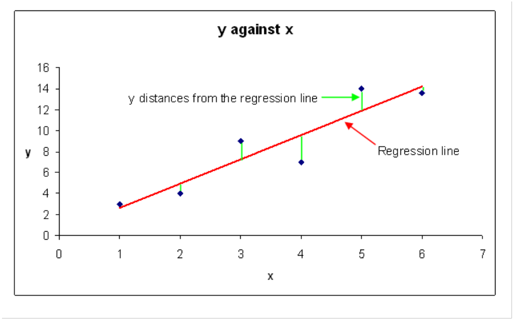
Recap: One-hot encoding of categorical variables

- *Continuous* variables are mapped to scalars
  - $y_{unitsSold} \in \mathbb{R}$.
- 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 $\mathbf{x}_{\texttt{country}} = [x_{\texttt{vietnam}}, x_{\texttt{england}}]^\top \in (\{0,1\}^2)^\top$.
  - $\mathbf{x}_{\texttt{country}} = [0,1]^\top$ for a tuple with country = ''england''

$$\boxed{\text{One-hot encoding leads to very wide training datasets and many 0-values.}}$$

Recap: Role of the Least Square Loss Function

Goal: Describe a linear relationship $fun(x) = \theta_1 x + \theta_0$ between variables x and $y = fun(x)$, 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 $fun(x)$ such that the (green) error $\sum_{i\in[n]}(fun(x_i) - y_i)^2$ is minimized
- The role of the ℓ_2-regularization $\|\boldsymbol{\theta}\|_2^2 = \theta_0^2 + \theta_1^2$ is to avoid over/under-fitting. It gives preference to functions *fun* with smaller norms.

From Optimization to SumProduct FAQ Queries

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

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

From Optimization to SumProduct FAQ Queries

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

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

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

$$\boldsymbol{\sigma}_{ij} = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top \qquad \mathbf{c}_i = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} y \cdot \mathbf{x}_i \qquad s_Y = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} y^2.$$

From Optimization to SumProduct FAQ Queries

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

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

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

$$\boldsymbol{\sigma}_{ij} = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top \qquad \mathbf{c}_i = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} y \cdot \mathbf{x}_i \qquad s_Y = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} y^2.$$

Then,

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

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

$$\boldsymbol{\nabla} J(\boldsymbol{\theta}) = \boldsymbol{\Sigma}\boldsymbol{\theta} - \mathbf{c} + \lambda \boldsymbol{\theta}$$

Expressing $\boldsymbol{\Sigma}$, $\mathbf{c}$, s_Y as SumProduct FAQ Queries

FAQ queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

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

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

- $\boldsymbol{x}_i$ categorical, x_j continuous $\Rightarrow$ FAQ query with one free variable

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

- $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ categorical $\Rightarrow$ FAQ query with two free variables

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

$\mathcal{S}(R_k)$ is the set of variables of R_k; $\mathbf{a}_{\mathcal{S}(R_k))}$ is a tuple in relation R_k; $\mathbf{1}_E$ is the Kronecker delta that is 1 (0) whenever the event E holds (does not hold).

Expressing $\boldsymbol{\Sigma}$, $\mathbf{c}$, s_Y as SQL Queries

SQL queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

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

  SELECT SUM(x_i*x_j) **FROM** D;

- $\boldsymbol{x}_i$ categorical, x_j continuous $\Rightarrow$ SQL query with one group-by attribute

  SELECT x_i, **SUM**(x_j) **FROM** D **GROUP BY** x_i;

- $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ categorical $\Rightarrow$ SQL query with two free variables

  SELECT x_i, x_j, **SUM**(1) **FROM** D **GROUP BY** x_i, x_j;

- $\boldsymbol{\Sigma}$, $\mathbf{c}$, s_Y are all aggregates that can be computed inside the database!
- We avoid one-hot/sparse encoding of the input data.

