Towards end to end in db data science

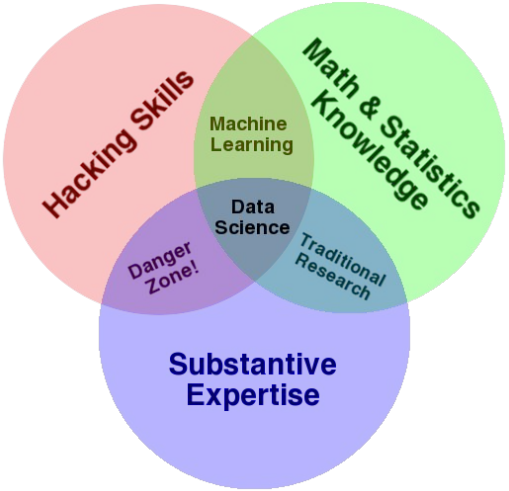
Nikolaos Vasiloglou

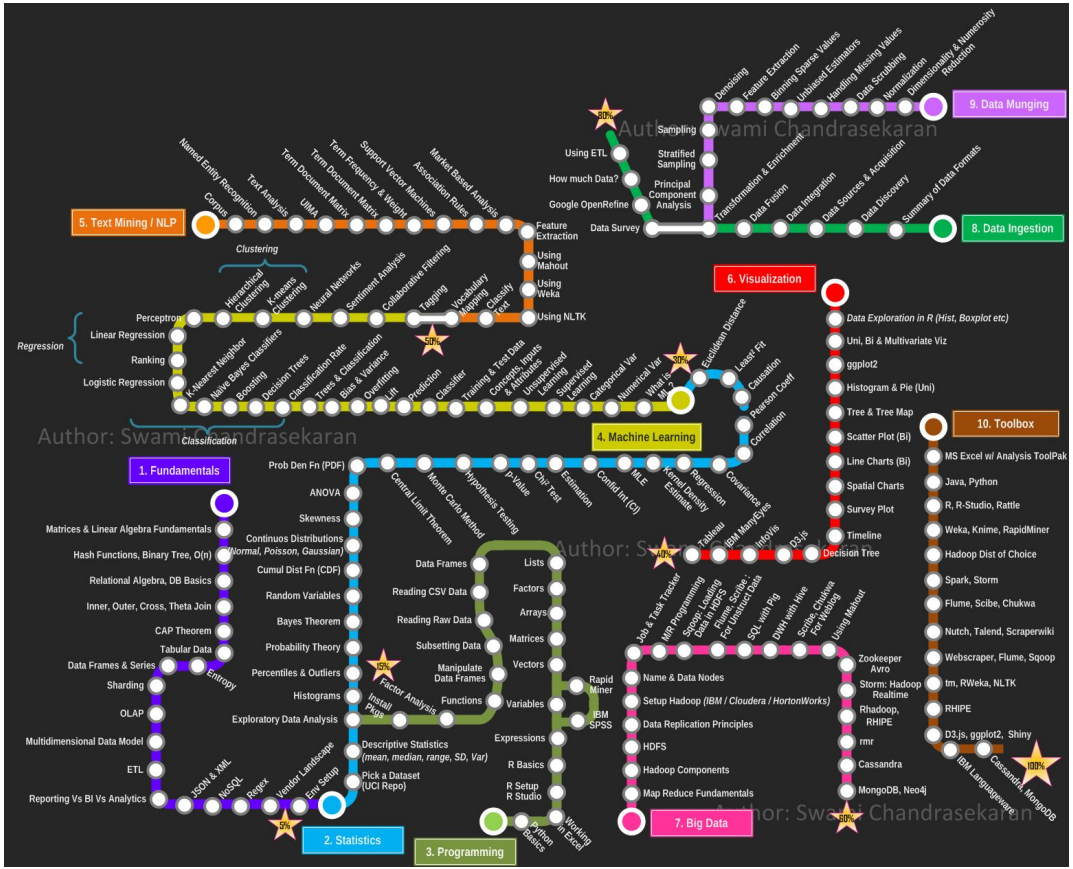
Outline

- The basic components of Data science
- Why declarative?
- Why relational?
- The current status of data science
- Fixing the problems
- Interfacing solvers to databases
- Building Solvers in database

The basic ingredients of data science

- Data transformations
- Machine Learning
- Deployment
- Model analysis





The excel syndrome

- Back in the 80/90s people suffered from the Excel Syndrome

I once worked for a manager who suffered from the Excel syndrome. Despite knowing what a database was (the company was one he created and MD of, and was in the field of IT support, so one would hope that was the case at least as most of the clients had SBS systems that used MS SQL Server) he insisted on the mailing "database" to be managed in spreadsheets. This was back when Microsoft though we'd only ever need a maximum of 65535 rows in a spreadsheet, and the mailing list was often much larger than that. So what was the solution? Well, when you hit the limit, just create a new spreadhseet of course (note that he didn't actually want to use separate sheets within the one file, each spreadsheet was its own file on the server)

Oh and of course the list had to be alphabetically organised (by some column I forget now) so that when a new entry had to be entered into a spreadsheet that was already full, entry 65536 and above had to be popped into the next file in the series, repeated as necessary if that was full too, etc.

I did suggest using a proper database for this but he was of the mind that it wouldn't be simple enough for the non-technical secretarial staff to handle. That manager was probably one of the sources of the most IT WTFery I've seen in years, made more funny by the fact that he was supposedly technically adept and the company was responsible for the IT support of a lot of businesses in the area.

Data Frame the new excel syndrome

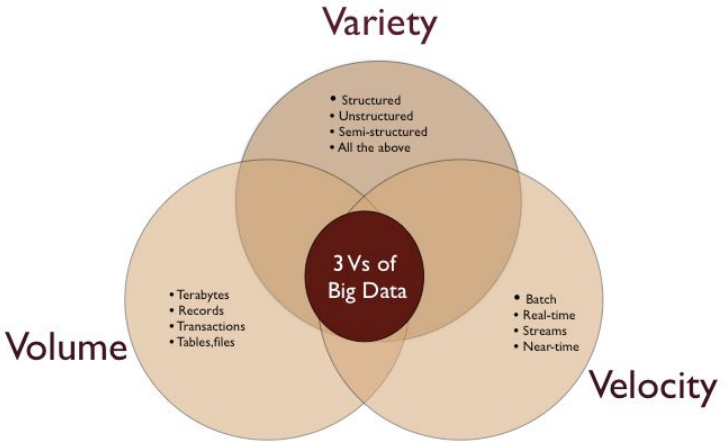
	account	campaign	date	successes	trials	rate
455	1	Campaign #76	2012-08-14 11:56:20 -0400	2	2	1.000000
449	1	Campaign #78	2012-08-14 12:06:20 -0400	2	2	1.000000
438	1	Campaign #87	2012-08-14 18:06:30 -0400			
431	1	Campaign #95	2012-08-15 00:07:42 -0400			
422	1	Campaign #99	2012-08-15 01:27:48 -0400			

Data: result.df

	bib	category	swim_cat	swim_ov	swim_time	TT1	bike_cat	bike_ov	bike_time	TT2	run_cat	run_ov	run_time	overall_cat	overall_ov	overall_time
1	71	M 35-39	2	30	00:12:07	00:03:18	1	4	00:28:09	00:00:36	1	2	00:20:06	1	1	01:04:18
2	1	M 40-44	4	23	00:11:46	00:04:00	1	1	00:27:03	00:00:36	1	5	00:21:08	1	2	01:04:36
3	2	M 45-49	1	3	00:10:06	00:03:36	2	5	00:28:40	00:00:45	1	12	00:22:45	1	3	01:05:54
4	39	M 25-29	5	51	00:12:55	00:03:11	1	8	00:29:17	00:00:43	1	1	00:19:57	1	4	01:06:04
5	111	M 40-44	3	11	00:11:07	00:04:04	3	6	00:28:48	00:00:52	2	9	00:22:19	2	5	01:07:11
6	120	M 45-49	9	57	00:13:08	00:03:51	1	3	00:27:50	00:00:39	3	18	00:23:24	2	6	01:08:55
7	114	M 40-44	7	45	00:12:36	00:03:47	4	10	00:29:26	00:00:46	3	10	00:22:21	3	7	01:08:57
8	17	M 20-24	2	12	00:11:08	00:03:48	1	28	00:31:00	00:00:35	1	15	00:23:02	1	8	01:09:35
9	3	M 45-49	2	5	00:10:35	00:04:27	5	18	00:30:05	00:00:32	6	28	00:24:11	3	9	01:09:52
10	57	M 30-34	2	34	00:12:17	00:04:27	1	11	00:29:32	00:00:44	4	16	00:23:03	1	10	01:10:05
11	232	F 35-39	2	21	00:11:33	00:04:21	1	9	00:29:20	00:00:38	2	31	00:24:20	1	11	01:10:14
12	98	M 40-44	5	40	00:12:29	00:03:37	7	22	00:30:44	00:00:32	4	14	00:23:01	4	12	01:10:24
13	303	M 30-34	5	106	00:14:34	00:03:43	2	29	00:31:06	00:00:51	1	3	00:20:09	2	13	01:10:25
14	321	M 25-29	2	17	00:11:21	00:04:53	2	23	00:30:46	00:01:12	5	11	00:22:37	2	14	01:10:52
15	24	M 25-29	7	90	00:14:10	00:03:42	3	30	00:31:07	00:00:41	3	6	00:21:36	3	15	01:11:18
16	30	M 25-29	1	7	00:10:48	00:05:35	5	46	00:32:09	00:01:27	4	7	00:21:39	4	16	01:11:39
17	144	M 50-54	6	28	00:12:00	00:05:19	1	15	00:29:44	00:00:59	1	20	00:23:40	1	17	01:11:43
18	227	F 35-39	4	46	00:12:38	00:03:53	2	27	00:30:54	00:00:35	1	22	00:23:46	2	18	01:11:47
19	118	M 45-49	6	49	00:12:45	00:04:23	8	25	00:30:50	00:00:50	2	17	00:23:07	4	19	01:11:56
20	135	M 45-49	4	27	00:11:56	00:05:03	6	19	00:30:18	00:00:38	5	24	00:24:00	5	20	01:11:58
21	313	M 40-44	6	42	00:12:32	00:04:32	9	33	00:31:12	00:01:02	5	21	00:23:46	5	21	01:13:06

