# Set-up slide

Please, do not read yet.

# Declarative Modeling for Query Mining

## using Logic Programming

*Sergey Paramonov, Matthijs van Leeuwen, Marc Denecker and Luc De Raedt*

KU Leuven

August 22, 2015

# Outline

## Outline

# Main ideas of declarative data mining

- Formalize data mining tasks in logic
- Investigate current modeling possibilities and limits
- Evaluate these models in the current logic programming solvers (ASP)
- Propose/implement solver extensions
- Long-term: create efficient declarative mining languages

# Frequent query mining problem

**Given:**

- a relational database $D$,
- the entity of interest determining the *key* predicate,
- a frequency threshold $t$,
- a language bias $\mathcal{L}$ of logical queries of the form $key(X) \leftarrow b_1, ..., b_n$ defining $key/1$ ($b_i$'s are atoms).

**Find:** all queries $q \in \mathcal{L}$ s.t. $freq(q, D) \geq t$, where

$$freq(q, D) = |\{\theta \mid D \cup q \models key(X)\theta\}|$$

# Query mining example

- Relational graph database $D =$

$$\{edge(g_1, e_1, e_2),\ edge(g_1, e_2, e_3),\ edge(g_1, e_1, e_3),\ edge(g_2, e_1, e_2),\ edge(g_2, e_2, e_3),\ edge(g_2, e_1, e_3), \ldots\}$$

- Frequency threshold $t = 2$,
- The following query has frequency of 2, therefore it is frequent

$$key(K) \leftarrow edge(K, B, C), edge(K, C, D), edge(K, B, D)$$

# Important observations

- Data mining problems are essentially constraint satisfaction problems and optimization
- Data is often structured and relational
- Many of the interesting problems are NP-complete (and higher), perfect fit for SAT/ASP
- Many new problems are mathematical variations of known problems
- Use of solvers is very common in statistical learning (convex optimization for SVM etc)

# Why don't we just write some C-code?

## Key issues $U^4$

- unreliable: written by one or two researchers who are typically not professional developers
- unreadable: written a week or two before deadline
- unprovable: written without SQA
- unextendable: does not satisfy the *elaboration tolerance* principle

# Example: unreadable mining software

```
void TRSACT_shrink ( ARY *T, QUEUE *jump, long *p ){
  int ii, j, t, tt, v, vv;
  QUEUE_INT *jt, *jtt, *jq=jump->q, *jqq=jump->q+jump->end+1;
  long *pp=&p[jump->end+1],*q=&p[jump->end*2+2],*qq=&p[jump->end*2+2+T->num*2];
  QUEUE *Q = T->h;

  for ( t=0,jtt=jqq ; t<T->num ; t++ ){
    ii = Q[t].q[0];
    if ( pp[ii] == -1 ){ *jtt = ii; jtt++; }
    qq[t*2] = pp[ii];
    qq[t*2+1] = 0;
    pp[ii] = t;
  }

  for ( j=1 ; jtt>jqq ; j++ ){
    for ( jt=jq ; jtt>jqq ; ){
      jtt--;
      if ( *jtt == jump->end ) goto END2;
      t = pp[*jtt];
      pp[*jtt] = -1;
      v = -1;
      do{
        tt = qq[t*2];
        if ( v != qq[t*2+1] ){
          v = qq[t*2+1];
          vv = t;
          if ( tt<0 ) goto END2;
          if ( qq[tt*2+1] != v ) goto END1;
        }
        ii = Q[t].q[j];
        if ( p[ii] == -1 ){ *jt = ii; jt++; }
        q[t*2] = p[ii];
        p[ii] = t;
        q[t*2+1] = vv;
        END1:;
        t = tt;
```

# Core principles

- Data Mining = Modeling + Solving (De Raedt 2015)
- Focus on general principles and modeling rather than specific implementations
- Model reflects the mathematical properties of the task
- Itemsets mining has been investigated in CP framework (Guns, Nijssen, and De Raedt 2013; Negrevergne et al. 2013)
- Here we work with structured pattern mining

