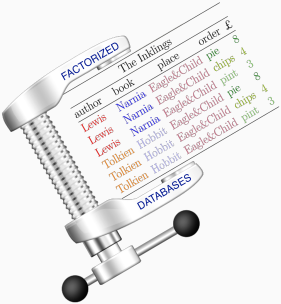
Learning Models over Relational Databases

fdbresearch.github.io relational.ai

Dan Olteanu

Oxford & relationalAI

ICDT & EDBT Keynote
Lisbon, March 2019



Acknowledgments

FDB team, in particular:

Jakub

Max

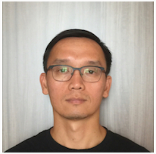
Milos

Ahmet

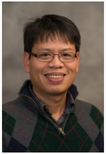
Fabian

relationalAI team, in particular:

Mahmoud



Hung



Long

ML: The Next Big Opportunity

ML is emerging as general purpose technology

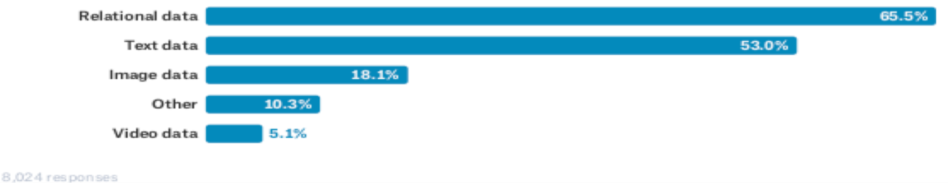
- Just as computing became general purpose 70 years ago

A core ability of intelligence is the ability to predict

- Turn information you have into information you need

The quality of the prediction is increasing as
the cost per prediction is decreasing

Most Enterprises Rely on Relational Data for Their ML Models



Retail: 86% relational

Insurance: 83% relational

Marketing: 82% relational

Financial: 77% relational

Source: The State of Data Science & Machine Learning 2017, Kaggle, October 2017
(based on 2017 Kaggle survey of 16,000 ML practitioners)

Relational Model: The Jewel in the Data Management Crown

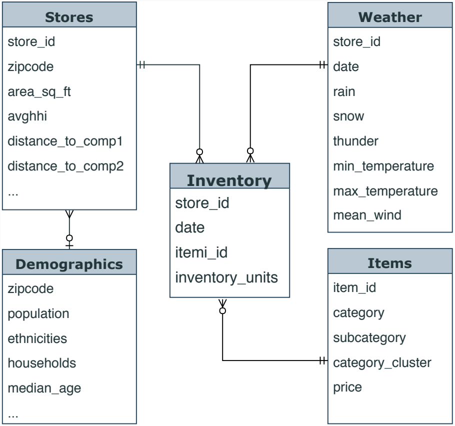
Last decades have witnessed massive adoption of the Relational Model

Many human hours invested in building relational models

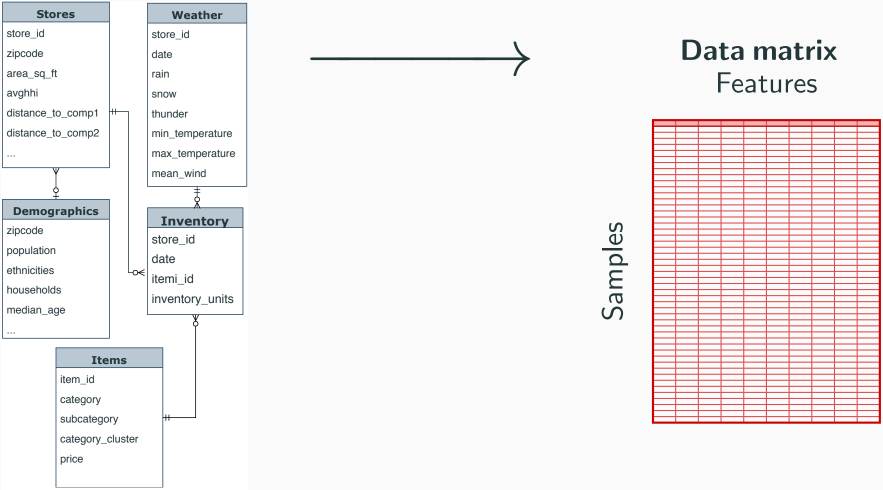
Relational databases are rich with knowledge of the underlying domains

Availability of curated data made it possible to learn from the past and to predict the future for both

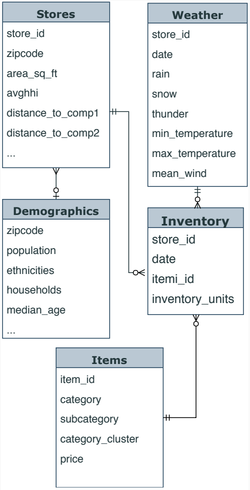
humans (BI) and machines (ML)



Current State of Affairs in Building Predictive Models



Current State of Affairs in Building Predictive Models



Current ML technology

THROWS AWAY

the relational structure

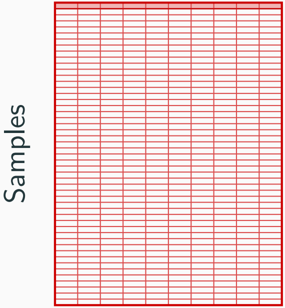
that can help build

FASTER

BETTER MODELS

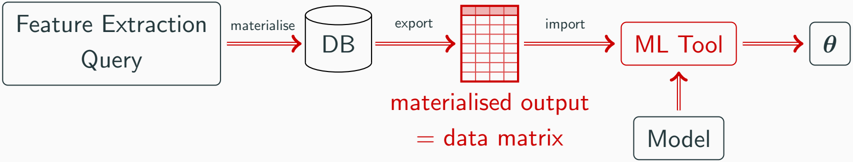
Data matrix

Features



Learning over Relational Databases: Revisit from First Principles

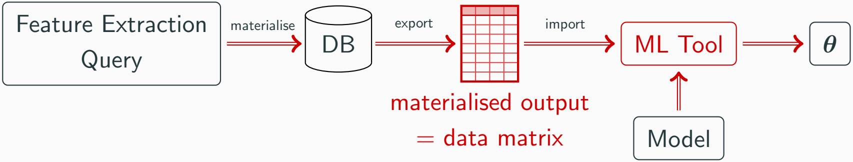
Structure-aware versus Structure-agnostic Learning