Pros

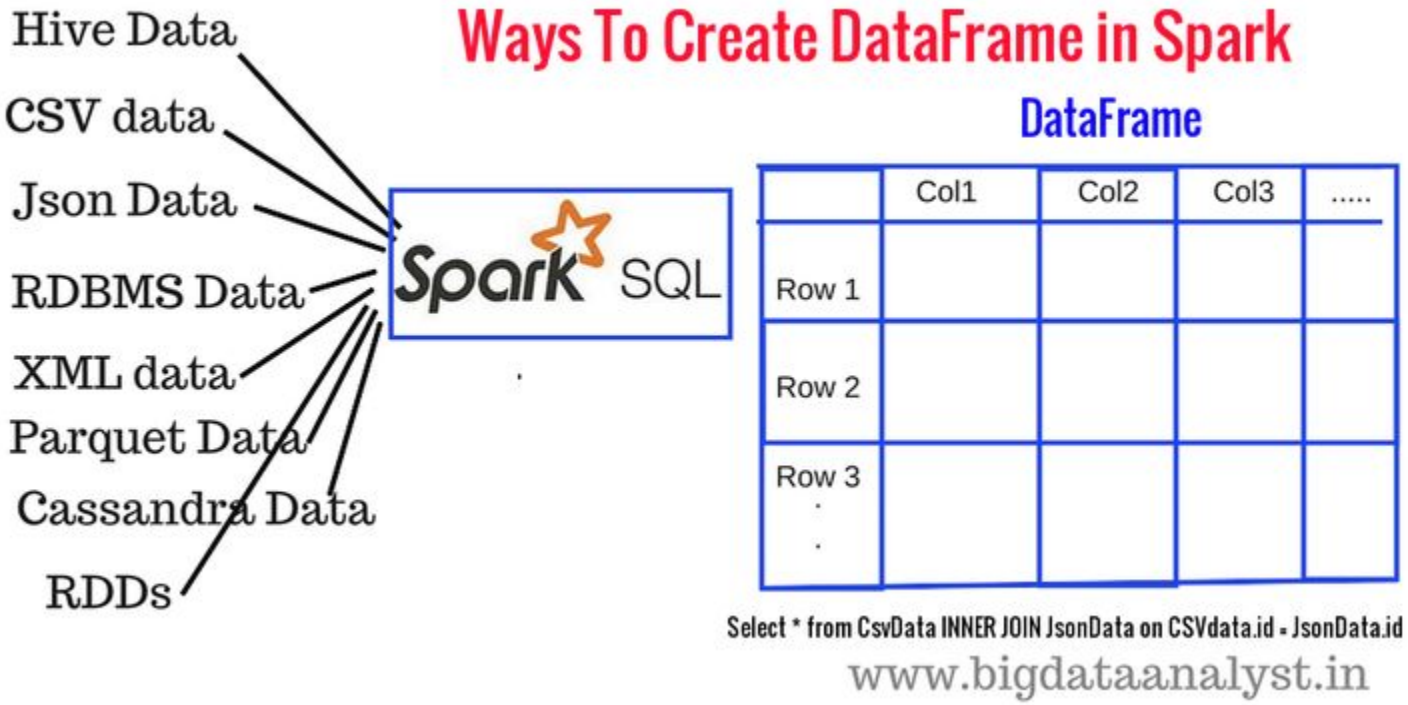
- Easy view of the data
- Easy filtering
- Easy selection of columns
- Easy conversion to arrays
- Data inspection is one of the most common tasks in datascience



Cons

- Real life queries are much more complex
- They make small data look very big
- Adds another level of complexity on the systems architecture

Why do databases want to look like dataframes?



But maybe ML should want to look like a database

Databases are inherently declarative

So we need a declarative platform for machine learning

Declarative vs Imperative Machine Learning

Imperative Machine Learning

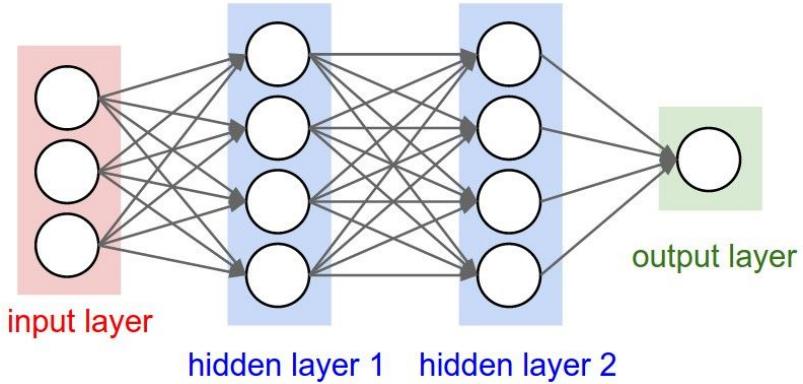
- Step by step description of the learning algorithm implementation.
- Explicit memory management by scientist
- Resource management by scientist
- Order matters!!!!

What is problematic about imperative programming?

- Performance optimization and semantics are entangled
- It is very hard to add new components on an algorithm
- What can these components be?

Declarative Machine Learning abstractions

- Neural Networks
- Convex optimization
- Probabilistic Programming



Hello Uncertain World

```
string A = random new Uniform<string>();
string B = random new Uniform<string>();
string C = A+" "+B;
constrain(C == "Hello Uncertain World");

infer(A)
// 50%: "Hello", 50%: "Hello Uncertain"
infer(B)
// 50%: "Uncertain World", 50%: "World"
```

Operators

- Objective
- Constraints
- Generators
  - Linear Algebra
  - Sample Generators
- Gradients

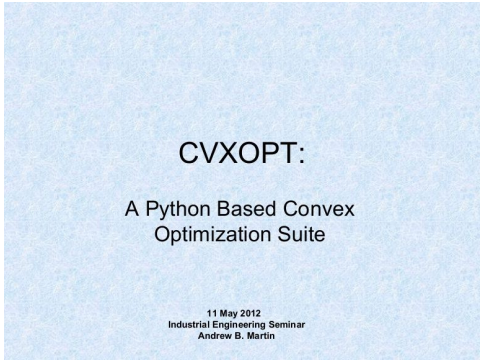
Declarative Platforms

TensorFlow

PYTORCH

Microsoft CNTK

mxnet



GUROBI OPTIMIZATION

Did machine learning become easier?

- YES

```
def MLP(inputs):
    W_1 = tf.Variable(tf.random_normal([784, 256]))
    b_1 = tf.Variable(tf.zeros([256]))

    W_2 = tf.Variable(tf.random_normal([256, 256]))
    b_2 = tf.Variable(tf.zeros([256]))

    W_out = tf.Variable(tf.random_normal([256, 10]))
    b_out = tf.Variable(tf.zeros([10]))

    h_1 = tf.add(tf.matmul(inputs, W_1), b_1)
    h_1 = tf.nn.relu(h_1)

    h_2 = tf.add(tf.matmul(h_1, W_2), b_2)
    h_2 = tf.nn.relu(h_2)

    out = tf.add(tf.matmul(h_2, W_out), b_out)

    return out

net = MLP(x)

# define loss and optimizer
loss_op = tf.reduce_mean(
              tf.nn.softmax_cross_entropy_with_logits(net, y))
opt     = tf.train.AdamOptimizer(learning_rate).minimize(loss_op)
```

Did data science become easier?

No, we still need a data frame

The MLP code for a retail problem that has data in 10 different tables is about 10x bigger and 100x slower

What did we miss here?

Observation

Denormalizing relational data is not wise. …

“We need a machine learning system that works on the relational domain”

Attempts to do machine learning in the database

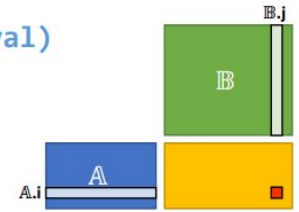
The UDF approach

- SIGMOD 2017 Tutorial Data Management in Machine Learning

Matrix Multiply: Take 1

- **Data:** A(i,j,val), B(i,j,val)
  - Basically a sparse representation
- SELECT A.i, B.j, SUM(A.val*B.val)
  FROM A, B
  WHERE A.j = B.i
  GROUP BY A.i, B.j;

MAD Skills [VLDB'09]



- **Works pretty well for sparse matrices**
- **Not so good for dense matrices, but still beats "small-data" platforms when data doesn't fit in memory**

The factorized approach

- SIGMOD 2017 Tutorial Data Management in Machine Learning

Overview: Learning Over Joins

Problem: Many datasets are multi-table ⟷ ML toolkits assume single-table inputs → ML after joining tables

Overheads:
Extra storage
Computational redundancy
Join time
Maintenance headaches

Learning Over Joins: "Push Down" ML through joins

1) Over standard data systems: Orion, Santoku, Morpheus
2) Over a "factorized database" system: FDB-F
3) Special-purpose tools: libFM, TensorDB, Compressed ML

Why did they all fail?

- More or less they are a systems integration (UDF approach) and not an algorithmic integration
- The operators were limited
- Difficult to add new operators
- Data operations and ML require a different language

Expressing data relations and ML algorithms in the same language

And the language is …

Expressing and computing data queries

Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm

Todd L. Veldhuizen
LogicBlox Inc.
Two Midtown Plaza
1349 West Peachtree Street NW
Suite 1880, Atlanta GA 30309
tveldhui@{logicblox.com,acm.org}

ABSTRACT

Recent years have seen exciting developments in join algorithms. In 2008, Atserias, Grohe and Marx (henceforth AGM) proved a tight bound on the maximum result size of a full conjunctive query, given constraints on the input relation sizes. In 2012, Ngo, Porat, Ré and Rudra (henceforth NPRR) devised a join algorithm with worst-case running time proportional to the AGM bound [8]. Our commercial database system LogicBlox employs a novel join algorithm, *leapfrog triejoin*, which compared conspicuously well to the NPRR algorithm in preliminary benchmarks. This spurred us to analyze the complexity of leapfrog triejoin. In this paper we establish that leapfrog triejoin is also worst-case optimal, up to a log factor, in the sense of NPRR. We improve on the results of NPRR by proving that leapfrog triejoin achieves worst-case optimality for finer-grained classes of database instances, such as those defined by constraints on projection cardinalities. We show that NPRR is *not* worst-case optimal for such classes, giving a counterexample where leapfrog triejoin runs in $O(n \log n)$ time and NPRR runs in $\Theta(n^{1.375})$ time. On a practical note, leapfrog triejoin can be implemented using conventional data structures such as B-trees, and extends naturally to $\exists_1$ queries. We believe our algorithm offers a useful addition to the existing toolbox of join algorithms, being easy to absorb, simple to implement, and having a concise optimality proof.