Dimensionality Reduction with Functional Dependencies

Consider the functional dependency `city` $\rightarrow$ `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:

- $x_{\texttt{vietnam}} = x_{\texttt{saigon}} + x_{\texttt{hanoi}}$

  That is: If country is vietnam, then city is either saigon or hanoi
  if $x_{\texttt{vietnam}} = 1$ then either $x_{\texttt{saigon}} = 1$ or $x_{\texttt{hanoi}} = 1$

- $x_{\texttt{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}$

  That is: If country is england, then city is either oxford, leeds, or bristol
  if $x_{\texttt{england}} = 1$ then either $x_{\texttt{oxford}} = 1$ or $x_{\texttt{leeds}} = 1$ or $x_{\texttt{bristol}} = 1$

Dimensionality Reduction with Functional Dependencies

- Identities due to one-hot encoding

$$x_{\texttt{vietnam}} = x_{\texttt{saigon}} + x_{\texttt{hanoi}}$$

$$x_{\texttt{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}$$

- Encode $\mathbf{x}_{\texttt{country}}$ as $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$, where

$$\mathbf{R} = \begin{array}{ccccc|l} \text{saigon} & \text{hanoi} & \text{oxford} & \text{leeds} & \text{bristol} & \\ 1 & 1 & 0 & 0 & 0 & \text{vietnam} \\ 0 & 0 & 1 & 1 & 1 & \text{england} \end{array}$$

For instance, if $\texttt{city}$ is $\texttt{saigon}$, i.e., $\mathbf{x}_{\texttt{city}} = [1, 0, 0, 0, 0]^\top$, then $\texttt{country}$ is $\texttt{vietnam}$, i.e., $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}} = [1, 0]^\top$.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Dimensionality Reduction with Functional Dependencies

- Functional dependency: `city` $\rightarrow$ `country`
- $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$
- Replace all occurrences of $\mathbf{x}_{\texttt{country}}$ by $\mathbf{R}\mathbf{x}_{\texttt{city}}$:

$$
\begin{aligned}
& \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{x}_{\texttt{country}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
= & \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{R}\mathbf{x}_{\texttt{city}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
= & \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \left\langle \underbrace{\mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}}_{\gamma_{\texttt{city}}}, \mathbf{x}_{\texttt{city}} \right\rangle
\end{aligned}
$$

Dimensionality Reduction with Functional Dependencies

- Functional dependency: `city` $\rightarrow$ `country`
- $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$
- Replace all occurrences of $\mathbf{x}_{\texttt{country}}$ by $\mathbf{R}\mathbf{x}_{\texttt{city}}$:

$$
\begin{aligned}
&\sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{x}_{\texttt{country}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
=& \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{R}\mathbf{x}_{\texttt{city}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
=& \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \left\langle \underbrace{\mathbf{R}^{\top} \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}}_{\gamma_{\texttt{city}}}, \mathbf{x}_{\texttt{city}} \right\rangle
\end{aligned}
$$

- We avoid computing aggregates over $\mathbf{x}_{\texttt{country}}$.
- We reparameterize the problem and ignore parameters $\boldsymbol{\theta}_{\texttt{country}}$.
- What about the penalty term in the loss function?

Dimensionality Reduction with Functional Dependencies

- Functional dependency: `city` $\rightarrow$ `country`
- $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$
- $\boldsymbol{\gamma}_{\texttt{city}} = \mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}$
- Rewrite the penalty term

$$\|\boldsymbol{\theta}\|_2^2 = \sum_{j \neq \texttt{city}} \|\boldsymbol{\theta}_j\|_2^2 + \left\|\boldsymbol{\gamma}_{\texttt{city}} - \mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}}\right\|_2^2 + \|\boldsymbol{\theta}_{\texttt{country}}\|_2^2$$

- "Optimize out" $\boldsymbol{\theta}_{\texttt{country}}$ by expressing it in terms of $\gamma_{\texttt{city}}$:

$$\boldsymbol{\theta}_{\texttt{country}} = (\mathbf{I}_{\texttt{country}} + \mathbf{R}\mathbf{R}^\top)^{-1}\mathbf{R}\boldsymbol{\gamma}_{\texttt{city}} = \mathbf{R}(\mathbf{I}_{\texttt{city}} + \mathbf{R}^\top\mathbf{R})^{-1}\boldsymbol{\gamma}_{\texttt{city}}$$