Structure-agnostic learning requires:

1. Materialisation of the query result
   (Recomputation in case of data updates)
2. Data export from DBMS and import into ML tool
3. One/multi-hot encoding of categorical variables

Structure-aware versus Structure-agnostic Learning

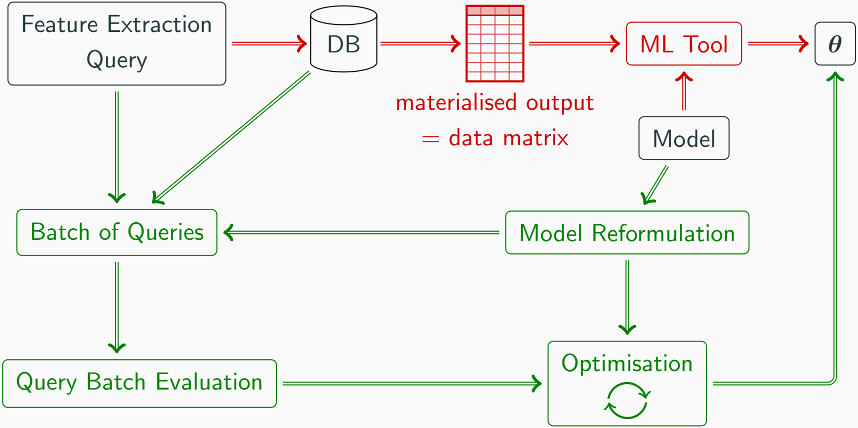


Structure-agnostic learning requires:

1. Materialisation of the query result
   (Recomputation in case of data updates)
2. Data export from DBMS and import into ML tool
3. One/multi-hot encoding of categorical variables

All these steps are very expensive and unnecessary!

Structure-aware versus Structure-agnostic Learning



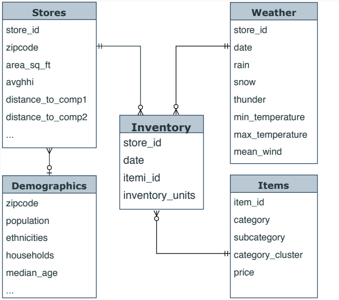
Structure-aware learning avoids the three expensive steps.

Structure-aware Learning

FASTER even than

Feature Extraction Query!

Example: A Retailer Use Case



Relation	Cardinality	Arity (Keys+Values)	File Size (CSV)
Inventory	84,055,817	3 + 1	2 GB
Items	5,618	1 + 4	129 KB
Stores	1,317	1 + 14	139 KB
Demographics	1,302	1 + 15	161 KB
Weather	1,159,457	2 + 6	33 MB
Join	84,055,817	3 + 41	23GB

Structure-aware versus Structure-agnostic Learning

Train a linear regression model to predict *inventory* given all features

	PostgreSQL+TensorFlow	
	Time	**Size (CSV)**
Database	–	2.1 GB
Join	152.06 secs	23 GB
Export	351.76 secs	23 GB
Shuffling	5,488.73 secs	23 GB
Query batch	–	–
Grad Descent	7,249.58 secs	–
Total time	13,242.13 secs	

Structure-aware versus Structure-agnostic Learning

Train a linear regression model to predict *inventory* given all features

	PostgreSQL+TensorFlow		Our approach (SIGMOD'19)	
	Time	**Size (CSV)**	**Time**	**Size (CSV)**
Database	–	2.1 GB	–	2.1 GB
Join	152.06 secs	23 GB	–	–
Export	351.76 secs	23 GB	–	–
Shuffling	5,488.73 secs	23 GB	–	–
Query batch	–	–	6.08 secs	37 KB
Grad Descent	7,249.58 secs	–	0.05 secs	–
Total time	13,242.13 secs		6.13 secs	

$2,160\times$ faster while $600\times$ more accurate (RMSE on 2% test data)

TensorFlow trains one model. Our approach takes < 0.1 sec for any extra model over a subset of the given feature set.

TensorFlow's Behaviour is the Rule, not the Exception!

Similar behaviour (or outright failure) for more:

- **datasets**: Favorita, TPC-DS, Yelp, Housing
- **systems**:
  - used in industry: R, scikit-learn, Python StatsModels, mlpack, XGBoost, MADlib
  - academic prototypes: Morpheus, libFM
- **models**: decision trees, factorization machines, k-means, ..

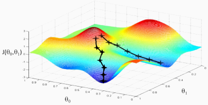
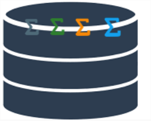
This is to be contrasted with the scalability of DBMSs!

How to Achieve This Performance Improvement?