General Terms

Algorithms,Theory

this Datalog rule:

$$Q(a,b,c) \leftarrow R(a,b), S(b,c), T(a,c). \quad (1)$$

where a, b, c are query variables (for intuition: if $R = S = T$, then Q finds triangles.)

Given constraints on the sizes of the input relations such as $|R| \leq n$, $|S| \leq n$, $|T| \leq n$, what is the maximum possible query result size $|Q|$? This question has practical import, since a tight bound $|Q| \leq f(n)$ implies an $\Omega(f(n))$ worst-case running time for algorithms answering such queries.

Atserias, Grohe and Marx (AGM [2]) established a tight bound on the size of Q: the *fractional edge cover* bound Q^* (Section 2.2). For the case where $|R| = |S| = |T| = n$, the fractional cover bound yields $|Q| \leq Q^* = n^{3/2}$. In earlier work, Grohe and Marx [6] gave an algorithm with running time $O(|Q^*|^2 g(n))$, where $g(n)$ is a polynomial determined by the fractional cover bound. In 2012, Ngo, Porat, Ré and Rudra (NPRR [8]) devised a groundbreaking algorithm with worst-case running time $O(Q^*)$, matching the AGM bound. The algorithm is non-trivial, and its implementation and analysis depend on rather deep machinery developed in the paper.

The NPRR algorithm was brought to our attention by Dung Nguyen, who implemented it experimentally using our framework. LogicBlox uses a novel and hitherto proprietary join algorithm we call *leapfrog triejoin*. Preliminary benchmarks suggested that leapfrog triejoin performed dramatically better than NPRR on some test problems [9]. These benchmark results motivated us to analyze our algorithm,

An example

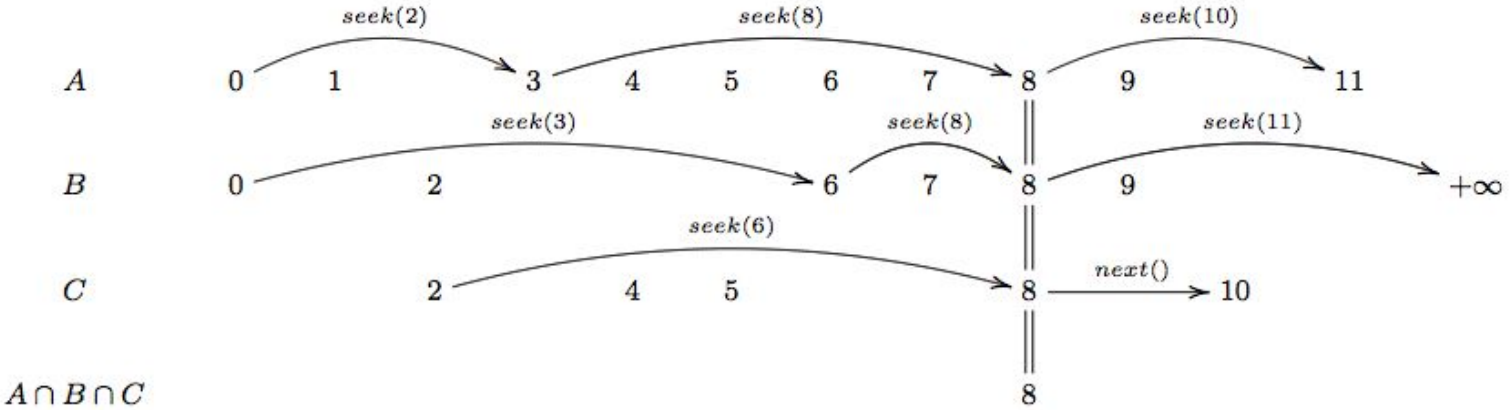


Figure 1: Example of a leapfrog join of three relations A, B, C, with $A = \{0, 1, 3, 4, 5, 6, 7, 8, 9, 11\}$ and B, C as shown in the second and third rows. Initially the iterators for A, B, C are positioned (respectively) at 0, 0, and 2. The iterator for A performs a seek(2) which lands it at 3; the iterator for B then performs a seek(3) which lands at 6; the iterator for C does seek(6) which lands at 8, etc.

Expressing optimization problems

```
// entities
FOOD(f), FOOD_id(f:s) -> string(s).
NUTR(n), NUTR_id(n:s) -> string(s).
// predicates populated in the database
amt[n, f] = v -> NUTR(n), FOOD(f), float(v).
nutrLow[n] = v -> NUTR(n), float(v).
cost[f] = v -> FOOD(f), float(v).
// unknown predicate
Buy[f] = v -> FOOD(f), float(v).
lang:solver:variable(`Buy).
// objective function and target (minimize)
objective[] = v -> float(v).
objective[] = v <- agg << v=total(z) >>
FOOD(f), cost[f] = v1, Buy[f] = v2, z = v1*v2.
lang:solver:minimal(`objective).
```

```
// constraints
NUTR(n), totalNutr[n] = v1, nutrLow[n] = v2 -> v1 >= v2.

// more predicate definitions
totalNutr[n] = v -> NUTR(n), float(v).
totalNutr[n] = v <- agg << v = total(z) >>
NUTR(n), FOOD(f),
amt[n, f] = v1, Buy[f] = v2, z = v1*v2.
```

Expressing ML problems as Convex Optimization

Data Science with Linear Programming

Nantia Makrynioti[1, 2] **Nikolaos Vasiloglou**[1] **Emir Pasalic**[1] **Vasilis Vassalos**[2]

[1]LogicBlox
{nantia.makrynioti, nikolaos.vasiloglou, emir.pasalic}@logicblox.com
[2]Department of Informatics, Athens University of Economics and Business
{vassalos}@aueb.gr

Abstract

The standard process of data science tasks is to prepare features inside a database, export them as a denormalized data frame and then apply machine learning algorithms. This process is not optimal for two reasons. First, it requires denormalization of the database that can convert a small data problem into a big data problem. The second problem is that it assumes that the machine learning algorithm is disentangled from the relational model of the problem. That seems to be a serious limitation since the relational model contains very valuable domain expertise. In this paper we explore the use of convex optimization and specifically linear programming as a data science tool that can express most of the common machine learning algorithms and at the same time it can be natively integrated inside a declarative database. We are using SolverBlox, a framework that accepts as an input Datalog code and feeds it into a linear programming solver. We demonstrate the expression of three common machine learning algorithms, Linear Regression, Factorization Machines and Spectral Clustering, and present use case scenarios where data processing and modelling of optimization problems can be done step by step inside the database.

1 Introduction

As data science becomes more and more prevalent in the industry, mathematical modelling languages, such as R and Matlab, remain popular, but users also seek other solutions, which will provide a declarative framework for defining machine learning algorithms, as well as allow them to work on data stored in relational databases. In the following sections, we describe how the "say what you want to do and not how to *et al.*, 2016], focus on supporting a number of linear algebra operators, which are common in building machine learning models, and techniques for optimizing plans consisting of these operators. However, in real world problems data is not given as a matrix or tensor, but as relational tables. These systems ignore the relational nature of data and require conversion to matrices. Apart the tedious process of exporting/importing data between a database and a machine learning system, denormalization also results in losing important domain information embedded in the relational representation. Moreover, the computation of the optimal parameters of the model should still be described and implemented by the user, which diverges from the concept of declarative programming. By defining machine learning models inside a database with a declarative language, such as LogiQL, casting them as linear programs and then delegating their solution to an appropriate solver, the user needs only to define the model and the objective function, as well as any constraints that may be useful to the task. Moreover, the machine learning algorithm workflow can include database queries. For example the user might want to apply different regularization to data points that satisfy a complicated data constraint, which is not possible to compute once the data is flattened.

We argue that linear programming is a convenient interface of non-probabilistic machine learning to logical programming. At a syntax level possible nonlinearities could be expressed either with language directives or with logical predicates, which a compiler will be able to rewrite to mathematical expressions that can be passed to a lower level solver. For example several nonlinearities can be automatically relaxed with rewritings and be processed by branch and bound solvers or they can be converted to nonconvex equivalents. To unify linear programming with logical programming, we use the SolverBlox framework and demonstrate that we can express different classes of machine learning problems, which can then be exported to the exact same format (.lp Gurobi) and be handled by external solvers. SolverBlox can also be

Other approaches

RELOOP: A Python-Embedded Declarative Language for Relational Optimization

Martin Mladenov Danny Heinrich* Leonard Kleinhans* Felix Gonsior Kristian Kersting
Computer Science Department, TU Dortmund University
{fn.ln}@cs.tu-dortmund

Abstract

We present RELOOP, a domain-specific language for relational optimization embedded in Python. It allows the user to express relational optimization problems in a natural syntax that follows logic and linear algebra, rather than in the restrictive standard form required by solvers, and can automatically compile the model to a lower-order but equivalent model. Moreover, RELOOP makes it easy to combine relational optimization with high-level features of Python such as loops, parallelism and interfaces to relational databases. RELOOP is available at http://www-ai.cs.uni-dortmund.de/weblab/static/RLP/html/ along with documentation and examples.

Introduction

"Democratization of data" does not mean dropping a huge

natural
chine l
mining
lining
the pro
guage a

Argu
than ju
functio
place,
model
ten bui
the don
and cor
commu
Clarke
Riedel

MiningZinc: A Modeling Language for Constraint-based Mining

Tias Guns[1], Anton Dries[1], Guido Tack[2], Siegfried Nijssen[1,3] and Luc De Raedt[1]
[1] Department of Computer Science, KU Leuven {*firstname.lastname*}@cs.kuleuven.be
[2] Caulfield School of Information Technology, Monash University guido.tack@monash.ed
[3] LIACS, Universiteit Leiden snijssen@liacs.nl

Abstract

We introduce MiningZinc, a general framework for constraint-based pattern mining, one of the most popular tasks in data mining. MiningZinc consists of two key components: a language component and a toolchain component.

The language allows for high-level and natural modeling of mining problems, such that MiningZinc models closely resemble definitions found in the data mining literature. It is inspired by the *Zinc* family of languages and systems and supports user-defined constraints and optimization criteria.