# Outline

# KR and Logic Programming (De Cat et al. 2014)

Map Coloring: find a map coloring function such that…

```
vocabulary V{
  type Color
  type Area
  Border(Area,Area)
  Coloring(Area):Color
}
theory T:V{
  Border(a1,a2) → Coloring(a1) ≠ Coloring(a2).
}
structure S:V{
  Area={Belgium;Holland;Germany;Luxembourg;Austria;Swiss;France
       }
  Color={Blue;Red;Yellow;Green}
  Border={(Belgium,Holland);(Belgium,Germany);
    (Belgium,Luxembourg);(Belgium,France);(Holland,Germany);
  ...}
}
```

# Graph Mining: Homomorphism existence

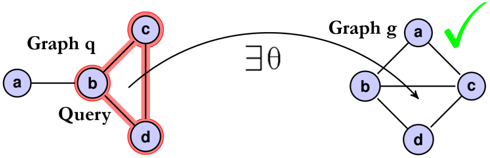
Find: subgraphs (indicated in red) of graph $q$ (called bottom) that can be homomorphically mapped to graph $g$ (fixed constant here)

Given:

$bedge(x, y), blabel(x) : l$ – edges and labels of $q$

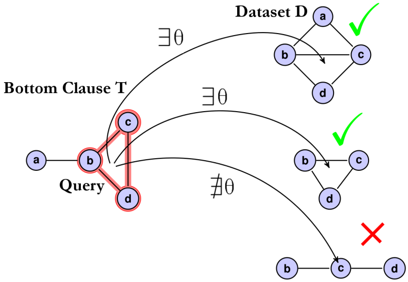
$edge(g, x, y), label(g, x) : l$ – edges and labels of $g$

Model exists iff $\theta :: node \mapsto node$ exists




$$
\begin{aligned}
inq(x) \wedge inq(y) \wedge bedge(x, y) &\implies edge(g, \theta(x), \theta(y)). \\
inq(x) \wedge blabel(x) = l &\implies label(g, \theta(x)) = l. \\
inq(x) \wedge inq(y) \wedge x \neq y &\implies \theta(x) \neq \theta(y).
\end{aligned}
$$

# Second-Order Model: Multiple Graphs




Multiple Graph Homomorphism Check:

$$\begin{aligned} homo(g) \iff \exists\theta : (&bedge(x,y) \land inq(x) \land inq(y) \implies edge(g,\theta(x),\theta(y)).\\ &inq(x) \land blabel(x) = y \implies label(g,\theta(x)) = y.\\ &x \neq y \implies \theta(x) \neq \theta(y)). \end{aligned}$$

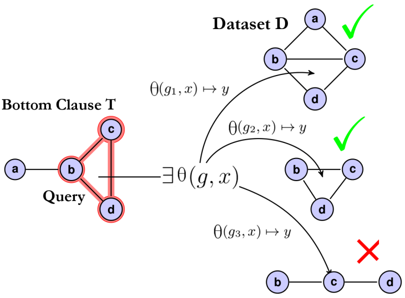
Frequency Constraint: $\#\{graph : homo(graph)\} \geq t.$

# Proposal: Second-Order Extension

$\psi(\bar{x})$, $\phi_i(\bar{x})$ – FOL formulae;
$f(\bar{x})$ – a function;
$\circ$ – logical connector ($\{\wedge, \vee, \longleftrightarrow, \rightarrow, \ldots\}$);
$Q$, $Q_i$ – sequences of quantifiers.

$$
\begin{aligned}
Q : \psi(\bar{x}) \circ [\neg]\exists_H f\,\big( & \\
& Q_1 : \phi_1(\bar{x}_1, f(\bar{y}_1)). \\
& \ldots \\
& Q_n : \phi_n(\bar{x}_n, f(\bar{y}_n)). \\
\big) &
\end{aligned}
$$