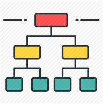
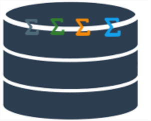
Idea 1: Turn the Learning Problem into a Database Problem

Through DB Glasses, Everything is a Batch of Queries

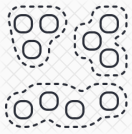
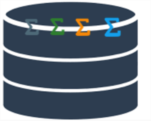
Gradient Computation $\longrightarrow$ Query Batch

Decision Tree Split Cost $\longrightarrow$ Query Batch

k-means Clustering Assignment $\longrightarrow$ Query Batch

Models under Consideration

So far:

- Polynomial regression
- Factorization machines
- Classification/regression trees
- Mutual information
- Chow Liu trees
- k-means clustering
- k-nearest neighbours
- PCA
- SVM

On-going:

- Boosting regression trees
- AdaBoost
- Sum-product networks
- Random forests
- Logistic regression
- Linear algebra:
  - QR decomposition
  - SVD
  - low-rank matrix factorisation

Models under Consideration

So far:

- Polynomial regression
- Factorization machines
- Classification/regression trees
- Mutual information
- Chow Liu trees
- k-means clustering
- k-nearest neighbours
- PCA
- SVM

On-going:

- Boosting regression trees
- AdaBoost
- Sum-product networks
- Random forests
- Logistic regression
- Linear algebra:
  - QR decomposition
  - SVD
  - low-rank matrix factorisation

All these cases can benefit from structure-aware computation

Example: Ridge Linear Regression

Linear function: $f_{\boldsymbol{\theta}}(\mathbf{x}) = \langle \boldsymbol{\theta}, \mathbf{x} \rangle = \theta_0 x_0 + \theta_1 x_1 + \dots$

- Training dataset D defined by feature extraction query and consists of tuples $(\mathbf{x}, y)$ of feature vector $\mathbf{x}$ and response y
- Parameters $\boldsymbol{\theta}$ obtained by minimising the objective function:

$$J(\boldsymbol{\theta}) = \overbrace{\frac{1}{2|D|} \sum_{(\mathbf{x},y)\in D} (\langle \boldsymbol{\theta}, \mathbf{x} \rangle - y)^2}^{\text{least square loss}} + \overbrace{\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2}^{\ell_2-\text{regulariser}}$$

From Optimisation to Query Batch

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:

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

Model reformulation idea: Decouple

- data-dependent ($\mathbf{x}$, y) computation from
- data-independent ($\boldsymbol{\theta}$) computation

in the formulations of the objective $J(\boldsymbol{\theta})$ and its gradient $\boldsymbol{\nabla} J(\boldsymbol{\theta})$.

From Optimisation to Query Batch

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

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

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

From Optimisation to Query Batch

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

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

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

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

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

Σ, **c**, s_Y can be Expressed as Batch of Queries

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

Σ, c, s_Y can be Expressed as Batch of Queries

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

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

where D is the feature extraction query over the input DB.

Σ, c, s_Y can be Expressed as Batch of Queries

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

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

- $\boldsymbol{x}_i$ categorical, x_j continuous

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

where D is the feature extraction query over the input DB.

Σ, c, s_Y can be Expressed as Batch of Queries

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

  SELECT SUM (x_i * x_j) FROM D;

- $\boldsymbol{x}_i$ categorical, x_j continuous

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

- $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ categorical

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

where D is the feature extraction query over the input DB.

Aggregation is the Aspirin to All Problems

Query batch sizes in practice:

Application/Dataset	**Retailer**	**Favorita**	**Yelp**	**TPC-DS**
Covariance Matrix	937	157	730	3,299
Decision Tree (one node)	3,150	273	1,392	4,299
k-means	44	19	38	92

Aggregation is the Aspirin to All Problems

Query batch sizes in practice:

Application/Dataset	Retailer	Favorita	Yelp	TPC-DS
Covariance Matrix	937	157	730	3,299
Decision Tree (one node)	3,150	273	1,392	4,299
k-means	44	19	38	92

Queries in a batch:

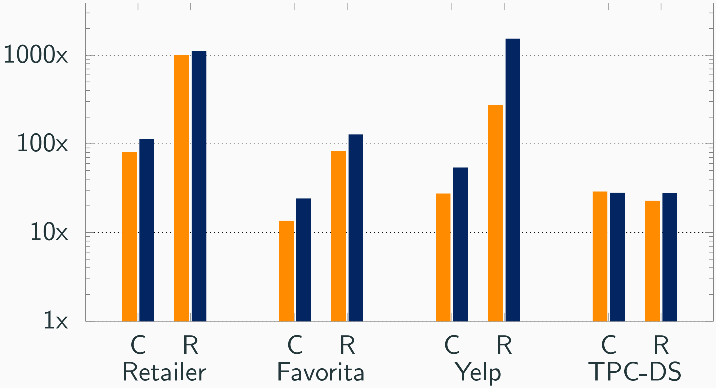
- Same aggregates but over different attributes
- Expressed over the same join of the database relations

AMPLE opportunities for sharing computation in a batch.

Natural Attempt:
Use Existing DB Technology
to Compute the Query Batch

Existing DBMSs are NOT Designed for Query Batches

Relative Speedup for **Our Approach** over **DBX** and **MonetDB**

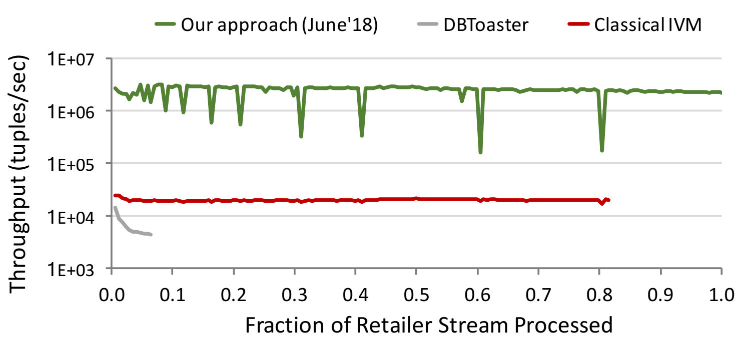


C = Covariance Matrix; R = Regression Tree Node; AWS d2.xlarge (4 vCPUs, 32GB)

