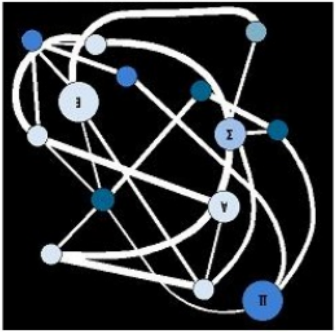
Handling Multi-Relational Data

April 23, 2019



Handling Multi-Relational Data

- Value: ability to transform a “tsunami” of data into knowledge
- Veracity: quality

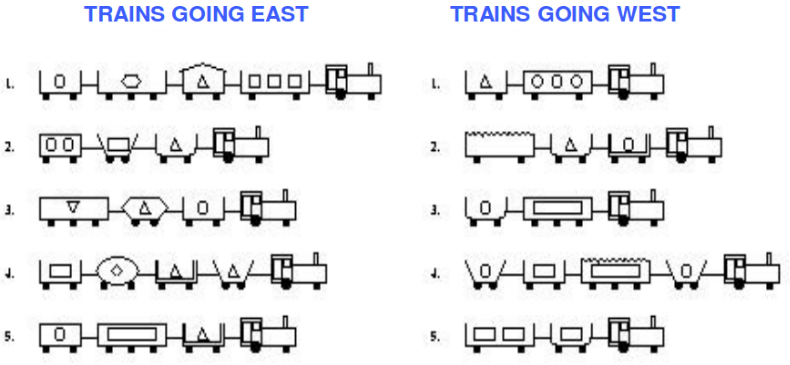
Handling Multi-Relational Data

- Most machine learning methods build models based on a 2-dimensional table, where, usually, rows are instances and columns are variables
  - exception: itemset mining
- How about multi-relational or multi-modal data?
- **Relations** may exist among instances, not only among variables
- How to capture all relations?

Limitations of a single 2-dimensional table

- data may need to be preprocessed: often big/full join of multiple tables
- joining multiple tables may introduce redundancy or bias
- consume space
  - if data is missing, may use compression techniques for sparse matrices (e.g., CSR - compact sparse format)

Handling Multi-Relational Data: example 1



Handling Multi-Relational Data: example 2

Relations among instances, variables with multiple values

Patient	Location	Size	Date	Calcifications
$\boxed{\textbf{P1}}$	C	0.1	20050403	$\boxed{\textbf{F, A}}$
$\boxed{\textbf{P1}}$	C	0.2	20060412	F
$\boxed{\textbf{P1}}$	9	0.1	20060412	A
P2	12	0.3	20050415	M
...	...	...	...	...

Ideal system to handle big data

- uncertainty
- multiple relations
- multiple modalities
- streamed data
- explain the model!
- ...all of this consuming a minimum number of resources!

Models that can explain

- Rules
- Decision trees (hierarchical propositional rules)
- Bayesian networks
- But...
  - Decision trees and Bayesian networks are not multi-relational
  - Rules can be multi-relational if the representation is in first-order logic

Example of first order logic representation

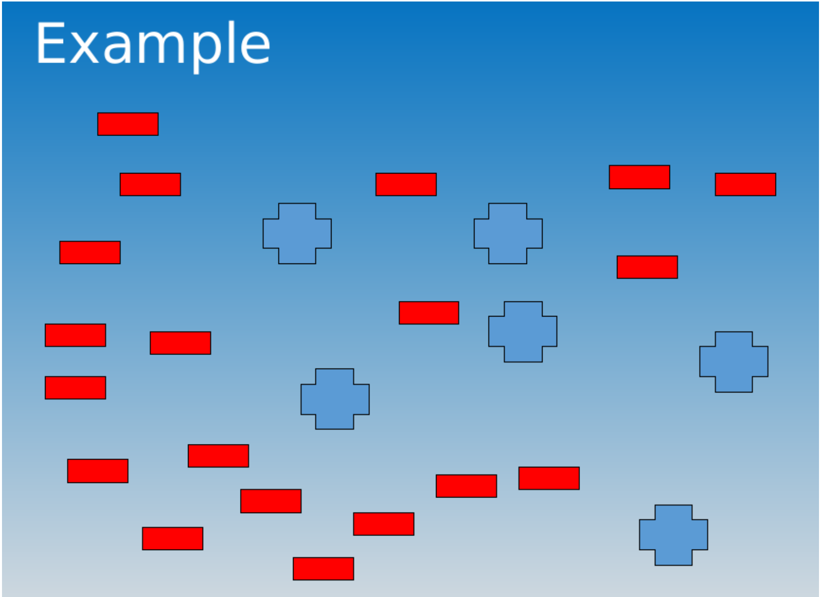
short(car_12).
closed(car_12).
long(car_11).
long(car_13).
short(car_14).
open_car(car_11).
open_car(car_13).
open_car(car_14).
shape(car_11,rectangle).
shape(car_12,rectangle).
shape(car_13,rectangle).
shape(car_14,rectangle).