The toolchain allows for finding solutions to the models. It ensures the solver independence of the language and supports both standard constraint solvers and specialized data mining systems. Automatic model transformations enable the efficient use of different solvers and systems.

The combination of both components allows one to rapidly model constraint-based mining problems and execute these with a wide variety of methods. We demonstrate this experimentally for a number of well-known solvers and data mining tasks.

1 Introduction

The data mining practice contrasts sharply witl
constraint programming, where high-level language
Zinc [Marriott *et al.*, 2008], Essence [Frisch *et al.*, 2
OPL [Van Hentenryck, 1999] are used to *model* p
and general purpose *solvers* are provided to comp
tions. Motivated by the success of this declarative
in constraint programming, we propose a *modeling*
approach for data mining. This makes data mining n
ible, as it becomes easy to change the model and to
best solvers to get solutions.

As the field of data mining is diverse, we focus in t
on one of the most popular tasks, namely constrai
pattern mining. Even for the restricted data type
databases, many settings (supervised and unsupervi
corresponding systems have been proposed in the l
this makes it an adequate showcase for a declarative
to data mining. Dealing with a diverse set of constrai
pattern mining problems remains an unsolved and
challenge in data mining.

The key contribution of this paper is the introduction of a general-purpose, declarative mining framework called MiningZinc. The design criteria for MiningZinc are:

- to support the *high-level and natural modeling* of pattern mining tasks; that is, MiningZinc models should closely correspond to the definitions of data mining problems found in the literature;
- to support *user-defined constraints and criteria* such that existing problem formulations can be extended and

Foundations of Declarative Data Analysis Using Limit Datalog Programs (Extended Abstract)

Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks

Department of Computer Science, University of Oxford, UK

Analysing complex datasets is currently a hot topic in information systems. The term 'data analysis' covers a broad range of techniques that often involve tasks such as data aggregation, property verification, or query answering. Such tasks are currently often solved imperatively (e.g., using Java or Scala) by specifying *how* to manipulate the data, and this is undesirable because the objective of the analysis is often obscured by evaluation concerns. It has recently been argued that data analysis should be *declarative* [1, 12, 16, 17]: users should describe *what* the desired output is, rather than how to compute it. For example, instead of computing shortest paths in a graph by a concrete algorithm, one should (i) describe what a path length is, and (ii) select only paths of minimum length. Such a specification is independent of evaluation details, allowing analysts to focus on the task at hand. An evaluation strategy can be chosen later, and general parallel and/or incremental evaluation algorithms can be reused 'for free'.

An essential ingredient of declarative data analysis is an efficient language that can capture the relevant tasks, and Datalog is a prime candidate since it supports recursion. Apart from recursion, however, data analysis usually also requires integer arithmetic to capture quantitative aspects of data (e.g., the length of a shortest path). Research on combining the two dates back to the '90s [14, 10, 2, 18, 4, 8, 15], and is currently experiencing a revival [7, 13]. This extensive body of work, however, focuses primarily on integrating recursion and arithmetic with *aggregate functions* in a coherent semantic framework, where technical difficulties arise due to nonmonotonicity of aggregates.

How efficient is this?

- Managed to express data relations and ML operations in the same language
- Translated the problem to its algebraic representation (Grounding)
- But we still denormalized in the end the database
- Can we do anything about that?

Lifting

Relational Linear Programs, Kersting et.al

Algorithm 2: Color-Passing

Input: A graph $G = (V, E)$, an initial coloring function $\lambda_0 : V \cap E \to \mathbb{N}$
Output: A partition $\mathcal{U} = \{U_1, ..., U_k\}$ of V

Initialize $i \leftarrow 0, \mathcal{U}_0 = \{V\}$
repeat
  foreach $v \in V$ **do**
    $c \leftarrow \lambda_i(v)$
    foreach $u \in Nb_v$ **do**
      $c \leftarrow \langle c \cup (\lambda_i(u), \lambda_0(\{u, v\})) \rangle$
    end
    $\lambda_{i+1}(u) \leftarrow \text{hash}(c)$
  end
  $\mathcal{U}_{i+1} \leftarrow \{\{v \in V | \lambda_{i+1} = k\}\}$
  $i \leftarrow i + 1$
until $\mathcal{U}_{i-1} = \mathcal{U}_i$;
return $\mathcal{U}_i$;

5 Exploiting Symmetries for Reducing the Dimension of LPs

As we have already mentioned in the introduction, one of the features of many relational models is that they can produce model instances with a lot of symmetries. These symmetries in turn can be exploited to perform inference at a "lifted" level, i.e., at the level of groups of variables. For probabilistic relational models, this lifted inference can yield dramatic speed-ups, since one reasons about the groups of indistinguishable variables as a whole, instead of treating them individually.

Triggered by this success, we will now show that linear programming is liftable, too.

5.1 Detection Symmetries using Color-Passing

One way to devise a lifted inference approach is the following. One starts with a standard inference algorithm and introduces some notion of indistinguishability among the variables in the model (instance) at hand. For example, we can say that two variables X and Y in a linear program are indistinguishable, if there exist a permutation of all variables, which exchanges X and Y, yet still yields back the same model in terms of the solutions. Then, given a particular model instance, one detects, which variables are exchangeable in that model instance. The standard inference algorithm is modified in such a way that it can deal with groups of indistinguishable variables as a whole, instead of treating them individually. This approach was for instance followed to devise a lifted version of belief propagation [SD08, KAN09, AKMN13], a message-passing algorithm for approximate inference in Markov random fields (MRFs), which we will not briefly sketch in order to prepare the stage for lifted linear programming. In doing so, we will omit many details, since they are not important for developing lifted linear programming.

Belief propagation approximately computes the single-variable marginal probabilities $P(X_i)$ in an MRF encoding the joint distribution over the random variables $X_1, X_2, \ldots, X_n$. It does so by passing messages within a graphical represention of the MRF.

The main idea to lift belief propagation is to simulate it keeping track of which X_is and clauses send identical messages. These elements of the model can then be merged into groups, whose members are indistinguishable in terms of belief propagation. After grouping elements together into a potentially smaller (lifted) MRF, a modified message-passing computes the same beliefs as standard belief propagation on the original MRF.

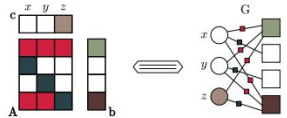


Figure 7: Construction of the coefficient graph G_L of L^0. On the left-hand side, the coloring of the LP is shown. This turns into the colored coefficient graph shown on the right-hand side.

with knowledge base LogKB (recall that logical atoms are assumed to evaluate to 0 and 1 within an RLP):

```
widget(x).
widget(y).
gadget(z).
```

If we ground this linear program and convert it to dual form (as in Eq. 1), we obtain the following linear program $L^0 = (\mathbf{A}, \mathbf{b}, \mathbf{c})$

$$\begin{array}{ll} \underset{[x,y,z]^T \in \mathbb{R}^3}{\text{minimize}} & 0x + 0y + 1z \\ \text{subject to} & \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \leq \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}, \end{array}$$

where for brevity we have substituted $\text{p}(x), \text{p}(y), \text{p}(z)$ by x, y, z respectively. The coefficient graph of L^0 is shown in Fig. 7.

We call an **equitable partition of a linear program** L the equitable partition of the graph G_L[3]. Suppose now we compute an equitable partition $\mathcal{U} = \{P_1, \ldots, P_p, Q_1, \ldots, Q_q\}$ of G_L using Algorithm 2 and compute the corresponding fractinal automorphism $(\mathbf{X}_P, \mathbf{X}_Q)$ as in Eq. 3. Observer that $(\mathbf{X}_P, \mathbf{X}_Q)$ will have the following properties:

i) due to Theorem 3, we have $\mathbf{X}_Q\mathbf{A} = \mathbf{A}\mathbf{X}_P$;

ii) by our choice of initial colors of G_L, the partition $\mathcal{U}$ will never group together variable vertices i, j with $\mathbf{c}_i \neq \mathbf{c}_j$, nor will it group constraint vertices i, j with $\mathbf{b}_i \neq \mathbf{b}_j$. By Eq. 3, this implies

$$\mathbf{c}^T\mathbf{X}_P = \mathbf{c}^T$$

and

$$\mathbf{X}_Q\mathbf{b} = \mathbf{b}\,.$$

This yields the definition of a **fractional automorphism of linear programs** – we call a pair of doubly stochastic matrices $(\mathbf{X}_P, \mathbf{X}_Q)$ a **fractional automorphism of the linear program** L if it satisfies properties *i*) and *ii*) as above.

[3] using the notion of equitable partitions of bipartite colored graphs from the previous section.

Does not solve the problem completely

It requires temporary denormalization but the final matrix is small

There are tricks that can save temporary memory

Observation

We need to build a framework for mathematical operations in the relational domain

Building solvers in the database

FAQ: Questions Asked Frequently

Mahmoud Abo Khamis[*†] Hung Q. Ngo[*†]

[*] LogicBlox Inc.
{mahmoud.abokhamis,hung.ngo}@logicblo

[†] Department of Computer Science and Eng
University at Buffalo, SUNY
{mabokham,hungngo,atri}@buffalo.e

Abstract

We define and study the **F**unctional **A**ggregate **Q**uery (FAQ) problem frequently asked questions in constraint satisfaction, databases, matrix opera models and logic. This is our main conceptual contribution.

We then present a simple algorithm called InsideOut to solve this ge a variation of the traditional dynamic programming approach for constr variable elimination. Our variation adds a couple of simple twists to basic to deal with the generality of FAQ, to take full advantage of Grohe and framework, and of the analysis of recent worst-case optimal relational join

As is the case with constraint programming and graphical model infer efficiently we need to solve an optimization problem to compute an approp main technical contribution of this work is a precise characterization of 'semantically equivalent' to the variable ordering given by the input FAQ ex approximation algorithm to find an equivalent variable ordering that has the Our results imply a host of known and a few new results in graphical model relational joins, and logic.

We also briefly explain how recent algorithms on beyond worst-case an solving SAT and #SAT can be viewed as variable elimination to solve FAQ input functions.