Existing DSMSs are NOT Designed for Query Batches

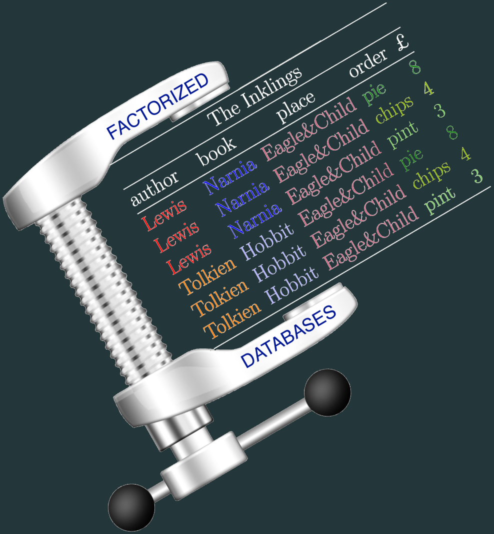
Task: Maintain the covariance matrix over Retailer

- Round-robin insertions in all relations
- All maintenance strategies implemented in DBToaster



Azure DS14, Intel Xeon, 2.40GHz, 112GB, 1 thread; one hour timeout

Idea 2: Exploit the Problem Structure to Lower the Complexity



Structure-aware Tools of a Database Researcher

Algebraic structure: (semi)rings $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$

- Distributivity law $\rightarrow$ Factorisation

  Factorised Databases [VLDB'12+'13]

  Factorised Machine Learning

  [SIGMOD'16,VLDB'16,SIGREC'16,DEEM'18,PODS'18+'19]

Structure-aware Tools of a Database Researcher

Algebraic structure: (semi)rings $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$

- Distributivity law $\rightarrow$ Factorisation

  Factorised Databases [VLDB'12+'13]

  Factorised Machine Learning

  [SIGMOD'16,VLDB'16,SIGREC'16,DEEM'18,PODS'18+'19]

- Additive inverse $\rightarrow$ Uniform treatment of updates

  Factorised Incremental Maintenance [SIGMOD'18]

Structure-aware Tools of a Database Researcher

Algebraic structure: (semi)rings $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$

- Distributivity law $\rightarrow$ Factorisation

  Factorised Databases [VLDB'12+'13]

  Factorised Machine Learning

  [SIGMOD'16,VLDB'16,SIGREC'16,DEEM'18,PODS'18+'19]

- Additive inverse $\rightarrow$ Uniform treatment of updates

  Factorised Incremental Maintenance [SIGMOD'18]

- Sum-Product abstraction $\rightarrow$ Same processing for distinct tasks

  DB queries, Covariance matrix, PGM inference, Matrix chain multiplication

  [SIGMOD'18]

Structure-aware Tools of a Database Researcher

Combinatorial structure: query width and data degree measures

- Width measure w for FEQ $\rightarrow$ Low complexity $\tilde{O}(N^w)$

  factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  worst-case optimal size and time for factorised joins

  [ICDT'12+'18,TODS'15,PODS'19]

Combinatorial structure: query width and data degree measures

- Width measure w for FEQ $\rightarrow$ Low complexity $\tilde{O}(N^w)$

  factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  worst-case optimal size and time for factorised joins

  [ICDT'12+'18,TODS'15,PODS'19]

- Degree $\rightarrow$ Adaptive processing depending on high/low degrees

  worst-case optimal incremental maintenance for triangle count [ICDT'19a]

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

Structure-aware Tools of a Database Researcher

Combinatorial structure: query width and data degree measures

- Width measure w for FEQ $\rightarrow$ Low complexity $\tilde{O}(N^w)$

  factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  worst-case optimal size and time for factorised joins

  [ICDT'12+'18,TODS'15,PODS'19]

- Degree $\rightarrow$ Adaptive processing depending on high/low degrees

  worst-case optimal incremental maintenance for triangle count [ICDT'19a]

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

- Functional dependencies $\rightarrow$ Learn simpler, equivalent models

  reparameterisation of polynomial regression models and factorisation machines

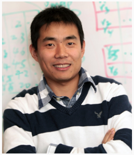
  [PODS'18,TODS'19]

Structure-aware Tools of a Database Researcher

Statistical structure: sampling for joins and models, coresets

- Sampling through joins: ripple/wander joins [SIGMOD'99+'16]






- Sampling specific to classes of models [SIGMOD'19]

- Succinct approximate data representations

  coresets for k-means with constant-factor approximation [submission'19]

Case in Point (1):

Factorised Query Computation

$\Rightarrow$

Exponential Time/Size Improvement

Example: Factorised Query Computation

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

Example: Factorised Query Computation

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

Example: Factorised Query Computation

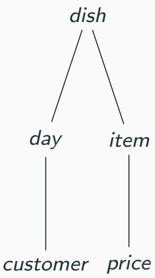
O(customer, day, dish), D(dish, item), I(item, price)

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

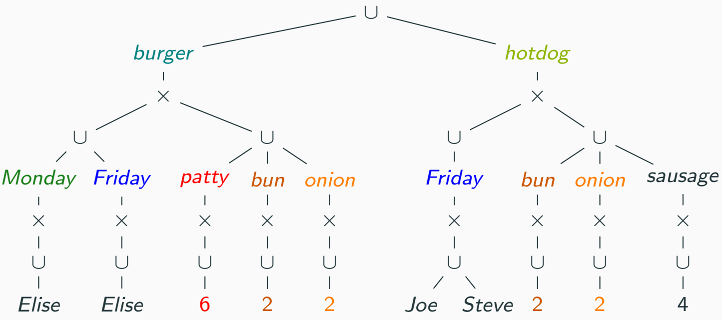
An algebraic encoding uses product ($\times$), union ($\cup$), and values:

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

Factorised Join



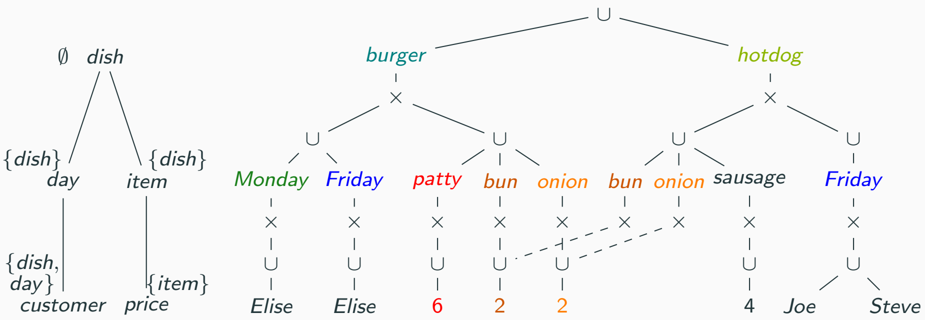
Variable order



Instantiation of the variable order over the input database

There are several algebraically equivalent factorised joins defined by distributivity of product over union and their commutativity.