  $\mathbf{I}_{\texttt{country}}$ is the order-$N_{\texttt{country}}$ identity matrix and similarly for $\mathbf{I}_{\texttt{city}}$.
- The penalty term becomes

$$\|\boldsymbol{\theta}\|_2^2 = \sum_{j \neq \texttt{city}} \|\boldsymbol{\theta}_j\|_2^2 + \left\langle (\mathbf{I}_{\texttt{city}} + \mathbf{R}^\top\mathbf{R})^{-1}\boldsymbol{\gamma}_{\texttt{city}}, \boldsymbol{\gamma}_{\texttt{city}} \right\rangle$$

The General Picture

General Problem Formulation

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

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

- $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p) \in \mathbf{R}^p$ are parameters
- functions $g : \mathbf{R}^p \to \mathbf{R}^m$ and $h : \mathbf{R}^n \to \mathbf{R}^m$ 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, Ω is the regularizer
- D is the training dataset with features $\mathbf{x}$ 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:

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

Under

- square loss $\mathcal{L}$, ℓ_2-regularization,
- data points $\mathbf{x} = (x_0, x_1, \ldots, x_n, y)$,
- $p = n + 1$ parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_n)$,
- $x_0 = 1$ corresponds to the bias parameter θ_0
- g and h identity functions $\mathbf{g}(\boldsymbol{\theta}) = \boldsymbol{\theta}$ and $h(\mathbf{x}) = \mathbf{x}$,

we obtain the following formulation for ridge linear regression:

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

Special Case: Degree-d Polynomial Regression

General problem formulation:

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

Under

- square loss $\mathcal{L}$, ℓ_2-regularization,
- data points $\mathbf{x} = (x_0, x_1, \ldots, x_n, y)$,
- $p = m = 1 + n + n^2 + \cdots + n^d$ parameters $\boldsymbol{\theta} = (\theta_{\mathbf{a}})$, where $\mathbf{a} = (a_1, \ldots, a_n)$ is a tuple of non-negative integers such that $\|\mathbf{a}\|_1 \leq d$.
- $g(\boldsymbol{\theta}) = \boldsymbol{\theta}$,
- the components of h are given by $h_{\mathbf{a}}(\mathbf{x}) = \prod_{i=1}^{n} x_i^{a_i}$.

we obtain the following formulation for polynomial regression:

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

Special Case: Factorization Machines

Under

- square loss $\mathcal{L}$, ℓ_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:

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

where

$$\begin{aligned} h_S(\mathbf{x}) &= \prod_{i\in S} x_i, \text{ for } S \subseteq [n], |S| \leq 2 \\ g_S(\boldsymbol{\theta}) &= \begin{cases} \theta_0 & \text{when } |S| = 0 \\ \theta_i & \text{when } S = \{i\} \\ \sum_{\ell=1}^{r} \theta_i^{(\ell)} \theta_j^{(\ell)} & \text{when } S = \{i, j\}. \end{cases} \end{aligned}$$

Special Case: Classification methods

$\boxed{\text{Examples: support vector machines, logistic regression, Adaboost}}$

- Typically, the regularizer is $\frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$
- The response is now binary: $y \in \{\pm 1\}$
- The loss function $\mathcal{L}(\gamma, y)$, where $\gamma := \langle g(\boldsymbol{\theta}), h(\mathbf{x})\rangle$, 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)

$\boxed{\text{Repeatedly update } \boldsymbol{\theta} \text{ in the direction of the gradient until convergence}}$

$\boldsymbol{\theta} \leftarrow$ a random point;
while *not converged yet* **do**
  $\alpha \leftarrow$ next step size;
  $\mathbf{d} \leftarrow \boldsymbol{\nabla} J(\boldsymbol{\theta})$;
  while $\left(J(\boldsymbol{\theta} - \alpha \cdot \mathbf{d}) \geq J(\boldsymbol{\theta}) - \frac{\alpha}{2} \cdot \|\mathbf{d}\|_2^2\right)$ **do** $\alpha \leftarrow \alpha/2$; // line search
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \mathbf{d}$;
end

BGD needs:

- Computation of the gradient vector $\boldsymbol{\nabla} J(\boldsymbol{\theta})$
  - Its data-dependent component is computed once for all iterations
- Point evaluation $J(\boldsymbol{\theta})$
  - A few times per iteration to adjust α using line search

Compute Parameters $\boldsymbol{\theta}$ using BGD

Immediate extension of the linear regression case discussed before.