In-Database Learning with Sparse Tensors

Mahmoud Abo Khamis[1] Hung Q. Ngo
Dan Olteanu[3] Maximilia

[1]LogicBlox, Inc. [2]University of Michigan
{mahmoud.abokhamis,hung.ngo}@logicblox.
{dan.olteanu,max.schleich}@

ABSTRACT

In-database analytics is of great practical importance as it avoids the costly repeated loop data scientists have to deal with on a daily basis: select features, export the data, convert data format, train models using an external tool, reimport the parameters. It is also a fertile ground of theoretically fundamental and challenging problems at the intersection of relational and statistical data models.

This paper introduces a unified framework for training and evaluating a class of statistical learning models inside a relational database. This class includes ridge linear regression, polynomial regression, factorization machines, and principal component analysis. We show that, by synergizing key tools from relational database theory such as schema information, query structure, recent advances in query evaluation algorithms, and from linear algebra such as various tensor and matrix operations, one can formulate in-database learning problems and design efficient algorithms to solve them.

The algorithms and models proposed in the paper have already been implemented inside the LogicBlox database engine and used in retail-planning and forecasting applications, with significant performance benefits over out-of-database solutions that require the costly data-export loop.

1. INTRODUCTION

Although both disciplines of databases and statistics occupy foundational roles for the emerging field of data science, they are largely seen as complementary. Most fundamental contributions made by statisticians and machine learning researchers are abstracted away from the underlying infrastructure for data man-

The Symbolic Interior Point Method

Martin Mladenov
TU Dortmund Univeristy
martin.mladenov@cs.tu-dortmund.de

Vaishak Belle
KU Leuven
vaishak@cs.kuleuven.be

Kristian Kersting
TU Dortmund University
kristian.kersting@cs.tu-dortmund.de

Abstract

A recent trend in probabilistic inference emphasizes the codification of models in a formal syntax, with suitable high-level features such as individuals, relations, and connectives, enabling descriptive clarity, succinctness and circumventing the need for the modeler to engineer a custom solver. Unfortunately, bringing these linguistic and pragmatic benefits to numerical optimization has proven surprisingly challenging. In this paper, we turn to these challenges: we introduce a rich modeling language, for which an interior-point method computes approximate solutions in a generic way. While logical features easily complicates the underlying model, often yielding intricate dependencies, we exploit and cache local structure using algebraic decision diagrams (ADDs). Indeed, standard matrix-vector algebra is efficiently realizable in ADDs, but we argue and show that well-known second-order methods are not ideal for ADDs. Our engine, therefore, invokes a sophisticated matrix-free approach. We demonstrate the flexibility of the resulting symbolic-numeric optimizer on decision making and compressed sensing tasks with millions of non-zero entries.

1 Introduction

USE CASE: LINEAR ALGEBRA

LogiQL – linear algebra via relational programming

```
vecA[x] = vecB[x] + vecC[x] + c[].        // pointwise addition

vecA[x] = vecB[x] * vecC[x].              // pointwise multiplication

scalar[] += vecB[x] * vecC[x].         // dot product

matA[x, y] = vecB[x] * vecC[y].           // vector multiplication

matA[x, y]  = matB[x, y] + matC[x, y].    // matrix addition

matA[x, y] += matB[x, t] * matC[t, y].    // matrix multiplication
```

USE CASE: LINEAR ALGEBRA

LogiQL – basic feed-forward neural network

```
out[Node] = sigmoid[in[Node]].
sigmoid[X] = 1/(1+exp[-X]).

in[Node] += weight[Node,Child]*out[Child].
in[Node] += input[Node].

error += (out[Node]-target[Node])**2.
```

FAQ

Problem	FAQ formulation	Previous Algo.	**Our Algo.**
#QCQ	$\sum_{(x_1,\ldots,x_f)} \bigoplus_{x_{f+1}}^{(f+1)} \cdots \bigoplus_{x_n}^{(n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	Open	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{faqw}(\varphi)}$
QCQ	$\bigoplus_{x_{f+1}}^{(f+1)} \cdots \bigoplus_{x_n}^{(n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{PW}(\mathcal{H})}$ [21]	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{faqw}(\varphi)}$
#CQ	$\varphi = \sum_{(x_1,\ldots,x_f)} \max_{x_{f+1}} \cdots \max_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{DM}(\mathcal{H})}$ [31]	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{faqw}(\varphi)}$
Marginal Distribution	$\sum_{(x_{f+1},\ldots,x_n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(\lvert\mathcal{H}\rvert) \cdot N^{\mathsf{htw}(\varphi)}$ [49]	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{faqw}(\varphi)}$
MAP query	$\max_{(x_{f+1},\ldots,x_n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(\lvert\mathcal{H}\rvert) \cdot N^{\mathsf{htw}(\varphi)}$ [49]	$\lvert\mathcal{H}\rvert^{O(1)} \cdot N^{\mathsf{faqw}(\varphi)}$
Matrix Chain Mult.	$\sum_{x_2,\ldots,x_{n-1}} \prod_{i=1}^{n-1} \psi_{i,i+1}(x_i, x_{i+1})$	DP bound [25]	DP bound
DFT	$\sum_{(y_0,\ldots,y_{m-1})\in\mathbb{Z}_p^m} b_y \cdot \prod_{0\le j+k<m} e^{i2\pi \frac{x_j \cdot y_k}{p^{m-j-k}}}$	$O(n \log_p n)$ [24]	$O(n \log_p n)$

Table 1: Runtimes of algorithms assuming optimal variable ordering is given. Problems shaded red are in CSPs and logic ($\mathbf{D} = \{0,1\}$), problems shaded green fall under PGMs ($\mathbf{D} = \mathbb{R}_+$), and problems shaded blue fall under matrix operations ($\mathbf{D} = \mathbb{C}$). N denotes the size of the largest factor (assuming they are represented with the listing format). $\mathsf{htw}(\varphi)$ is the notion of integral cover width defined in [49] for PGM.

Toy Query 1

```
_f(x) <- int:range(1,100,1,x).  // n = 100

R[] = n <- agg << n=count() >>
           _f(x), _f(y), _f(z), _f(w).
```

n	With InsideOut	Without InsideOut
100	0.0665121	16.6821
1000	0.0605559	Killed after 1 hour
10000	0.092855	

InsideOut Rewrite

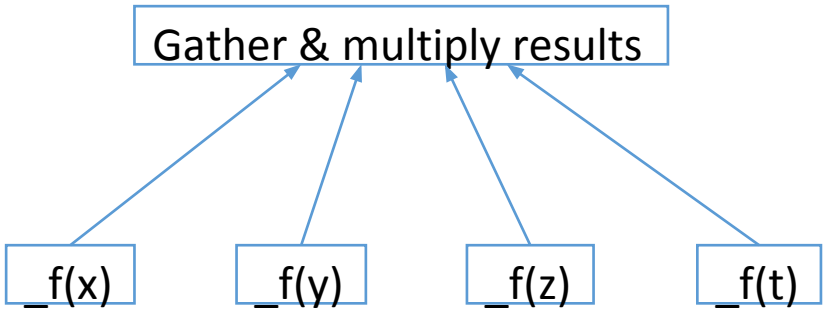
```
_CNT_a[] = a <- agg<<a = count()>> _f(z).
_CNT_b[] = b <- agg<<b = count()>> _f(y).
_CNT_c[] = c <- agg<<c = count()>> _f(x).
_CNT_d[] = d <- agg<<d = count()>> _f(w).
```

```
S[]=n <-
    _CNT_a[] = a,
    _CNT_b[] = b,
    _CNT_c[] = c,
    _CNT_d[] = d,
    int:multiply[c, d]   = cd,
    int:multiply[b, cd]  = bcd,
    int:multiply[a, bcd] = abcd,
    int:eq_2(n, abcd).
```

What Did InsideOut Exploit?

Independence!

```
R[] = n <- agg << n=count() >>
      _f(x), _f(y), _f(z), _f(w).
```



Toy Query 2

```
_f(x) <- int:range(1,100,1,x).  // n = 100

R[] = n <- agg << n=count() >>
           _f(x), _f(y), _f(z), _f(w),
           x < y < z < w.
```

n	With InsideOut	Without InsideOut
100	0.137632	1.19114
1000	0.450424	?
10000	25.6839	?

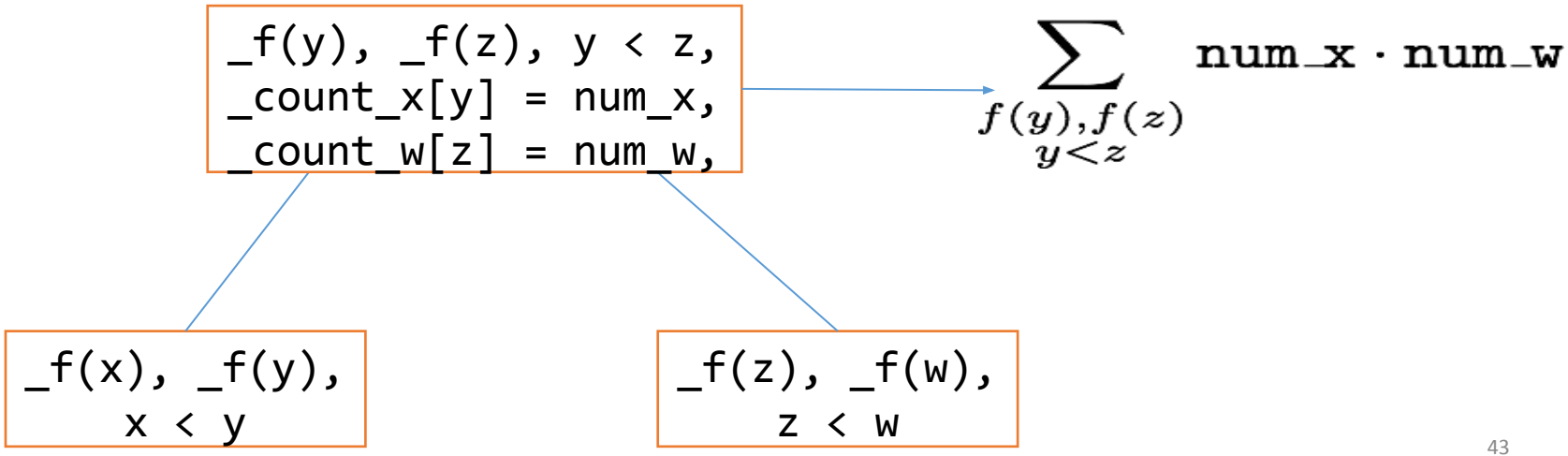
InsideOut Rewrite

```
_count_x[y] = num_x <-
    agg<<numx = count()>>
       _f(x),
       _f(y),
       x < y.

_count_w[z] = num_w <-
    agg<<num_w = count()>>
       _f(z),
       _f(w),
       z < w.

T[] = n <- agg<<n = total(xw)>>
       y < z,
       _count_x[y] = num_x,
       _count_w[z] = num_w,
       int:multiply[num_x, num_w] = xw.
```

What did it take advantage of?