.. Now with Further Compression

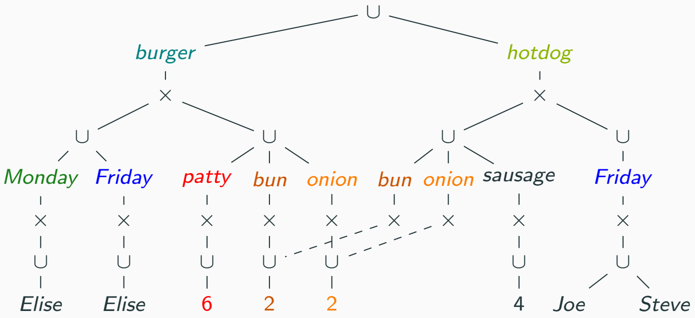


Observation:

- `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!

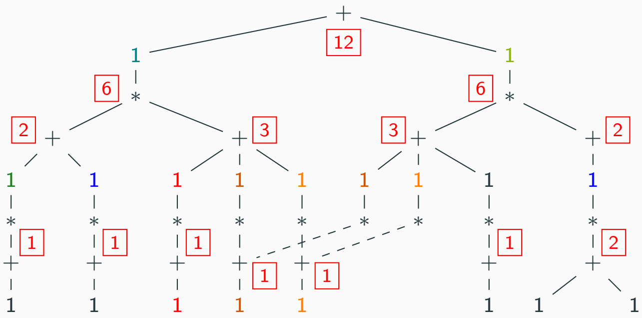
Factorising the Computation of Aggregates (1/2)



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

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

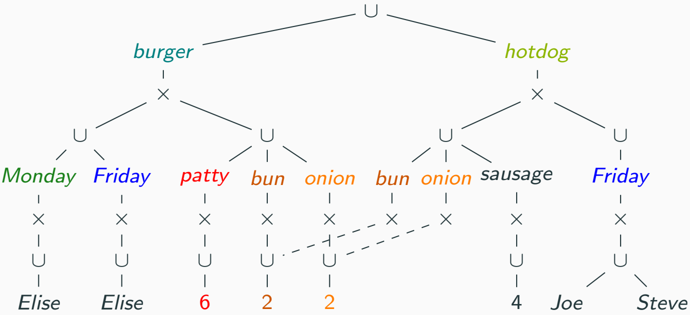
Factorising the Computation of Aggregates (1/2)



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

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

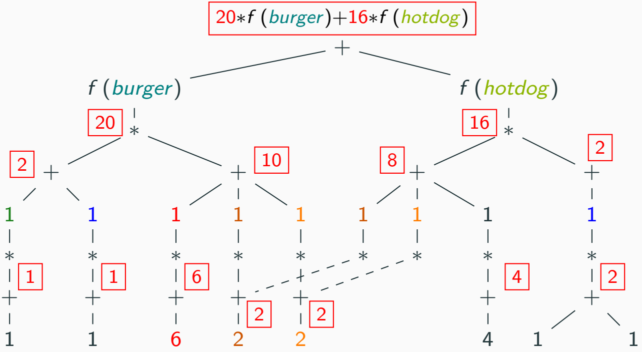
Factorising the Computation of Aggregates (2/2)



`SUM(dish * price)` computed in one pass over the factorisation:

- Assume there is a function f that turns `dish` into numbers.
- All values except for `dish` & `price` $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

Factorising the Computation of Aggregates (2/2)



SUM(dish * price) computed in one pass over the factorisation:

- Assume there is a function f that turns dish into numbers.
- All values except for dish & price $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

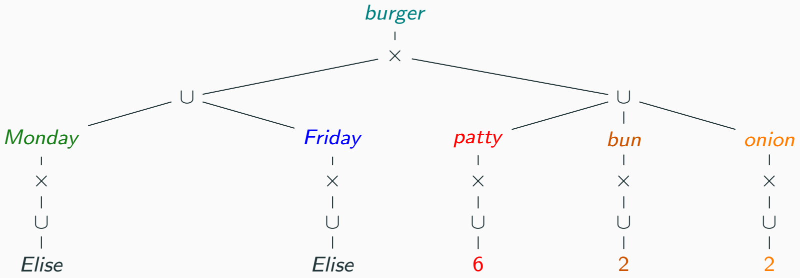
Case in Point (2):

Sum-Product Ring Abstraction

$\Rightarrow$

Sharing Aggregate Computation

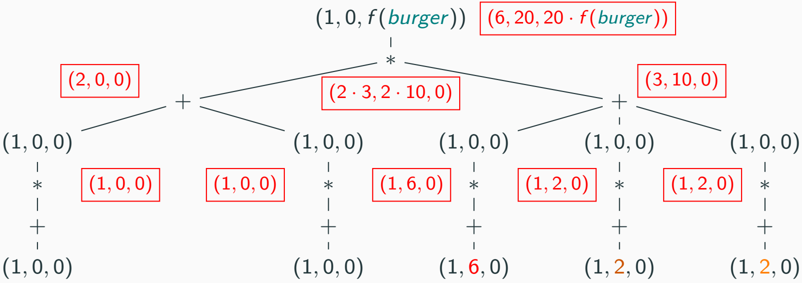
Shared Computation of Several Aggregates (1/2)