load(car_11,rectangle,3).
load(car_12,triangle,1).
load(car_13,hexagon,1).
load(car_14,circle,1).
wheels(car_11,2).
wheels(car_12,2).
wheels(car_13,3).
wheels(car_14,2).
has_car(east1,car_11).
has_car(east1,car_12).
has_car(east1,car_13).
has_car(east1,car_14).

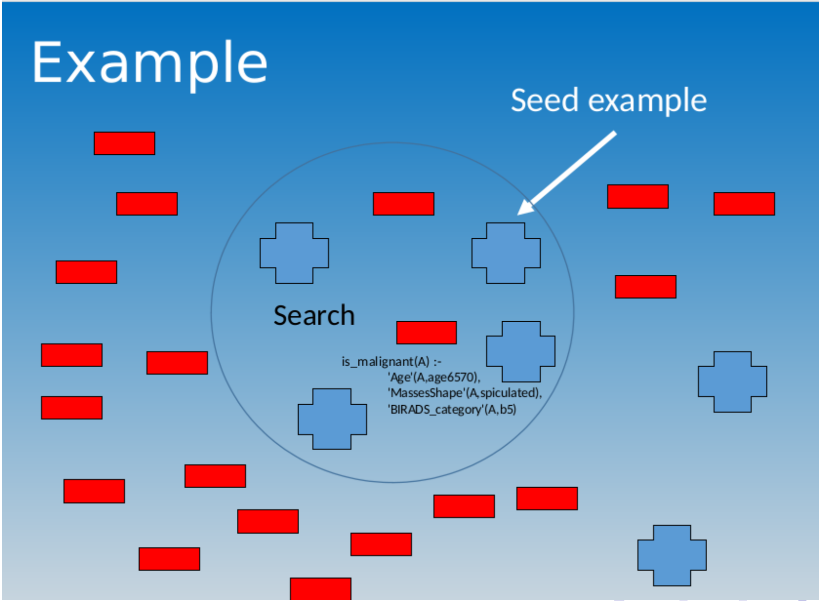
Main algorithm to learn first-order rules

- Given E = E+ U E-,BK, language and constraints C
- Repeat until E+ is empty:
  - Select any example **e** from E+
  - Build a list of candidate literals using C, BK and **e**
  - Search for a "good" hypothesis H **(parallelization is here on the coverage step)**
  - Add H to theory T
  - Remove from E+ positive examples covered by H
- Return T and its confusion matrix

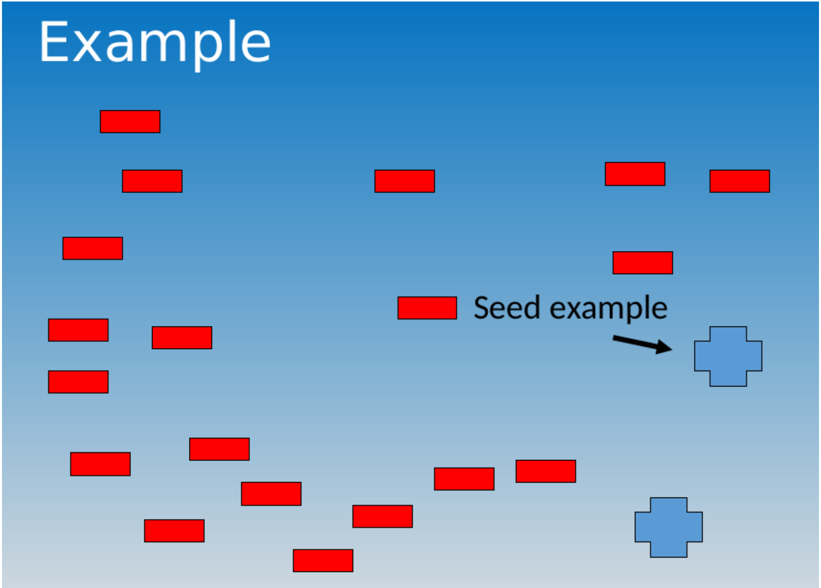
Main algorithm to learn first-order rules