# First-Order Model: Multiple Graphs




Multiple Graph Homomorphism Check:

$$homo(g) \wedge inq(x) \wedge inq(y) \wedge bedge(x, y) \Longrightarrow edge(g, \theta(g, x), \theta(g, y)).$$

$$homo(g) \wedge inq(x) \iff \exists y : y = \theta(g, x).$$

$$homo(g) \wedge inq(x) \wedge inq(y) \wedge x \neq y \Longrightarrow \theta(g, x) \neq \theta(g, y).$$

$$homo(g) \wedge inq(x) \wedge blabel(x) = l \Longrightarrow label(g, \theta(g, x)) = l.$$

Frequency Constraint: $\#\{graph : homo(graph)\} \geq t.$

# Other computational challenges

- Canonicity – CoNP check
- Frequency anti-monotonicity – pruning the space of models
- Parallel search over homomorphisms and patterns – optimization and beyond
- Language bias construction – often domain specific

# We do not solve a problem but a *class* of problems

Elaboration principle:

*A small change in the problem should lead to a small change in the model*

**Connectedness constraint**

$$\{path(X,Y) \leftarrow inq(X) \wedge inq(Y) \wedge bedge(X,Y).$$
$$path(X,Y) \leftarrow \exists Z : inq(Z) \wedge path(X,Z) \wedge bedge(Z,Y) \wedge inq(Y).$$
$$path(Y,X) \leftarrow path(X,Y).\}$$
$$inq(X) \wedge inq(Y) \wedge X \neq Y \implies path(X,Y).$$

**Objective function: max-size constraint**

$$|\{X : inq(X)\}| \mapsto \max$$

**If then constraint**

$$bedge(a,b) \implies bedge(a',b')$$

# So is it a kind of magic?

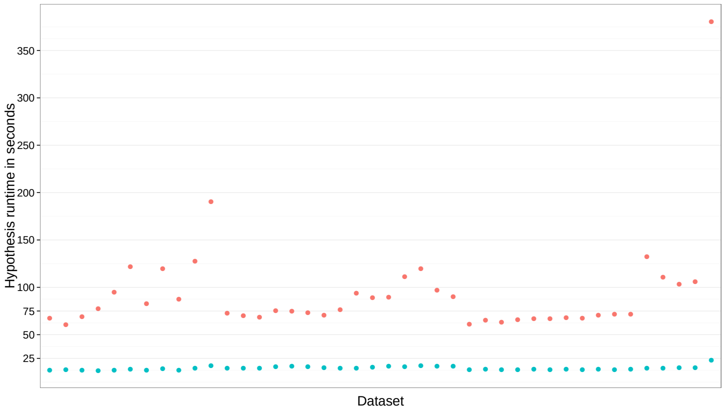
You might wonder why isn’t everyone using it all the time

# Outline

# Subsumption testing – sanity check

Comparison: declarative model ($\sim$ 10 lines of ASP) with a specialized Prolog $\theta$-subsumption engine Subsumer (Santos and Muggleton 2010)

Single $\theta$-subsumption test. IDP (red) and Subsumer (blue)
(avg time per hypothesis in seconds; the phase transition data)



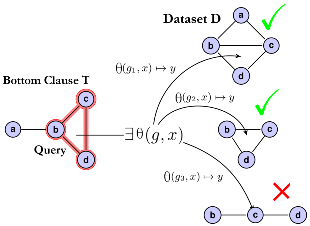
# Graph dataset description

Known datasets in the graph mining community.
Vertices, edges and labels are averaged per graph.