Ring for computing `SUM(1)`, `SUM(price)`, `SUM(price * dish)`:

- Elements = triples of numbers, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates

Shared Computation of Several Aggregates (2/2)

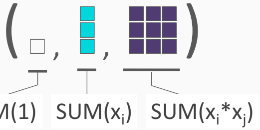


Ring for computing `SUM(1), SUM(price), SUM(price * dish)`:

- Elements = triples of numbers, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates

Ring Generalisation for the Entire Covariance Matrix

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



$$(c_1, \mathbf{s}_1, \mathbf{Q}_1) + (c_2, \mathbf{s}_2, \mathbf{Q}_2) = (c_1 + c_2, \mathbf{s}_1 + \mathbf{s}_2, \mathbf{Q}_1 + \mathbf{Q}_2)$$

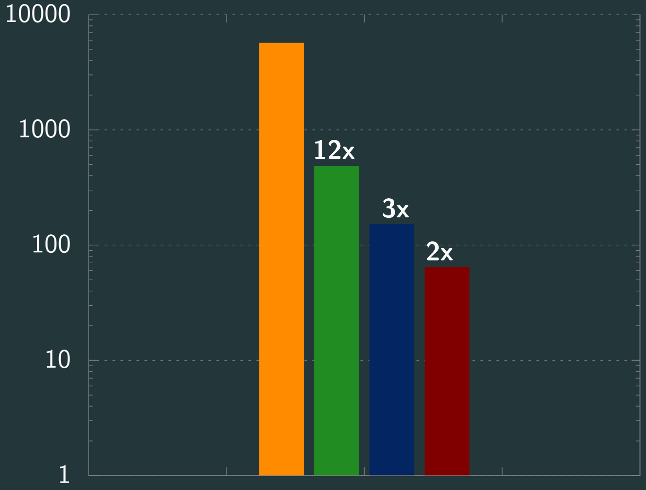
$$(c_1, \mathbf{s}_1, \mathbf{Q}_1) * (c_2, \mathbf{s}_2, \mathbf{Q}_2) = (c_1 \cdot c_2, c_2 \cdot \mathbf{s}_1 + c_1 \cdot \mathbf{s}_2, c_2 \cdot \mathbf{Q}_1 + c_1 \cdot \mathbf{Q}_2 + \mathbf{s}_1 \mathbf{s}_2^T + \mathbf{s}_2 \mathbf{s}_1^T)$$

$$\mathbf{0} = (0, \mathbf{0}_{n \times 1}, \mathbf{0}_{n \times n})$$

$$\mathbf{1} = (1, \mathbf{0}_{n \times 1}, \mathbf{0}_{n \times n})$$

- SUM(1) reused for all SUM(x_i) and SUM$(x_i * x_j)$
- SUM(x_i) reused for all SUM$(x_i * x_j)$

Idea 3: Lower the Constant Factors



Engineering Tools of a Database Researcher

1. Specialisation for workload and data

   Generate code specific to the query batch and dataset

   Improve cache locality for hot data path

2. Sharing low-level data access

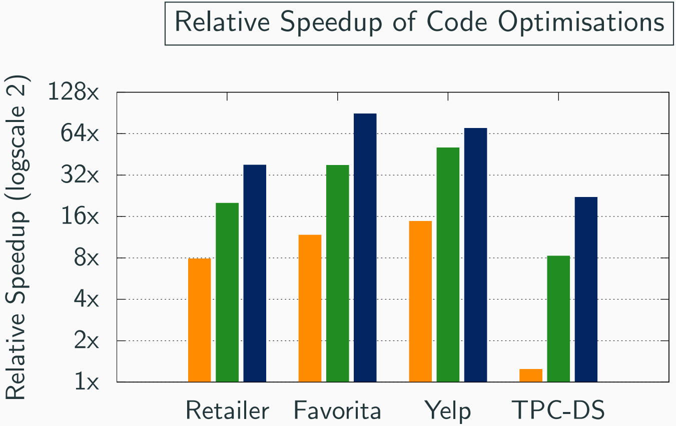
   Aggregates decomposed into views over join tree
   Share data access across views with different output schemas

3. Parallelisation: multi-core (SIMD & distribution to come)

   Task and domain parallelism

[DEEM'18,SIGMOD'19]

Engineering Tools of a Database Researcher



Added optimisations for covariance matrix computation:

specialisation → + sharing → + parallelization

AWS d2.xlarge (4 vCPUs, 32GB)

Case in Point (3):

Code Optimisations

$\Rightarrow$

Non-trivial Speedup

Sharing Computation for a Query Batch in Favorita

Sales: date, store, item, units, promo

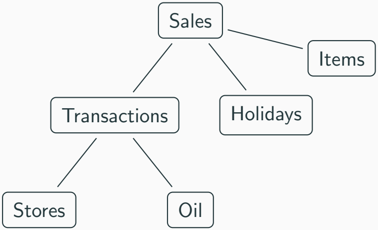
Holidays: date, htype, locale, transferred

Stores: store, city, state, stype, cluster

Items: item, family, class, perishable

Transactions: date, store, txns

Oil: date, price



Sharing Computation for a Query Batch in Favorita

Sales: date, store, item, units, promo

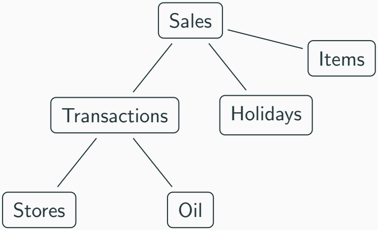
Holidays: date, htype, locale, transferred