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

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

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

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

Under square loss $\mathcal{L}$ and ℓ_2-regularization:

$$J(\boldsymbol{\theta}) = \frac{1}{2} g(\boldsymbol{\theta})^\top \mathbf{\Sigma} g(\boldsymbol{\theta}) - \langle g(\boldsymbol{\theta}), \mathbf{c}\rangle + \frac{s_Y}{2} + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$$

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

Summing Up

Insight #1:

- $\mathbf{\Sigma}$, $\mathbf{c}$, s_Y are queries that can be computed **inside the database**!
  - They can take **much less time** than computing the feature extraction query

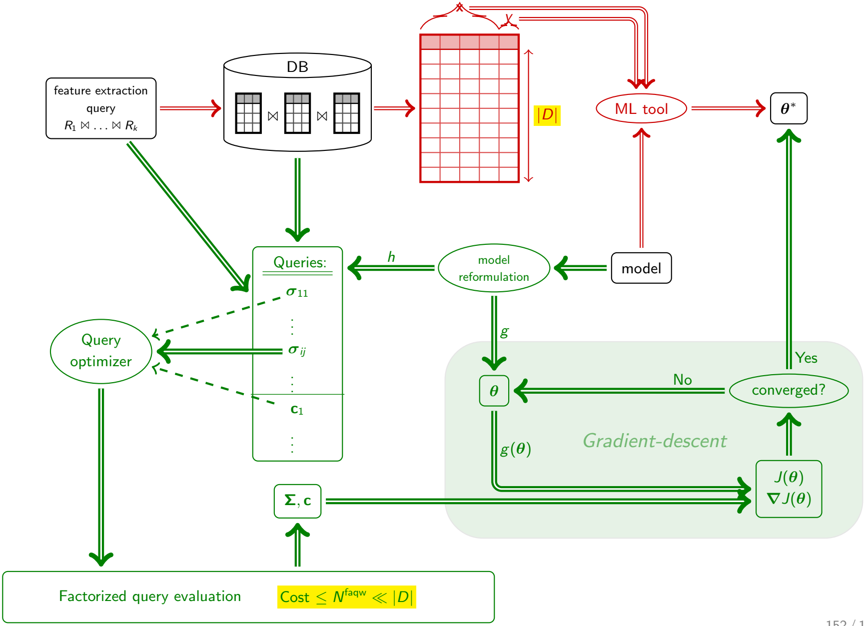
Insight #2:

- 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



Complexity & Experimental Analysis

Complexity Analysis: The General Case

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

Let:

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

The tensors $\boldsymbol{\sigma}_{ij}$ and $\mathbf{c}_j$ can be sparsely represented by queries with group-by variables and can be computed in time

$$\widetilde{O}\left(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [m]} \left(N^{\mathrm{faqw}(i,j)} + |\boldsymbol{\sigma}_{ij}|\right)\right).$$

Complexity Analysis: Continuous Features Only

Complexity in the general case: [ANNOS17]

$$\widetilde{O}\left(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [m]} (N^{\mathsf{faqw}(i,j)} + |\boldsymbol{\sigma}_{ij}|)\right).$$

Complexity in case all features are continuous: [SOC16]

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

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; $c \leq d$
- ρ^* = fractional edge cover number of query Q

Comparison against state of the art: [ANNOS17]

- $faqw(i,j) \leq fhtw + c - 1$ and $|\boldsymbol{\sigma}_{ij}| \leq \min\{|D|, N^c\}$.

- For any query Q with $\rho^* > fhtw + c - 1$, there are infinitely many database instances of size N for which

$$\lim_{N\to\infty} \frac{|D|}{\sum_{i,j\in[m]}(N^{\text{faqw}(i,j)} + |\boldsymbol{\sigma}_{ij}|)\log N} = \infty.$$

- Computing $\boldsymbol{\sigma}_{ij}$ for degree-d polynomial regression takes

$$\widetilde{O}(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot m^2 \cdot N^{\text{fhtw}+2d}).$$

under **one-hot encoding** of categorical variables.