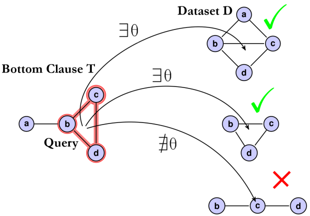
| **Name** | **Graphs** | **Vertices** | **Edges** | **Labels** |
|---|---|---|---|---|
| Mutagenesis | 230 | 26 | 27 | 9 |
| Enzymes | 600 | 33 | 124 | 3 |
| Toxinology | 417 | 26 | 26 | 22 |
| Bloodbarr | 413 | 21 | 23 | 9 |
| NCTRER | 232 | 19 | 20 | 9 |
| Yoshida | 265 | 20 | 23 | 9 |

# Graph Mining: runtime comparison (in s)

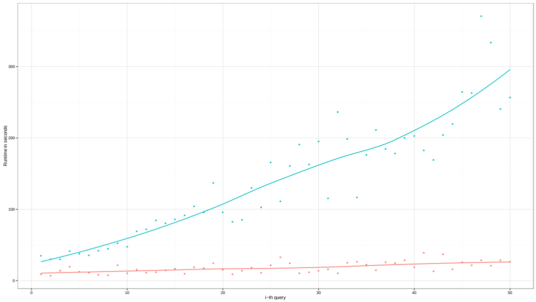
(a) IDP FOL Model (Blue)




(b) IDP Second-Order (Red)



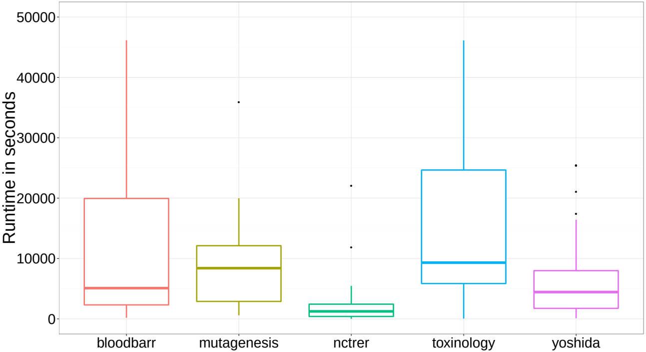

Frequent query enumeration; Yoshida dataset; y-axis runtime in seconds, x-axis i-th query.



# An open problem: structured pattern sets

No one knows how to search for patterns and homomorphisms efficiently at the same time, exploiting enumeration properties

Maximal size top-1 graph patterns. Runtime distribution.




There is no system yet that can solve the whole class in a declarative and principled way.

# Experimental summary

- Declarative models typically perform slower than specialized algorithms (by a factor or in an order of magnitude)
- Language extension is necessary for efficient computations
- Pattern sets, i.e. mining with optimization, requires new formalism and solving techniques
- Demonstrated performance allows declarative models to be used as prototypes

## Outline

# Historical analogy: SQL

- A long way in solver development e.g. SQL does not scale without indices, optimizers that involved three decades of research and IO-optimized data structures
- Modeling language: modification and extensions are necessary
- Application-driven: many particular features of the language reflect real life problems
- Family of language: SQL, NoSQL, newSQL etc
- Community: industry, developers and users participate in the evolution of the language

# Conclusions

- ASP (namely, IDP) can be applied to ILP tasks, such as query mining
- Experimental evidence shows that these models can serve as prototypes for new declarative mining languages
- Proposed a language extension and experimentally showed its effectiveness
- Provided a new computational and feature developing challenge for ASP solver community
- Demonstrated benefits of declarative models in mining tasks

# References


- Broes De Cat et al. "Predicate Logic as a Modelling Language: The IDP System". In: *CoRR* abs/1401.6312 (2014).
- Luc De Raedt. "Languages for Learning and Mining". In: *AAAI. 2015, Austin, Texas, USA*. 2015, pp. 4107–4111.
- Tias Guns, Siegfried Nijssen, and Luc De Raedt. "k-Pattern set mining under constraints". In: *Knowledge and Data Engineering, IEEE Transactions on* 25.2 (2013), pp. 402–418.
- Benjamin Negrevergne et al. "Dominance Programming for Itemset Mining". In: *ICDM 13*. 2013, pp. 557–566.
- Jose Santos and Stephen Muggleton. "Subsumer: A Prolog theta-subsumption engine". In: *ICLP Technical Communications*. Vol. 7. 2010, pp. 172–181.

Thank you for your attention