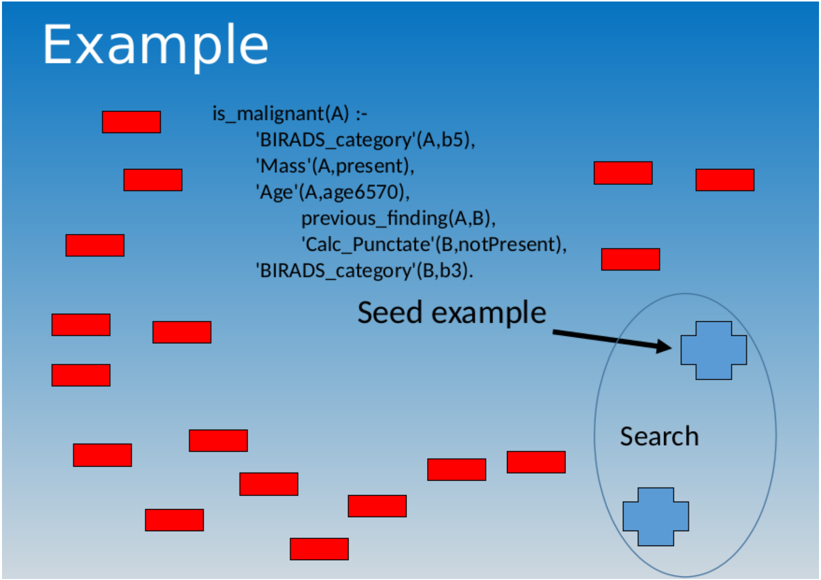
Main algorithm to learn first-order rules



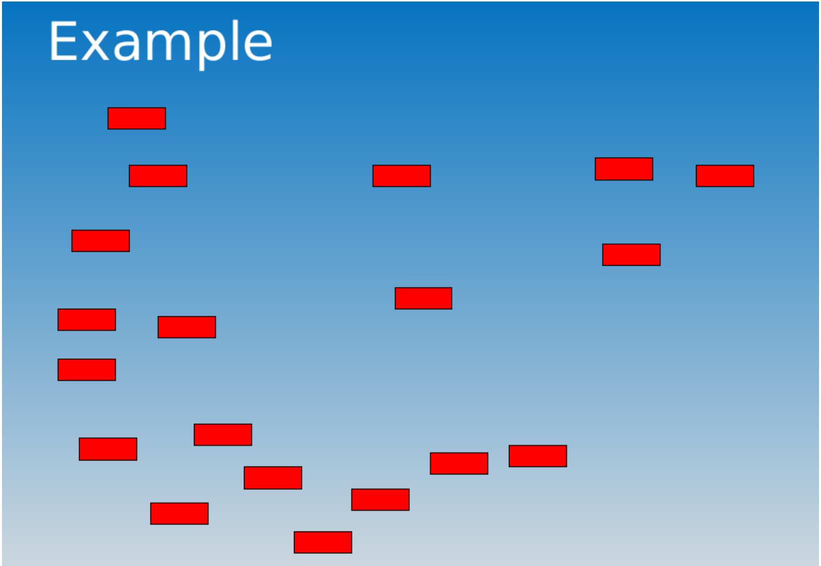
Main algorithm to learn first-order rules



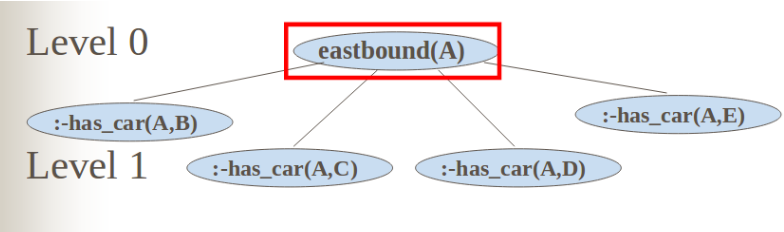
Main algorithm to learn first-order rules