Factorized Machine Learning in Practice

Experiments published in several papers, here a glimpse from [ANNOS17]

Retailer dataset (records)	excerpt (17M)	full (86M)
Linear Regression		
Number of features (Cont. + Categ.)	33+55	33+3702
MADlib (ols)	1,898.35 sec	–
PostgreSQL + R (qr decomposition)	798.96 sec	–
FDB	25.53 sec	380.31 sec
Polynomial Regression degree 2		
Number of features (Cont. + Categ.)	562+2363	562+154K
MADlib	$>$ 22h	–
PostgreSQL + R	–	–
FDB	135.7 sec	2039.31 sec

- We measure end-to-end performance: joins + aggregates + convergence
- R: "-" means R's data frame limit is exceeded and cannot run.
- MADlib: "-" means it cannot one-hot encode the data in a relation with more than 1600 columns.

One idea to rule them all

and at their core FACTORIZE them!

Thank you!

Quizzes

QUIZ 1: Joins (1/3)

For each of the following queries, please show the following:

1. Hypertree decomposition and variable order for query.
2. The fractional edge cover number and the fractional hypertree width (assume all relations have the same size).

Path Query of length n:

$$P_n(X_1, \ldots, X_{n+1}) = R_1(X_1, X_2), R_2(X_2, X_3), R_3(X_3, X_4), \ldots, R_n(X_n, X_{n+1}).$$

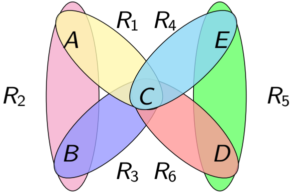
QUIZ 1: Joins (2/3)

For each of of the following queries, please show the following:

1. Hypertree decomposition and variable order for query.
2. The fractional edge cover number and the fractional hypertree width (assume all relations have the same size).

Bowtie Query:

$$Q_{\bowtie}(A, B, C, D, E) = R_1(A, C), R_2(A, B), R_3(B, C), R_4(C, E), R_5(E, D), R_6(C, D).$$



QUIZ 1: Joins (3/3)

For each of of the following queries, please show the following:

1. Hypertree decomposition and variable order for query.
2. The fractional edge cover number and the fractional hypertree width (assume all relations have the same size).

Loomis-Whitney Queries of length n: A LW_n query has n variables $X_1, \ldots, X_n$ and n relation symbols such that for every $i \in [n]$ the relation symbol R_i has variables $\{X_1, \ldots, X_n\} - \{X_i\}$:

$$LW_n(X_1, \ldots, X_n) = R_1(X_2, \ldots, X_n), \ldots, R_i(X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n), \ldots,$$
$$R_n(X_1, \ldots, X_{n-1})$$

LW_3 is the triangle query.

QUIZ 2: Aggregates (1/2)

For each of of the following functional aggregate queries:

1. Give a hypertree decomposition and variable order.
2. If you were to compute it as stated below (with all sums done after the products), what would be its time complexity? (Assume all functions have the same size.)
3. Is there an equivalent rewriting of φ that would allow for quadratic time complexity? What about linear time?

The *n*-hop query:

$$\varphi(x_1, x_{n+1}) = \sum_{x_2,\ldots,x_n} \psi_1(X_1, X_2) \cdot \psi_2(X_2, X_3) \cdot \psi_3(X_3, X_4) \cdot \ldots \cdot \psi_n(X_n, X_{n+1}).$$

QUIZ 2: Aggregates (2/2)

For each of of the following functional aggregate queries:

1. Give a hypertree decomposition and variable order.
2. If you were to compute it as stated below (with all sums done after the products), what would be its time complexity? Assume all functions have the same size.
3. Is there an equivalent rewriting of φ that would allow for quadratic time complexity? What about linear time?

Query:

$$\varphi = \sum_a \sum_b \sum_c \sum_f \sum_d \sum_e \psi_1(a, b) \cdot \psi_2(a, c) \cdot \psi_3(c, d) \cdot \psi_4(b, c, e) \cdot \psi_5(e, f).$$

QUIZ 3: Optimization

Assume that the natural join of the following relations provides the features we use to predict revenue:

```
Sales(store_id, product_id, quantity, revenue),
Product(product_id, color),
Store(store_id, distance_city_center).
```

Variables `revenue`, `quantity`, and `distance_city_center` stand for continuous features, while `product_id` and `color` for categorical features.

