

# Mining Queries

**Bart Goethals**

HIIT-BRU

University of Helsinki

<http://www.cs.helsinki.fi/bart.goethals/>

Work in progress

with Heikki Mannila and Jan Van den Bussche

## Outline

1. Patterns: from transactional data to relational data
2. Mining Conjunctive Queries: problem statement
3. General approach: the levelwise algorithm
4. Challenges
5. Interesting subclasses
6. A case study: solutions and more problems
7. Conclusions

## Patterns

- Sets (Agrawal et al. 1993)
- Trees (Zaki 2002)
- Graphs (Inokuchi et al. 2000, Kuramochi et al. 2001)
- Relational structures (Dehaspe et al. 1999)

⇒ **Transactional Data**

## Arbitrary Relational Database

*Likes(Drinker, Beer)*  
*Visits(Drinker, Bar)*  
*Serves(Bar, Beer)*

*Likes*

<i>Drinker</i>	<i>Beer</i>
Allen	Duvel
Allen	Trappist
Carol	Duvel
Bill	Duvel
Bill	Trappist
Bill	Jupiler

*Visits*

<i>Drinker</i>	<i>Bar</i>
Allen	Cheers
Allen	California
Carol	Cheers
Carol	California
Carol	Old Dutch
Bill	Cheers

*Serves*

<i>Bar</i>	<i>Beer</i>
Cheers	Duvel
Cheers	Trappist
Cheers	Jupiler
California	Duvel
California	Jupiler
Old Dutch	Trappist

Q: What does a pattern in a relational database look like?

A: A Query

Goal: Find all interesting queries

## Conjunctive Queries

```
select L1.Drinker  
from Likes as L1, Likes as L2  
where L1.Drinker = L2.Drinker  
and L1.Beer = 'Duvel'  
and L2.Beer = 'Trappist'
```

$$\pi_{x_1} \sigma_{x_1=x_3 \wedge x_2='Duvel' \wedge x_4='Trappist'} (Likes(x_1, x_2) \times Likes(x_3, x_4))$$
$$Q(x) :- likes(x_1, x_2), likes(x_3, x_4), x_1 = x_3, x_2 = 'Duvel', x_4 = 'Trappist'$$

## Where is the pattern?

Consider

$Q_1(x, y) :- \textit{likes}(x, \text{'Duvel'}), \textit{visits}(x, y).$

$Q_2(x, y) :- \textit{likes}(x, \text{'Duvel'}), \textit{visits}(x, y), \textit{serves}(y, \text{'Duvel'}).$

## Where is the pattern?

$Q_1 \Rightarrow Q_2$  (Note:  $Q_2 \subseteq Q_1$ )

if a person  $x$  that likes Duvel visits bar  $y$ ,  
then bar  $y$  serves Duvel

## Containment of Conjunctive Queries

- $Q_1 \subseteq Q_2$  if for every possible instance  $I$ :  $Q_1(I) \subseteq Q_2(I)$
- $Q_1 \equiv Q_2 \Leftrightarrow Q_1 \subseteq Q_2$  and  $Q_2 \subseteq Q_1$
- CQ containment is decidable

## Problem Statement

- Conjunctive Queries as Patterns
- **support** :=  $|Q(\mathbf{I})|$  (duplicate elimination)
- **Association Rule**  $Q_1 \Rightarrow Q_2$ , with  $Q_2 \subseteq Q_1$
- **Frequent** if  $|Q_2(\mathbf{I})| \geq \textit{minsup}$
- **Confident** if  $\frac{|Q_2(\mathbf{I})|}{|Q_1(\mathbf{I})|} \geq \textit{minconf}$
- **Challenge:** Find all frequent and confident association rules

## Example Rule

$Q_1(x, y) :- \textit{likes}(x, \text{'Duvel'}), \textit{visits}(x, y).$

$Q_2(x, y) :- \textit{likes}(x, \text{'Duvel'}), \textit{visits}(x, y), \textit{serves}(y, \text{'Duvel'}).$

$$Q_1 \Rightarrow Q_2$$

if a person  $x$  that likes Duvel visits bar  $y$ ,  
then bar  $y$  serves Duvel