Stores: store, city, state, stype, cluster

Items: item, family, class, perishable

Transactions: date, store, txns

Oil: date, price



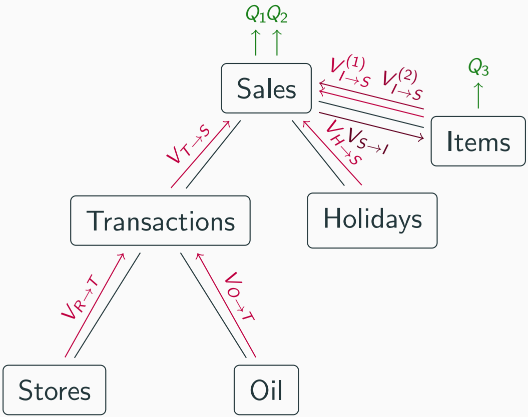
Aggregates to compute over the join of relations:

Q_1: $\texttt{SUM}\ (f(\texttt{units}))$

Q_2: $\texttt{SUM}\ (g(\texttt{item}) \cdot h(\mathit{date}, \mathit{family}))$ GROUP BY $\texttt{store}$

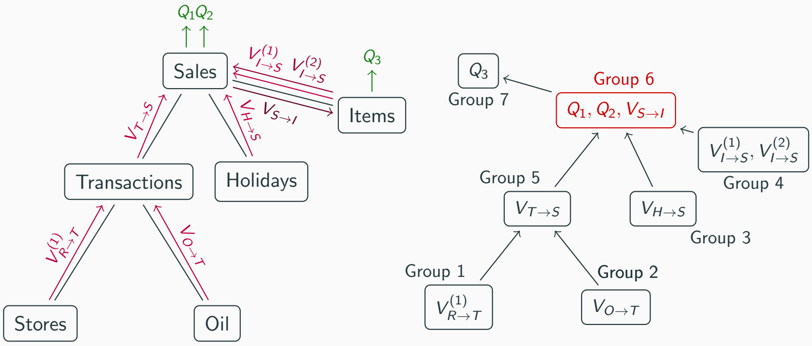
Q_3: $\texttt{SUM}\ (f(\texttt{units}) \cdot g(\texttt{item}))$ GROUP BY $\texttt{family}$

Shared Execution Plan for a Query Batch



- For each query, decide its output (root) node in the join tree
- Break down each query into directional views over the join tree
- Merge/share/group views for different queries
- Computational unit: group of views

Parallelisation: Dependency Graph of View Groups



- Task parallelism: Evaluate independent groups in parallel
- Domain parallelism: Partition the large relation used for each group

Code Generation for Executing View Group 6 over Sales

Traverse Sales as a trie following an order of its join variables

Code Generation for Executing View Group 6 over Sales

$V_I^{(1)} \to$ item
$V_I^{(2)} \to$ item

$V_H \to$ date

$V_T \to$ store

foreach $i \in \pi_{\text{item}}(S \ltimes_{\text{item}} V_I^{(1)} \ltimes_{\text{item}} V_I^{(2)})$

foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \ltimes_{\text{date}} V_H \ltimes_{\text{date}} V_T)$

foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \ltimes_{\text{store}} \sigma_{\text{date}=d} V_T)$

Lookup into incoming views, e.g., $V_I^{(1)}$, as early as possible

Code Generation for Executing View Group 6 over Sales

$V_I^{(1)} \to$ item
$V_I^{(2)} \to$ item

$V_H \to$ date

$V_T \to$ store

$\alpha_4 = 0;$
foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I^{(1)} \bowtie_{\text{item}} V_I^{(2)})$
  $\alpha_I^{(1)} = V_I^{(1)}(i)$
  $\alpha_3 = 0;$
  foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$
    $\alpha_H = V_H(d);$
    $\alpha_2 = 0;$
    foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$
      $\alpha_T = V_T(d, s); \quad \alpha_1 = 0;$
      foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S : \alpha_1 \mathrel{+}= f(u);$
      $\alpha_2 \mathrel{+}= \alpha_1 \cdot \alpha_T;$
    $\alpha_3 \mathrel{+}= \alpha_2 \cdot \alpha_H;$
  $\alpha_4 \mathrel{+}= \alpha_3 \cdot \alpha_I^{(1)}$
$Q_1 = \alpha_4;$

Insert code for partial aggregates as early as possible

Code Generation for Executing View Group 6 over Sales

$V_I^{(1)} \rightarrow$ item
$V_I^{(2)} \rightarrow$ item

$V_H \rightarrow$ date

$V_T \rightarrow$ store

$\alpha_4 = 0;$
foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I^{(1)} \bowtie_{\text{item}} V_I^{(2)})$
  $\alpha_I^{(1)} = V_I^{(1)}(i);$
  $\alpha_3 = 0;$
  foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$
    $\alpha_H = V_H(d);$
    $\alpha_2 = 0;$
    foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$
      $\alpha_T = V_T(d, s); \quad \alpha_1 = 0;$
      foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S : \alpha_1 \mathrel{+}= f(u);$
      $\alpha_2 \mathrel{+}= \alpha_1 \cdot \alpha_T;$
    $\alpha_3 \mathrel{+}= \alpha_2 \cdot \alpha_H;$
  $\alpha_4 \mathrel{+}= \alpha_3 \cdot \alpha_I^{(1)};$
$Q_1 = \alpha_4;$

Q_1: `SUM` $(f($`units`$))$

Code Generation for Executing View Group 6 over Sales

$V_I^{(1)} \rightarrow$, $V_I^{(2)} \rightarrow$ item

$V_H \rightarrow$ date

$V_T \rightarrow$ store

$\alpha_4 = 0;$

foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I^{(1)} \bowtie_{\text{item}} V_I^{(2)})$