1. Give the FAQs required to compute the gradient of the squares loss function for learning a ridge linear regression models with the above features.
2. We know that `product_id` functionally determines `color`. Give a rewriting of the objective function that exploits the functional dependency.
3. The FAQs require the computation of a lot of common sub-problems. Can you think of ways to share as much computation as possible?

References

References on Join Computation

GLS99 **Hypertree decompositions and tractable queries.**
Gottlob, Leone, Scarcello. In PODS 1999.
https://arxiv.org/abs/cs/9812022

AGM08 **Size bounds and query plans for relational joins.**
Atserias, Grohe, Marx. In FOCS 2008 and SIAM J. Comput., 42(4) 2013.
http://epubs.siam.org/doi/10.1137/110859440

M10 **Approximating fractional hypertree width.**
Marx. In ACM TALG 2010.
urlhttp://dl.acm.org/citation.cfm?id=1721845

NPRR12 **Worst-case optimal join algorithms: [extended abstract]**
Ngo, Porat, Ré, Rudra. In PODS 2012.
urhttp://dl.acm.org/citation.cfm?id=2213565

OZ12 **Factorised representations of query results: size bounds and readability.**
Olteanu, Zavodny. In ICDT 2012.
http://dl.acm.org/citation.cfm?doid=2274576.2274607
Also https://arxiv.org/abs/1104.0867, April 2011.

NRR13 **Skew Strikes Back: New Developments in the Theory of Join Algorithms.**
Ngo, Ré, Rudra. In SIGMOD Rec. 2013. https://arxiv.org/abs/1310.3314

References on Join Computation

V14 **Triejoin: A Simple, Worst-Case Optimal Join Algorithm.**
Veldhuizen. In ICDT 2014.
http://openproceedings.org/ICDT/2014/paper_13.pdf

OZ15 **Size Bounds for Factorised Representations of Query Results.**
Olteanu, Zavodny. In ACM TODS 2015.
http://dl.acm.org/citation.cfm?doid=2656335

CO15 **Worst-Case Optimal Join At A Time.**
Ciucanu, Olteanu. Technical report, Oxford, Nov 2015.

ANS17 **What do Shannon-type inequalities, submodular width, and disjunctive Datalog have to do with one another?**
Abo Khamis, Ngo, Suciu. In PODS 2017.
https://arxiv.org/abs/1612.02503

KO17 **Covers of Query Results.**
Kara, Olteanu. Technical report, Oxford, March 2017. To appear in ICDT 2018.
https://arxiv.org/abs/1709.01600

References on Aggregate Computation

BKOZ13 **Aggregation and Ordering in Factorised Databases.**
Bakibayev, Kocisky, Olteanu, Zavodny. In PVLDB 2013.
`https://arxiv.org/abs/1307.0441`

ANR16 **FAQ: Questions Asked Frequently.**
Abo Khamis, Ngo, Rudra. In PODS 2016.
`https://arxiv.org/abs/1504.04044`

References on In-Database Analytics

SOC16 **Learning Linear Regression Models over Factorized Joins.**
Schleich, Olteanu, Ciucanu. In SIGMOD 2016.
http://dl.acm.org/citation.cfm?doid=2882903.2882939

A17 **Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151).**
Abiteboul et al. In SIGMOD Rec. 2017.
https://arxiv.org/pdf/1701.09007.pdf

ANNOS17 **In-Database Learning with Sparse Tensors.**
Abo Khamis, Ngo, Nguyen, Olteanu, Schleich. March 2017. To appear in PODS 2018.
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 View Maintenance with Triple Lock Factorisation Benefits.**
Nikolic, Olteanu. March 2017. To appear in SIGMOD 2018.
https://arxiv.org/abs/1703.07484

PRWZ17 **Data Management Challenges in Production Machine Learning.**
Polyzotis, Roy, Whang, Zinkevich. In SIGMOD 2017, Tutorial.
http://dl.acm.org/citation.cfm?doid=3035918.3054782