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

Introduction
  Declarative data mining
  The core problem of frequent query mining
  Motivation for declarative methods

Modeling
  Logic programming
  Second-Order Model
  First-Order Model

Experiments
  Subsumption testing
  Graph mining

Lesson learned
  Historical analogy: SQL
  Conclusions
Outline

Introduction
  Declarative data mining
  The core problem of frequent query mining
  Motivation for declarative methods

Modeling
  Logic programming
  Second-Order Model
  First-Order Model

Experiments
  Subsumption testing
  Graph mining

Lesson learned
  Historical analogy: SQL

Conclusions
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 $\text{key}(X) \leftarrow b_1, \ldots, b_n$ defining $\text{key}/1$ ($b_i$’s are atoms).

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

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

- Relational graph database \( D = \)
  \[
  \{ \text{edge}(g_1, e_1, e_2), \text{edge}(g_1, e_2, e_3), \text{edge}(g_1, e_1, e_3), \\
  \text{edge}(g_2, e_1, e_2), \text{edge}(g_2, e_2, e_3), \text{edge}(g_2, e_1, e_3), \ldots \} 
  \]

- Frequency threshold \( t = 2 \),

- The following query has frequency of 2, therefore it is frequent
  
  \( \text{key}(K) \leftarrow \text{edge}(K, B, C), \text{edge}(K, C, D), \text{edge}(K, B, D) \)
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
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++; }
                qq[t*2] = p[ii];
                p[ii] = t;
                qq[t*2+1] = vv;
                END1:;
                t = tt;
            } while ( tt<0 );
        } while ( v != -1 );
        if ( p[ii] == -1 ) { *jt = ii; jt++;
    }
}

END:;

END2:;
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

Introduction
  Declarative data mining
  The core problem of frequent query mining
  Motivation for declarative methods

Modeling
  Logic programming
  Second-Order Model
  First-Order Model

Experiments
  Subsumption testing
  Graph mining

Lesson learned
  Historical analogy: SQL

Conclusions
Map Coloring: find a map coloring function such that...

```prolog
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:

- \( \text{bedge}(x, y), \text{blabel}(x) : l \) – edges and labels of \( q \)
- \( \text{edge}(g, x, y), \text{label}(g, x) : l \) – edges and labels of \( g \)

Model exists iff \( \theta : node \mapsto node \) exists

\[
\begin{align*}
\text{inq}(x) \land \text{inq}(y) \land \text{bedge}(x, y) & \implies \text{edge}(g, \theta(x), \theta(y)). \\
\text{inq}(x) \land \text{blabel}(x) = l & \implies \text{label}(g, \theta(x)) = l. \\
\text{inq}(x) \land \text{inq}(y) \land x \neq y & \implies \theta(x) \neq \theta(y).
\end{align*}
\]
Multiple Graph Homomorphism Check:

\[ \text{homo}(g) \iff \exists \theta : (\text{bedge}(x, y) \land \text{inq}(x) \land \text{inq}(y) \implies \text{edge}(g, \theta(x), \theta(y))). \]

\[ \text{inq}(x) \land \text{blabel}(x) = y \implies \text{label}(g, \theta(x)) = y. \]

\[ x \neq y \implies \theta(x) \neq \theta(y). \]

Frequency Constraint: \[ \#\{\text{graph} : \text{homo}(	ext{graph})\} \geq t. \]
Proposal: Second-Order Extension

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

\[
Q : \psi(\bar{x}) \circ [\neg] \exists H f (Q_1 : \phi_1(\bar{x}_1, f(\bar{y}_1)). \\
\quad \ldots \\
Q_n : \phi_n(\bar{x}_n, f(\bar{y}_n)).
\]

First-Order Model: Multiple Graphs

Multiple Graph Homomorphism Check:

\[\text{homo}(g) \land \text{inq}(x) \land \text{inq}(y) \land \text{bedge}(x, y) \implies \text{edge}(g, \theta(g, x), \theta(g, y)).\]

\[\text{homo}(g) \land \text{inq}(x) \iff \exists y : y = \theta(g, x).\]

\[\text{homo}(g) \land \text{inq}(x) \land \text{inq}(y) \land x \neq y \implies \theta(g, x) \neq \theta(g, y).\]

\[\text{homo}(g) \land \text{inq}(x) \land \text{blabel}(x) = l \implies \text{label}(g, \theta(g, x)) = l.\]

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

\[
\{ \text{path}(X, Y) \leftarrow \text{inq}(X) \land \text{inq}(Y) \land \text{bedge}(X, Y). \}
\]

\[
\text{path}(X, Y) \leftarrow \exists Z : \text{inq}(Z) \land \text{path}(X, Z) \land \text{bedge}(Z, Y) \land \text{inq}(Y).
\]

\[
\text{path}(Y, X) \leftarrow \text{path}(X, Y).
\}

\[
\text{inq}(X) \land \text{inq}(Y) \land X \neq Y \implies \text{path}(X, Y).
\]

Objective function: max-size constraint

\[
|\{ X : \text{inq}(X) \}| \rightarrow \max
\]

If then constraint

\[
\text{bedge}(a, b) \implies \text{bedge}(a', b')
\]
So is it a kind of magic?

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

Introduction
  Declarative data mining
  The core problem of frequent query mining
  Motivation for declarative methods

Modeling
  Logic programming
  Second-Order Model
  First-Order Model

Experiments
  Subsumption testing
  Graph mining

Lesson learned
  Historical analogy: SQL

Conclusions
Subsumption testing – sanity check

Comparison: declarative model (~ 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Graphs</th>
<th>Vertices</th>
<th>Edges</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutagenesis</td>
<td>230</td>
<td>26</td>
<td>27</td>
<td>9</td>
</tr>
<tr>
<td>Enzymes</td>
<td>600</td>
<td>33</td>
<td>124</td>
<td>3</td>
</tr>
<tr>
<td>Toxinology</td>
<td>417</td>
<td>26</td>
<td>26</td>
<td>22</td>
</tr>
<tr>
<td>Bloodbarr</td>
<td>413</td>
<td>21</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>NCTRER</td>
<td>232</td>
<td>19</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>Yoshida</td>
<td>265</td>
<td>20</td>
<td>23</td>
<td>9</td>
</tr>
</tbody>
</table>
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

Introduction
  Declarative data mining
  The core problem of frequent query mining
  Motivation for declarative methods

Modeling
  Logic programming
  Second-Order Model
  First-Order Model

Experiments
  Subsumption testing
  Graph mining

Lesson learned
  Historical analogy: SQL

Conclusions
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
Broes De Cat et al. “Predicate Logic as a Modelling Language: The IDP System”. In: *CoRR* abs/1401.6312 (2014).


Thank you for your attention