## Example Rule 2

$Q_1(x) :- \text{likes}(x, \text{'Duvel'}), \text{visits}(x, y).$

$Q_2(x) :- \text{likes}(x, \text{'Duvel'}), \text{visits}(x, y), \text{serves}(y, \text{'Duvel'}).$

$$Q_1 \Rightarrow Q_2$$

if a person  $x$  that likes Duvel visits a bar,  
then person  $x$  visits a bar that serves Duvel

### Example Rule 3

$Q_1(y) :- \text{likes}(x, \text{'Duvel'}), \text{visits}(x, y).$

$Q_2(y) :- \text{likes}(x, \text{'Duvel'}), \text{visits}(x, y), \text{serves}(y, \text{'Duvel'}).$

$$Q_1 \Rightarrow Q_2$$

if a bar  $y$  has a visitor that likes Duvel,  
then bar  $y$  serves Duvel

## General 2-phased Approach

Based on general levelwise algorithm (Mannila and Toivonen, 1997)

- A set of patterns  $\mathcal{L}$  and a specialization relation (partial order)  $\preceq$  on  $\mathcal{L}$ .
- A selection predicate  $q$  on  $\mathcal{L}$  that is monotone with respect to  $\preceq$ , i.e., for all  $\mathcal{D}$ , if  $\varphi \preceq \psi$  and  $q(\mathcal{D}, \varphi)$ , then  $q(\mathcal{D}, \psi)$ .
- Find  $Th(\mathcal{L}, \mathcal{D}, q) := \{\phi \in \mathcal{L} \mid q(\mathcal{D}, \phi) \text{ is true}\}$

## The levelwise algorithm

```
 $C_1 := \{\phi \in \mathcal{L} \mid \nexists \gamma \in \mathcal{L} : \phi \prec \gamma\};$   
 $i := 1;$   
while  $C_i \neq \emptyset$  do  
  // Candidate evaluation  
   $\mathcal{F}_i := \{\phi \in C_i \mid q(\mathcal{D}, \phi)\};$   
  // Candidate generation  
   $C_{i+1} := \{\phi \in \mathcal{L} \mid \forall \gamma, \phi \prec \gamma : \gamma \in \bigcup_{j \leq i} \mathcal{F}_j\} \setminus \bigcup_{j \leq i} C_j;$   
   $i := i + 1$   
end while  
return  $\bigcup_{j < i} \mathcal{F}_j;$ 
```

## Challenges

1. Generate initial set  $C_1$
2. Generate the smallest possible superset of  $C_{i+1}$  without duplicates
3. Check whether all generalizations are in  $\bigcup_{j \leq i} \mathcal{F}_j$
4. Efficiently evaluate  $q$  on all elements in  $C_{i+1}$ .

## Most general conjunctive queries?

- Do not exist in general
- (cartesian product)

## Most specific conjunctive queries?

- Do not necessarily exist
- (e.g. cycles)

## **Solution?**

- Study subclasses
- What are interesting subclasses?

## Option 1

- Only allow a maximum number of atoms in the body of a CQ, *maxatom*
- $C_1$  consists of all queries with *maxatom* atoms in the body, and all variables in the head

## Option 2

- Only allow a maximum number of variables in the head
- $C_1$  consists of all queries with *maxhead* atoms in the body using different variables (cartesian product)

## Problem

- Consider the simple case of a single binary relation (graph)
- Equivalence? Isomorphism?
- Focus on acyclic hypertrees first, ...

## Option 3

- Only allow each relation to occur in the body at most once!

$$\pi_X \sigma_\phi (R_1 \times \cdots \times R_n)$$

- Let  $V$  be the set of all variables occurring in  $(R_1 \times \cdots \times R_n)$
- $X \subseteq V$
- $\phi$  is a conjunction of equalities, i.e. a partition of  $V$
- Equivalence is easy!

## Expressiveness of Option 3

- **Inclusion dependencies**

Every inclusion dependency

$$R_i : A_1, \dots, A_n \subseteq R_j : B_1, \dots, B_n$$

will be found since it can be expressed by an association rule over conjunctive queries as follows:

$$\pi_{A_1, \dots, A_n}(R_i) \Rightarrow \pi_{A_1, \dots, A_n} \sigma_{A_1=B_1, \dots, A_n=B_n}(R_i \times R_j).$$

## Expressiveness of Option 3

- **Functional dependencies**

Also, every functional dependency

$$R : X \rightarrow Y$$

can be expressed by an association rule as follows:

$$\pi_{X,Y}(R) \Rightarrow \pi_X(R).$$

## Expressiveness of Option 3

- **Frequent Itemset Association Rules**

- Each relation is an item and each tuple is a transaction identifier (vertical layout, tidlist)
- Then, every non-empty frequent itemset  $\{A_i, \dots, A_j\}$  can be represented by the conjunctive query

$$\pi_{A_i} \sigma_{A_i=A_{i+1} \wedge \dots \wedge A_{j-1}=A_j} (A_1 \times \dots \times A_n),$$

with  $1 \leq i \leq j \leq n$ .