Conditional Independence!

```
R[] = n <- agg << n=count() >>
          _f(x), _f(y), _f(z), _f(w),
          x < y < z < w.
```



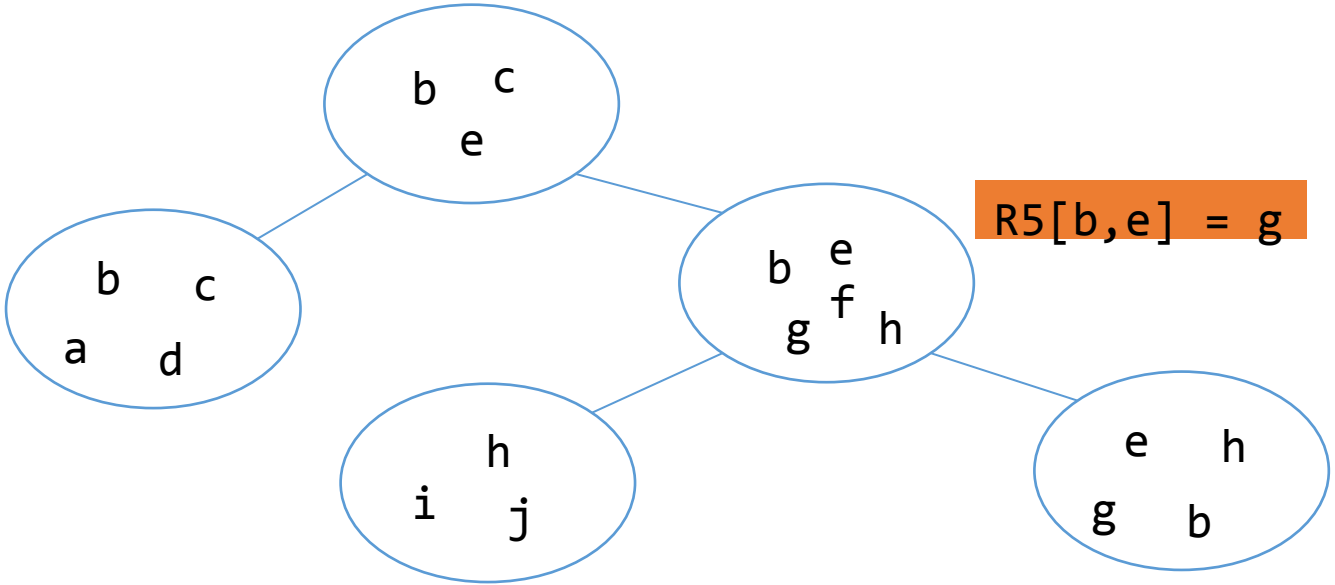
Tree Decomposition

- Rudolf Halin (1976)
- Neil Robertson and Paul Seymour (1984) – rediscovered

- Widely used in
  - Query optimization (?!)
  - CSP
  - Machine learning
  - Algorithm design (even for NP-hard problems)

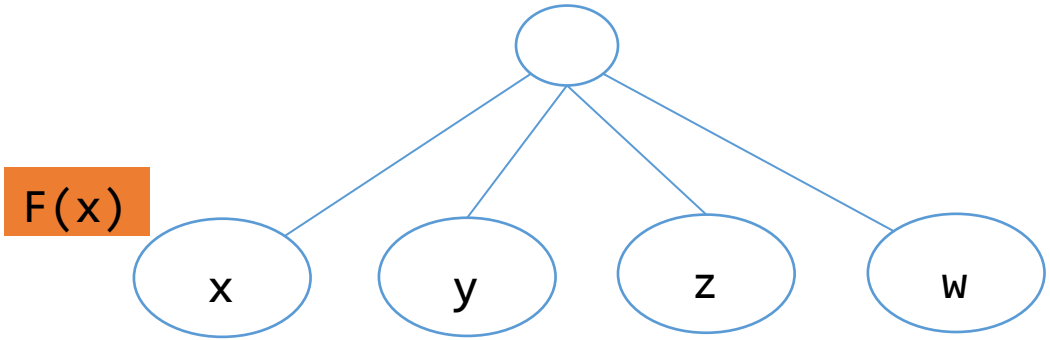
What is a Tree Decomposition (TD)?

```
R[] = n <- agg<<n = count()>>
            R1(a,b,d), c<d, R2(c,b,d),
            R3(b,e), R4(c,e), b+e=f, R5[b,e] = g, g/f = h,
            R7(i,j,h), R8(e,g), e*g=b, e-b=h.
```



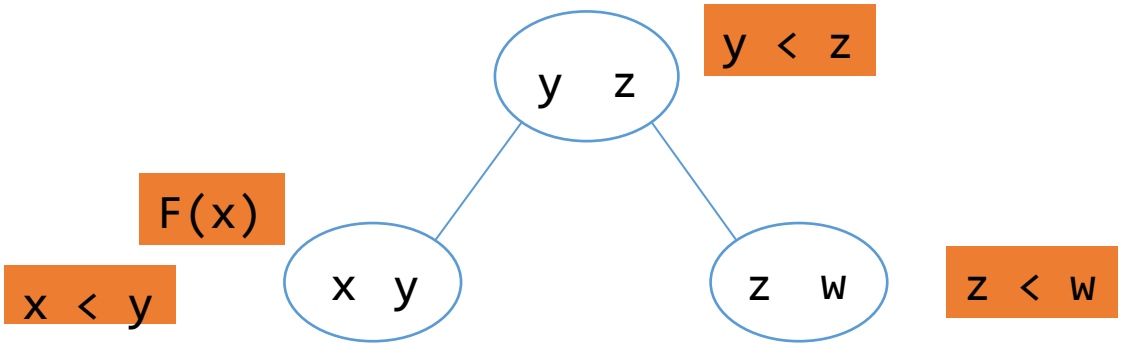
Example 1:

```
R[] = n <- agg<<n = count()>> F(x), F(y), F(z), F(w).
```



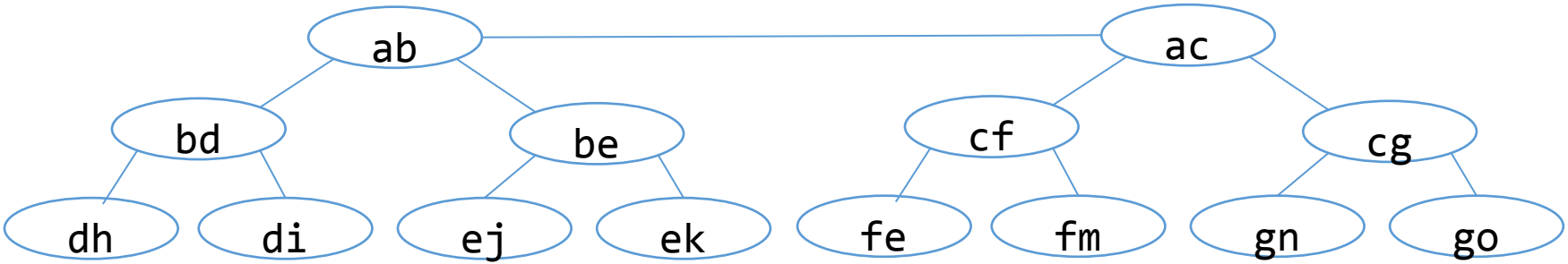
Example 2:

```
R[] = n <- agg << n=count() >>
      F(x), F(y), F(z), F(w),
       x < y < z < w.
```



Example 3: a big tree-like query

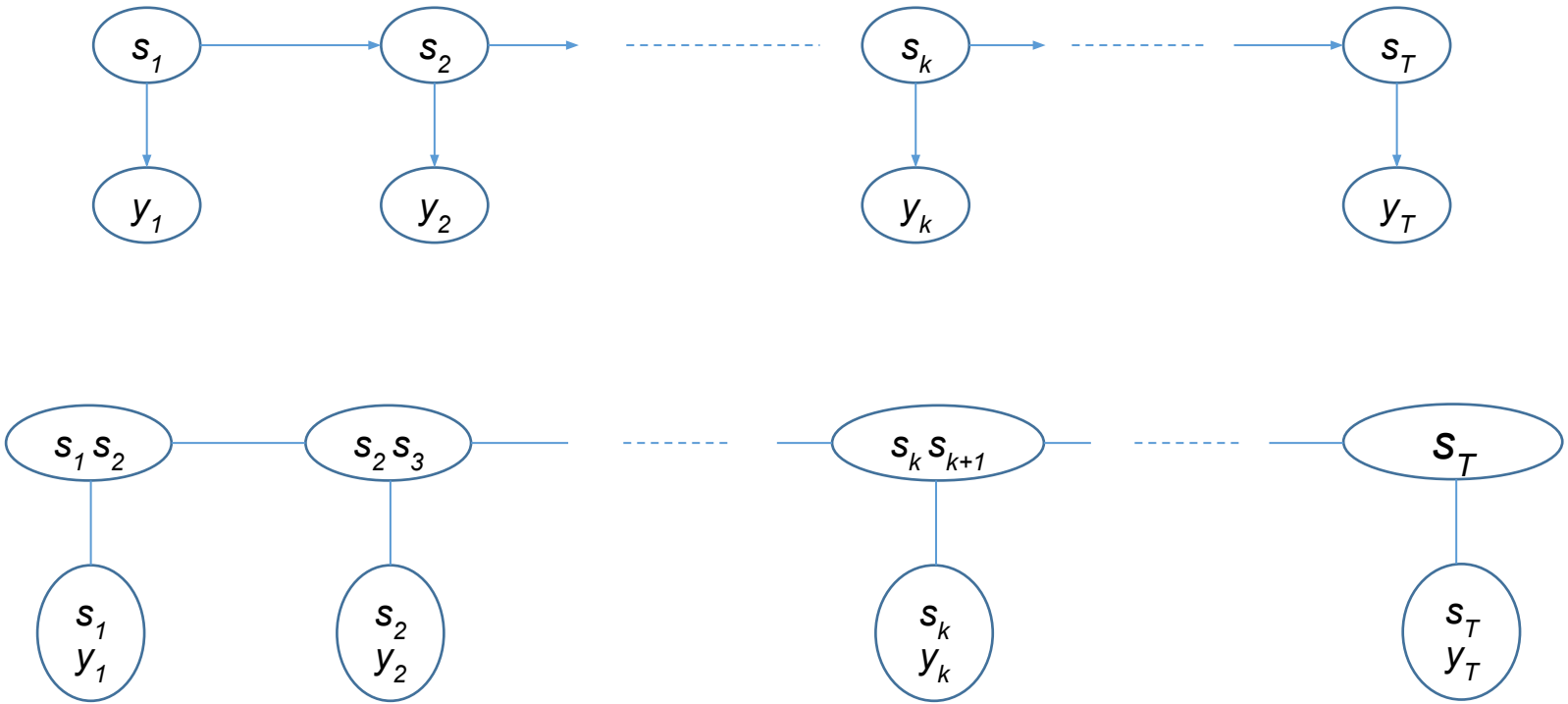
```
R[] = c <- agg<<c = count()>> E(a,b), E(a,c), E(b,d),
E(b,e), E(c,f), E(c,g), E(d,h), E(d,i), E(e,j), E(e,k),
E(f,l), E(f, m), E(g,n), E(g,o), V1(h), V2(i), V3(j),
V4(k), V5(l), V6(m), V7(n), V8(o).
```