$\quad \alpha_I^{(1)} = V_I^{(1)}(i);$

$\quad \alpha_5 = g(i);$

$\quad \alpha_3 = 0;$

$\quad$ foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$

$\quad\quad \alpha_H = V_H(d);$

$\quad\quad \alpha_2 = 0;$

$\quad\quad$ foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$

$\quad\quad\quad \alpha_T = V_T(d, s); \quad \alpha_1 = 0;$

$\quad\quad\quad$ foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S : \alpha_1 \mathrel{+}= f(u);$

$\quad\quad\quad \alpha_2 \mathrel{+}= \alpha_1 \cdot \alpha_T;$

$\quad\quad \alpha_3 \mathrel{+}= \alpha_2 \cdot \alpha_H;$

$\quad \alpha_4 \mathrel{+}= \alpha_3 \cdot \alpha_I^{(1)}; \quad V_{S\rightarrow I}(i) = \alpha_3 \cdot \alpha_5;$

$Q_1 = \alpha_4;$

$V_{S\rightarrow I}$: `SUM` $(f(\texttt{units}) \cdot g(\texttt{item}))$ `GROUP BY item`

Code Generation for Executing View Group 6 over Sales

$$\begin{array}{l} V_I^{(1)} \to \text{item} \\ V_I^{(2)} \to \text{item} \\ \quad | \\ V_H \to \text{date} \\ \quad | \\ V_T \to \text{store} \end{array}$$

$\alpha_4 = 0;$

foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I^{(1)} \bowtie_{\text{item}} V_I^{(2)})$

- $\alpha_I^{(1)} = V_I^{(1)}(i);$
- $\alpha_5 = g(i);$
- $\alpha_3 = 0;$
- foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$
  - $\alpha_H = V_H(d);\quad \alpha_6 = 0;$
  - foreach $y \in \pi_{\text{family}} \sigma_{\text{item}=i} V_I^{(2)}$: $\alpha_6 \mathrel{+}= h(d, y) \cdot V_I^{(2)}(i, y);$
  - $\alpha_2 = 0;\quad \alpha_7 = \alpha_6 \cdot \alpha_5 \cdot \alpha_H;$
  - foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$
    - $\alpha_T = V_T(d, s);\quad \alpha_1 = 0;\quad \alpha_8 = |\sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S|;$
    - foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S$: $\alpha_1 \mathrel{+}= f(u);$
    - $\alpha_2 \mathrel{+}= \alpha_1 \cdot \alpha_T;$
    - if $Q_2(s)$ then $Q_2(s) \mathrel{+}= \alpha_7 \cdot \alpha_8 \cdot \alpha_T$ else $Q_2(s) = \alpha_7 \cdot \alpha_8 \cdot \alpha_T;$
  - $\alpha_3 \mathrel{+}= \alpha_2 \cdot \alpha_H;$
- $\alpha_4 \mathrel{+}= \alpha_3 \cdot \alpha_I^{(1)};\quad V_{S \to I}(i) = \alpha_3 \cdot \alpha_5;$

$Q_1 = \alpha_4;$

Q_2: `SUM` $(g(\texttt{item}) \cdot h(\mathit{date}, \mathit{family}))$ `GROUP BY store`

Conclusion

Three-step recipe for efficient machine learning over databases:

1. Turn the learning problem into a database problem
2. Exploit the problem structure to lower the complexity
3. Specialise and optimise the code to lower the constant factors

Q.E.D.

Call to arms

We need more sustained work on

theory and systems for

Structure-aware Approaches to Data Analytics

References

References i

[ICDT'12] Dan Olteanu and Jakub Závodný. *Factorised Representations of Query Results: Size Bounds and Readability.* In ICDT 2012

[VLDB'12] Nurzhan Bakibayev, Dan Olteanu, and Jakub Závodný. *FDB: A Query Engine for Factorised Relational Databases.* In VLDB 2012

[VLDB'13] Nurzhan Bakibayev, Tomáš Kočiský , Dan Olteanu, and Jakub Závodný. *Aggregation and Ordering in Factorised Databases.* In VLDB 2013

[TODS'15] Dan Olteanu and Jakub Závodný. *Size Bounds for Factorised Representations of Query Results.* In TODS Vol. 40(1):2 2015

[SIGMOD'16] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. *Learning Linear Regression Models over Factorized Joins.* In SIGMOD 2016

[SIGREC'16] Dan Olteanu and Maximilian Schleich. *Factorized Databases.* In SIGMOD Record Vol 45 (2) 2016

References ii

[VLDB'16] Dan Olteanu and Maximilian Schleich. *F: Regression Models over Factorized Views.* In VLDB 2016

[DEEM'18] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. *AC/DC: In-Database Learning Thunderstruck.* In DEEM@SIGMOD 2018

[ICDT'18] Ahmet Kara and Dan Olteanu. *Covers of Query Results.* In ICDT 2018

[PODS'18] Mahmoud Abo Khamis, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. *In-Database Learning with Sparse Tensors.* In PODS 2018

[SIGMOD'18] Milos Nikolic and Dan Olteanu. *Incremental View Maintenance with Triple Lock Factorisation Benefits.* In SIGMOD 2018

[ICDT'19a] Ahmet Kara, Hung Q. Ngo, Miklos Nikolic, Dan Olteanu and Haozhe Zhang. *Counting Triangles under Updates in Worst-Case Optimal Time.* In ICDT 2019

[ICDT'19b] Mahmoud Abo Khamis, Hung Ngo, Dan Olteanu and Dan Suciu. *Boolean Tensor Decomposition for Conjunctive Queries with Negation.* In ICDT 2019

[PODS'19] Mahmoud Abo Khamis, Ryan Curtin, Ben Moseley, Hung Ngo, Long Nguyen, Dan Olteanu, and Maximilian Schleich. *On Functional Aggregate Queries with Additive Inequalities.* In PODS 2019

[SIGMOD'19] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Ngo, and Long Nguyen. *A Layered Aggregate Engine for Analytics Workloads.* In SIGMOD 2019