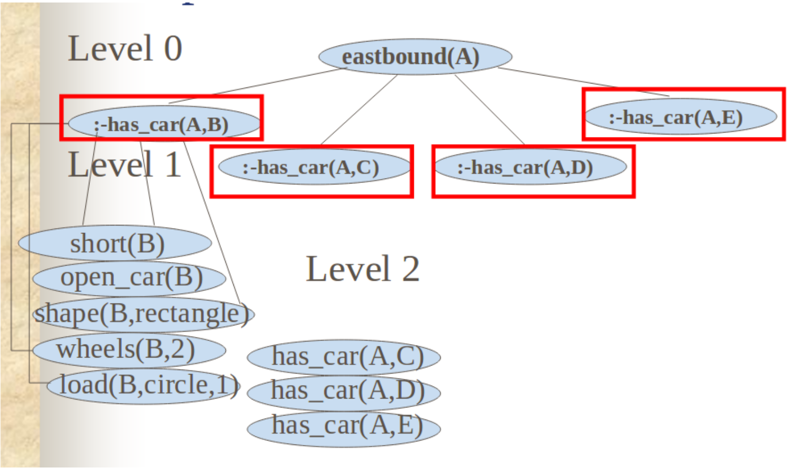
Main algorithm to learn first-order rules



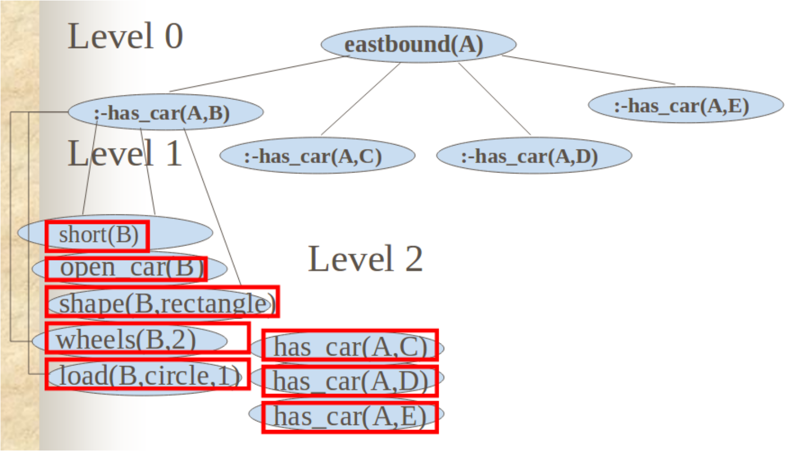
Search tree for hypothesis



Search tree for hypothesis



Search tree for hypothesis



Opportunities for optimization

- representation for the coverage lists
- parallel search
- parallel coverage
- tree compression

Parallelizing the coverage

MODEL

```
is_malignant(A) :-
      'BIRADS_category'(A,b5),
      'Mass'(A,present),
      'Age'(A,age6570),
            previous_finding(A,B),
            'Calc_Punctate'(B,notPresent),
      'BIRADS_category'(B,b3).
```

DATA

Case #	Mass	. . .	Age	Class
Case 1	absent	. . .	51	benign
Case 2	present	. . .	65	malignant
. . .	. . .	. . .	. . .	. . .
Case n	Present	. . .	55	benign

Handling Multi-Relational Data Handling Multi-Relational Data Handling M

Parallelizing the coverage

MODEL

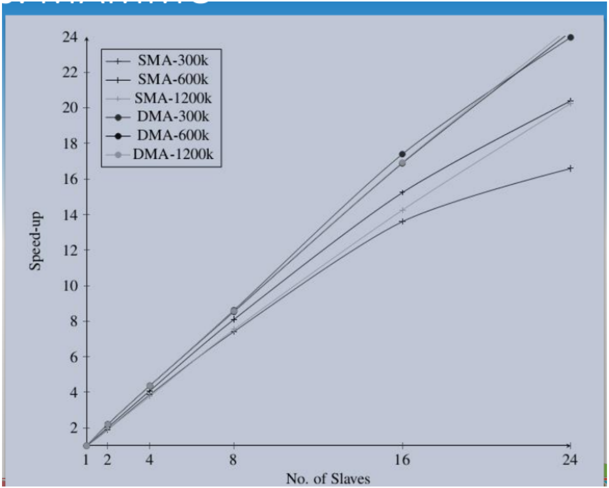
```
is_malignant(A) :-
        'BIRADS_category'(A,b5),
        'Mass'(A,present),
        'Age'(A,age6570),
            previous_finding(A,B),
            'Calc_Punctate'(B,notPresent),
        'BIRADS_category'(B,b3).
```