Example 4: dumbbell

```
R[] = t <-
        agg<<t = count()>>
      E(a,b), E(a,c), E(b,c),
      E(c,d),
      E(d,e), E(d,f), E(e,f).
```

Example 6: Hidden Markov Model



Example 4: dumbbell, $O(N^{3/2})$ vs. $O(N^3)$

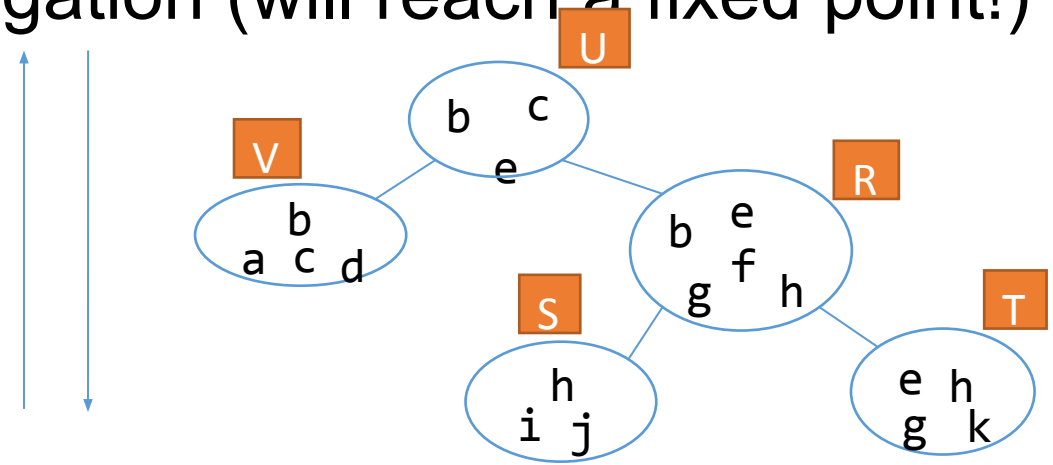
Graph	With InsideOut	Without InsideOut
facebook	4.37485	1608.97
loc-brightkite	1.03651	174.984
email-Enron	3.57298	562.598
wiki-vote	1.65755	1047.16

Example 5: 4-path query, O(N) vs $O(N^3)$

Data set	With InsideOut	Without	Speedup ratio
email-Enron	0.293187	8393.72	28,629
facebook	0.226596	3468.08	15,305
loc-brightkite	0.385222	1432.24	3,717
wiki-vote	0.165675	20736.6	125,164

Belief Propagation (will reach a fixed point!)

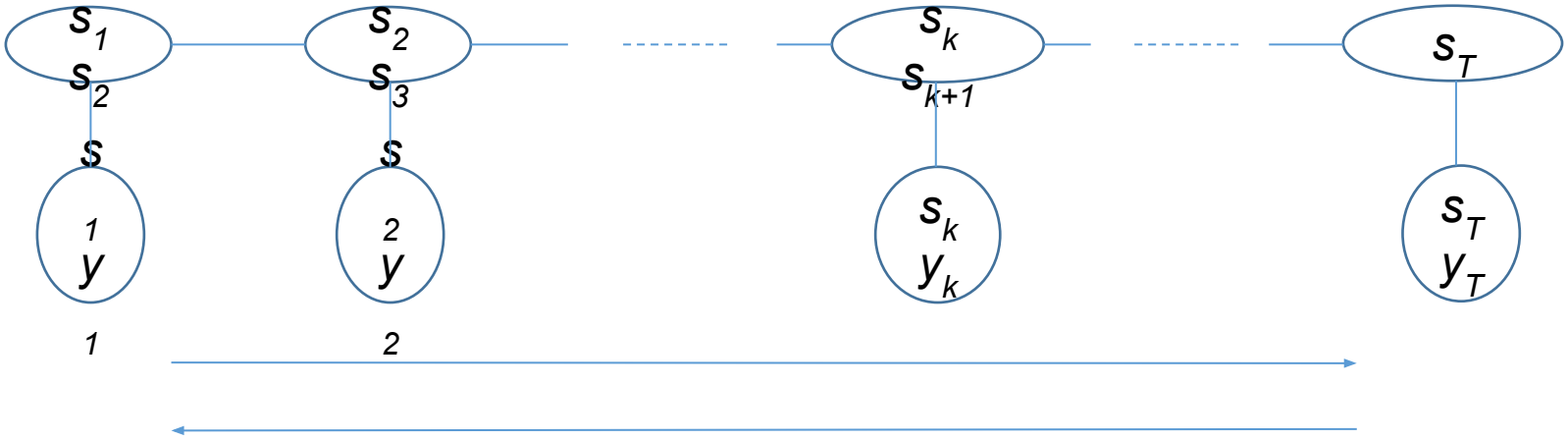
Asynchronous message passing




```
_U[b,c,e]      = u      // (also called clique tree, junction tree)
_V[a,b,c,d]    = v      // after MPs, factors in calibrated state!
_R[b,e,f,g,h]  = r      // Under Boolean Semiring, it means
_S[h,i,j]      = s      // _U = projection of output onto {b,c,e}
_T[e,g,h,k]    = t      // _V, _R, _S, _T = proj of output ...
```

In PGM, we can query for a gazillion things here, e.g.
+ What's the marginal probability on two variables {b,e}?
+ What's the {b,e} that has the maximum marginal probability?

HMM: Forward-backward & Viterbi



Asynchronous
message passing

Typical Example: Q10

```
_lostRevenue[customer] = r <-
   agg<< r = total(revenue) >>
      revenue = decimal:multiply[L_EXTENDEDPRICE[order, li],
                decimal:subtract[_one[], L_DISCOUNT[order, li]]],
      O_ORDERDATE[order]=orderdate,
      _date[] <= orderdate < _endDate[],
      L_RETURNFLAG[order, li] = RF_NAME_INV["R"],
      customer = O_CUSTKEY[order].
```

Typical Example: Q10

```
_CONJ_1ZVGA58Q[order]=customer <-
   O_ORDERDATE[order]=orderdate,
   datetime:le_2(t_Zdd2gZ4_QGJ,t_Zdd2gZ4_QEl),
   _date[]=t_Zdd2gZ4_QGJ,
   datetime:eq_2(orderdate,t_Zdd2gZ4_QEl),
   datetime:lt_2(t_Zdd2gZ4_QEl,t_Zdd2gZ4_QGK),
   _endDate[]=t_Zdd2gZ4_QGK,
   O_CUSTKEY[order]=customer.

_PRJ_1ZVGCXHS(order) <- _CONJ_1ZVGA58Q[order]=customer.

_TOT_1ZVGGU6O[order]=Var_1ZVGEZOY <-    agg<<Var_1ZVGEZOY = total(revenue)>>
      decimal:multiply[t_Zdd2gZ4_QGF,t_Zdd2gZ4_QGI]=revenue,
      L_EXTENDEDPRICE[order,li]=t_Zdd2gZ4_QGF,
      decimal:subtract[t_Zdd2gZ4_QGG,t_Zdd2gZ4_QGH]=t_Zdd2gZ4_QGI,
      _one[]=t_Zdd2gZ4_QGG,
      L_DISCOUNT[order,li]=t_Zdd2gZ4_QGH,
      L_RETURNFLAG[order,li]=t_Zdd2gZ4_QGL,
      RF_NAME_INV[t_Zdd2gZ4_QGa]=t_Zdd2gZ4_QGL,
      string:eq_2(t_Zdd2gZ4_QGa,"R"),
      _PRJ_1ZVGCXHS(order).
```

In DB model training

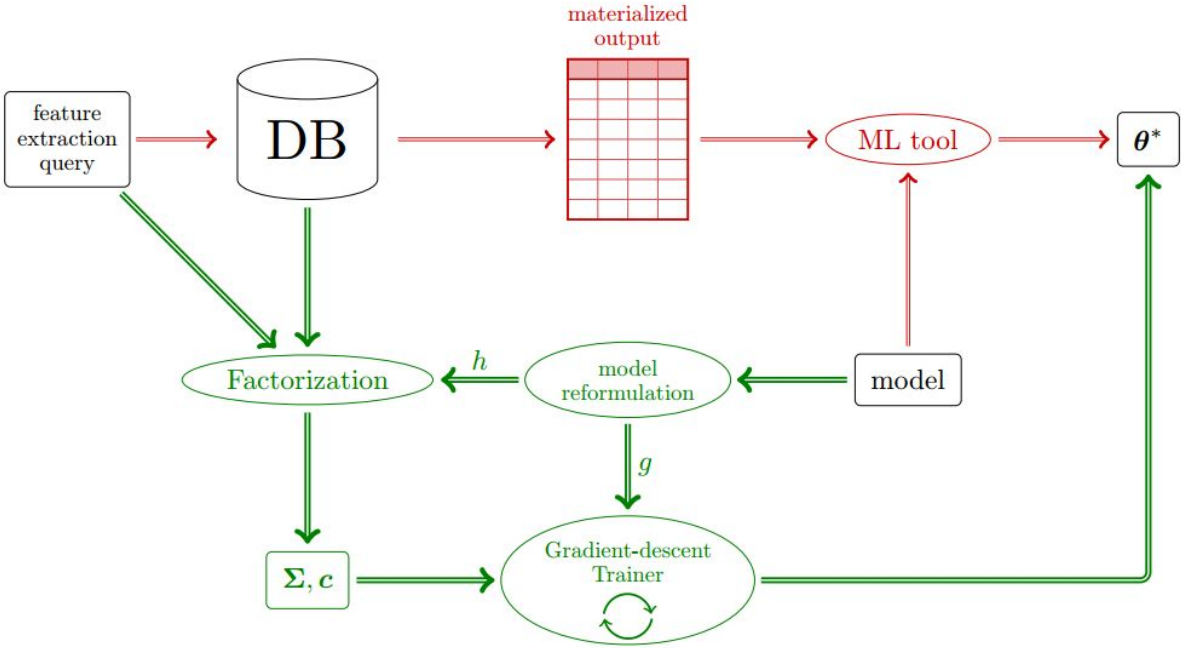


Figure 2: **In-database** vs. Out-of-database learning: High-level diagram. (See Section D.1.)