## Efficient algorithm

- Generate all subsets of  $V$  starting with  $V$  (most general)
- Generate all partitions of  $V$  starting with all blocks of size 1 (most general)
- A lot of optimizations possible!
- Problem: still a huge number of queries will be generated (Cartesian Product makes support threshold superfluous)

## Solutions and more problems

- Disallow Cartesian Product (allow only Equi-joins)
- Queries containing CP can easily be computed anyway (product of two EJ queries)
- Most general queries?
- Given  $n$  relations, we can generate more than  $2^n$  possible equi-join queries
- Only allow the variables of a single relation in the head (this still allows FS's, ID's, and AR's over IS's and many more)

## Efficient evaluation

- For many queries containing only a single equation, we can store the list of tuple identifiers that satisfy the query
- Contained queries with additional equalities can be evaluated by taking intersections
- Example

$$Q_1(x, y) \text{ :- } \textit{likes}(x, \text{'Duvel'}), \textit{visits}(x, y).$$

$$Q_2(x, y) \text{ :- } \textit{visits}(x, y), \textit{serves}(y, \text{'Duvel'}).$$

$$Q_3(x, y) \text{ :- } \textit{likes}(x, \text{'Duvel'}), \textit{visits}(x, y), \textit{serves}(y, \text{'Duvel'}).$$

## Conclusions

1. Efficiently generating candidate queries up to equivalence is a hard problem.
2. Consider other subclasses of conjunctive queries, acyclic queries
3. Allow constraints to limit the number of patterns
4. Other uses of conjunctive queries as patterns? e.g. classification, clustering
5. Efficient evaluation of count queries!

## **A perspective on Inductive Databases**

- What are we talking about?
- How do you want an IDB to look like?

## Option 1

a relational database with some additional plugins

- a plugin for association rules
- one for classifiers
- one for clustering
- etc.

## Option 1: Question

How would these plugins look like?

- query language extensions? e.g. Mine Rule?
- virtual tables?
- 'nuggets'? e.g. Clementine

## Option 2

a more general kind of database (?)

## A Conjunctive Query Mining Perspective

*There is no such thing as real “discovery.”  
Discovery is just a matter of the expressive power of a query  
language.”* (Imielinski and Mannila, 1996)

1. Each query is a potential “discovery” .
2. Automated discovery finds interesting queries.
3. Find a meta-query language with a focus on data mining.

**some definitions  
(and a lot of hand waving)**

A *relational IDB* schema  $\mathcal{L}$  is a relational language defined over a relational schema  $\mathcal{S}$ .

A *relational IDB* instance  $\mathbf{I}$  consists of a set of queries from  $\mathcal{L}$  over the relational database instance  $\mathbf{I}$  ( $\sim$  views?).

## **RIDL: a Relational Inductive Database Language**

A query in RIDL is a mapping from an rIDB to an rIDB.

## Examples

A query in RIDL is able to

- find association rules (over the queries in  $\mathcal{L}$ )  
e.g. “Find all couples of queries  $Q_1 \subseteq Q_2$  such that  $|Q_1| \geq \textit{minsup}$   
and  $\frac{|Q_1|}{|Q_2|} \geq \textit{minconf}$ .”
- find a classifier (as a set of association rules)
- find a clustering (as several sets of queries)
- find dependencies

## RIDL

Note that a query from RIDL can only select queries from **L**

**L** contains the 'identity' queries

## RIDL: primitives

- $\sigma, \times, \pi, -$
- quantifiers over queries
- $\subseteq$
- syntactical constraints
- aggregates on queries

## A lot of existing theory and practice

- ILP
- Meta query languages (PODS, LICS, etc.)
- Conjunctive query evaluation (SIGMOD, PODS, VLDB, etc.)
- ...

**The End (for now)**