Apply hyp. (MAP)
and count (REDUCE)
benign and
malignant

DATA

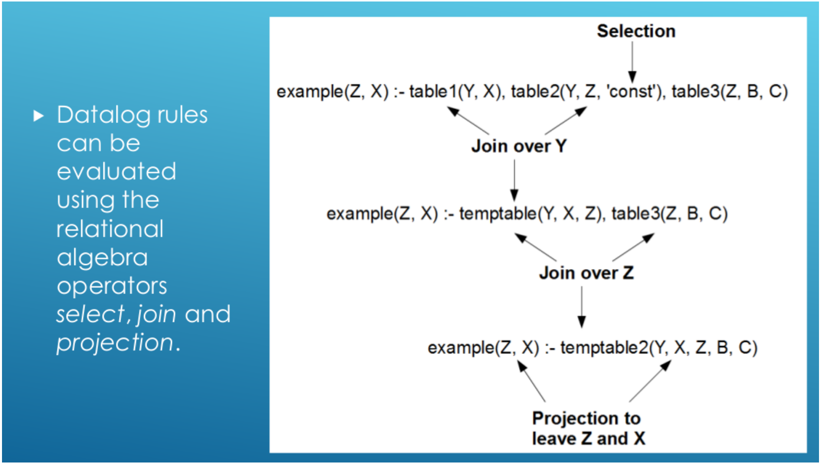
Case #	Mass	...	Age	Class
Case 1	absent	...	51	benign
Case 2	present	...	65	malignant
...	...	...	...	...
Case n	Present	...	55	benign

Parallelizing the coverage (mammo, 1.5GB)



 Handling Multi-Relational Data Handling Multi-Relational Data Handling M

Parallelizing the coverage: GPU-Datalog



Parallelizing the coverage: GPU-Datalog parsing

- Facts and rules are converted to numbers
- Each distinct string is assigned a unique id, equal strings are assigned the same id
- Capitalize on the GPU capability to process numbers

Parallelizing the coverage: GPU-Datalog preprocessing

- For each rule, specify which operations to perform and with which arguments
- Create small arrays for each operation, e.g.:
  `p(A,X,Y,Z), q(Z,X,B,C,Y).` → `[1,1,2,4,3,0]`
- Arrays are loaded in the shared memory of the GPU
  - allow each thread to work with the correct arguments without having to calculate them

Parallelizing the coverage: GPU-Datalog memory management

- Minimization of transfers between GPU memory and host memory by maintaining facts and rule results in GPU memory for as long as possible.
- To do so, maintain a list with information about each fact and rule result resident in GPU memory.
- Apply the Least Recently Used (LRU) page replacement algorithm.

Parallelizing the coverage: GPU-Datalog selection

- The size of the result in a selection is not known beforehand.
- Selection uses three different kernel executions:
  - first kernel: marks all the rows that satisfy the selection arguments with a value one.
  - second kernel: performs a prefix sum on the marks to determine the size of the results buffer and the location where each GPU thread must write the results.
  - last kernel: writes the results.

Parallelizing the coverage: GPU-Datalog and learning FOL

- BK and examples represented in first-order language (Prolog syntax)
- BK is parsed and sent to the GPUs with Examples
- While searching for a good hypothesis (on the host):
    - Coverage step (on the GPU) using bottom-up evaluation of the hypotheses
        - Parse the hypothesis
        - Send it to GPU
        - Perform database operations using BK and E
        - Return count to host

GPU Datalog: Experimental Evaluation

- Join over 4 tables of 5 million entries
- Transitive closure of a graph
- Same-generation benchmark

GPU Datalog: Experimental Evaluation

System	Join	Closure	Same-gen
YAP	125.20	3.51	0.02
XSB	287.81	4.08	0.03
MITRE	N/A	5.28	4.67
CUDA	**1.07**	**0.12**	0.02

Times in seconds

Parallelizing the coverage with GPU-Datalog: datasets

Application	BK	Examples
Carcino	21,303	297
hiv	2,310,575	48,766
omop	4,802,317	125,000
blog	5,124,092	50,000

Parallelizing the coverage with GPU-Datalog: datasets