More detailed

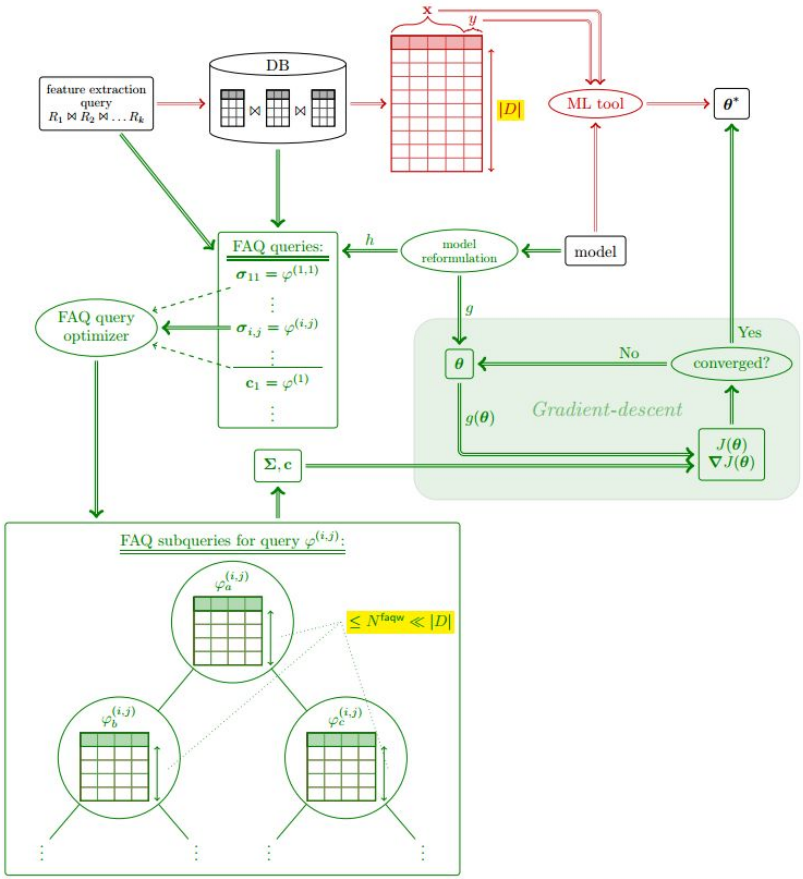


Figure 3: **In-database** vs. Out-of-database learning: Low-level diagram. (See Section D.2.)

Even more details

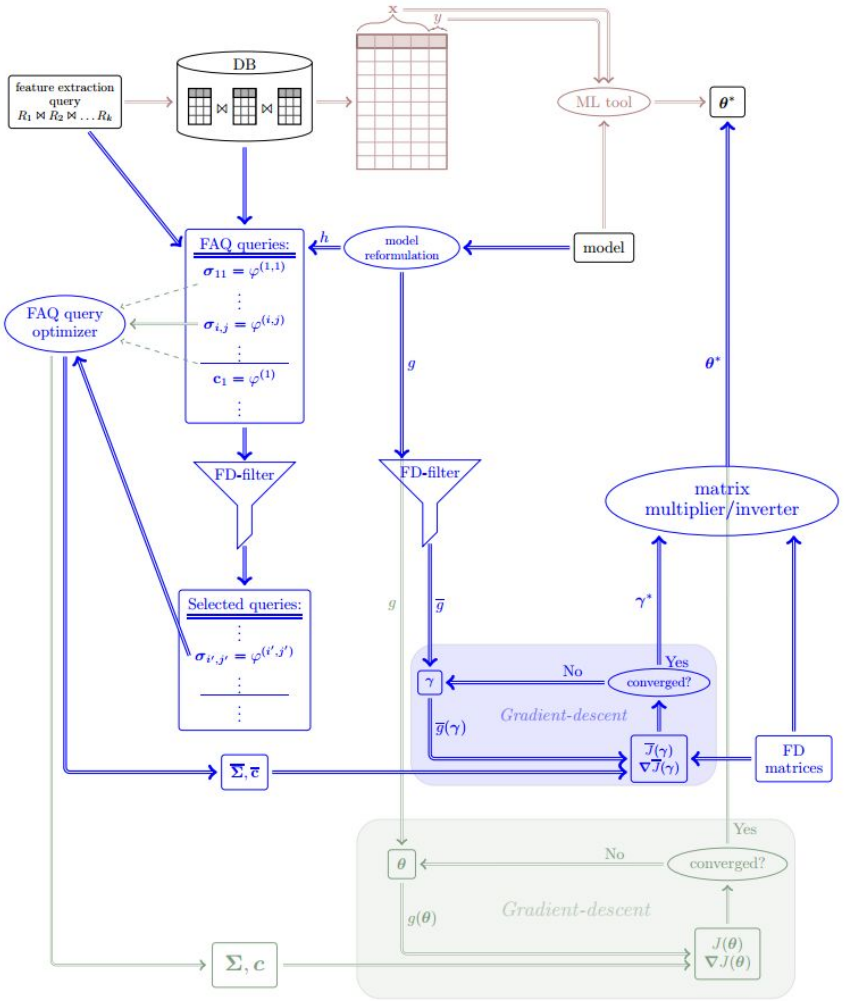


Figure 4: **FD-aware** in-database learning (vs. in-database learning without FD). (See Section D.3).

LINEAR REGRESSION

		V_1	V_2	V_3	V_4
Linear regression LR					
Features (cont.+categ.)	without FDs	33 + 55	33+55	33+1340	33+3702
	with FDs	33 + 55	33+55	33+1340	33+3653
Aggregates (cont.+categ.)	without FDs	595+2,418	595+2,421	595+111,549	595+157,735
	with FDs	595+2,418	595+2,421	595+111,549	595+144,589
M (ols)	Learn	1,898.35	8,855.11	> 79,200.00	–
R (qr)	Join (PSQL)	50.63	–	–	–
	Export/Import	308.83	–	–	–
	Learn	490.13	–	–	–
DC	Aggregate	93.31	424.81	OOM	OOM
	Converge (runs)	0.01 (359)	0.01 (359)		
AC	Aggregate	25.51	116.64	117.94	895.22
	Converge (runs)	0.02 (343)	0.02 (367)	0.42 (337)	0.66 (365)
AC+FD	Aggregate	same as **AC**			380.31
	Converge (runs)	there are no FDs			8.82 (366)
Speedup	**AC+FD**/M	74.36×	75.91×	> 669.14×	∞
	AC+FD/R	33.28×	∞	∞	∞
	AC+FD/DC	3.65×	3.64×	∞	∞
	AC+FD/AC	same as **AC**, there are no FDs			2.30×

POLYNOMIAL REGRESSION

Polynomial regression of degree 2 PR^2					
Features (cont.+categ.)	without FDs	562+2,363	562+2,366	562+110,209	562+154,033
	with FDs	same as above, there are no FDs			562+140,936
Aggregates (cont.+categ.)	without FDs	158k+742k	158k+746k	158k+65,875k	158k+46,113k
	with FDs	same as above, there are no FDs			158k+36,712k
M (ols)	Learn	> 79,200.00	> 79,200.00	> 79,200.00	–
AC	Aggregate	132.43	517.40	820.57	7,012.84
	Converge (runs)	3.27 (321)	3.62 (365)	349.15 (400)	115.65 (200)
AC+FD	Aggregate	same as **AC**			1819.80
	Converge (runs)	there are no FDs			219.51 (180)
Speedup	**AC+FD**/M	> 583.64×	> 152.01×	> 67.71×	∞
	AC+FD/AC	same as **AC**, there are no FDs			3.50×

FACTORIZATION MACHINES

Factorization machine of degree 2 and rank 8 FaMa_8^2					
Features (cont.+categ.)	without FDs	530+2,363	530+2,366	530+110,209	530+154,033
	with FDs	same as above, there are no FDs			562+140,936
Aggregates (cont.+categ.)	without FDs	140k+740k	140k+744k	140k+65,832k	140k+45,995k
	with FDs	same as above, there are no FDs			140k+36,595k
libFM (MCMC)	Join (PSQL)	50.63	216.56	216.56	216.56
	Export/Import	412.84	1462.54	3,096.90	3,368.06
	Learn (300 runs)	19,692.90	103,225.50	79,839.13	87,873.75
AC	Aggregate	128.97	498.79	772.42	6869.47
	Converge (runs)	3.03 (300)	3.05 (300)	262.54 (300)	166.60 (300)
AC+FD	Aggregate	same as **AC**			1672.83
	Converge (runs)	there are no FDs			144.07 (300)
Speedup	**AC+FD**/libFM	152.70×	209.03×	80.34×	50.33×
	AC+FD/**AC**	same as **AC**, there are no FDs			3.87×

Conclusions

- Efficient datascience requires convergence to one universal language
- ML, DB, etc have to co-exist
- Datalog is a good candidate but not the only one
- Algorithms should move beyond the input matrix paradigm
- Input = Data + Programs (joins)