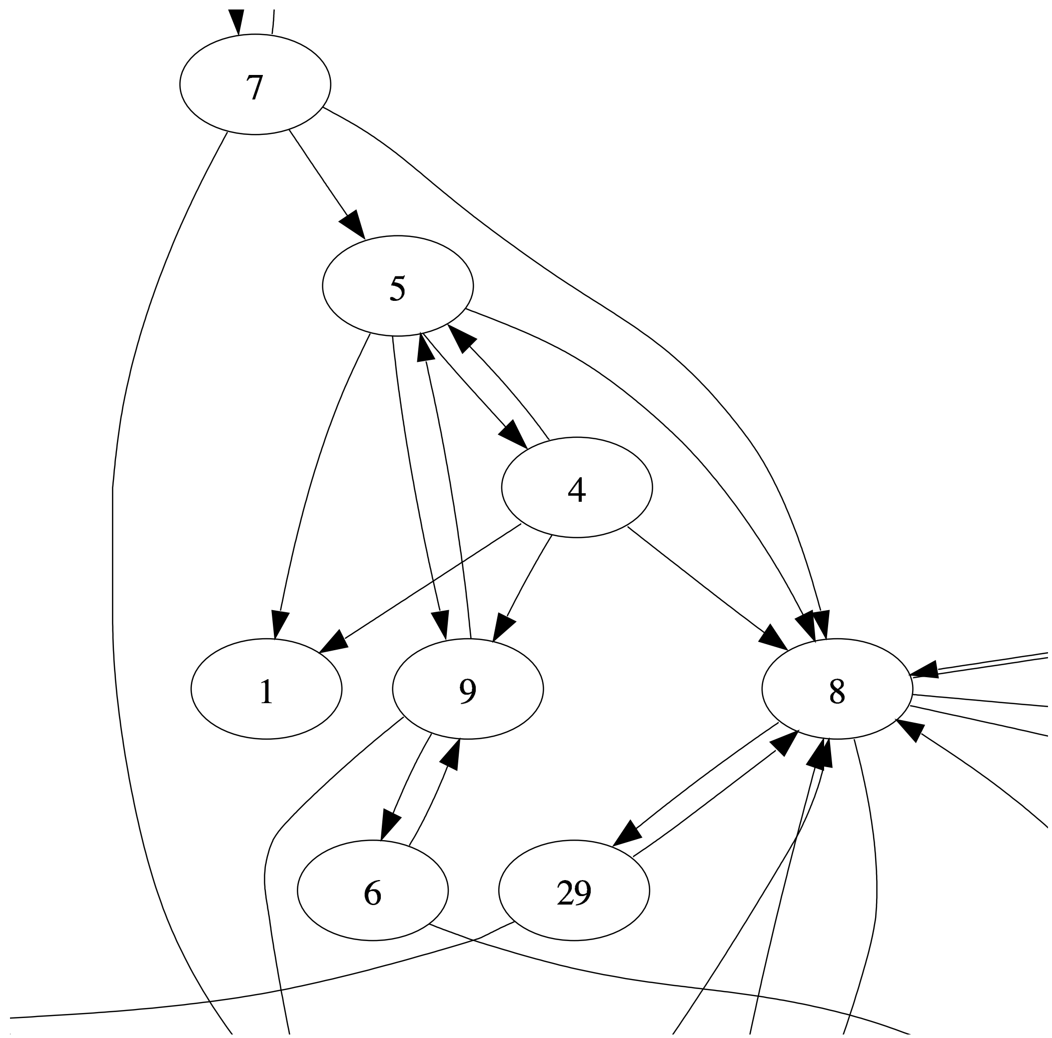
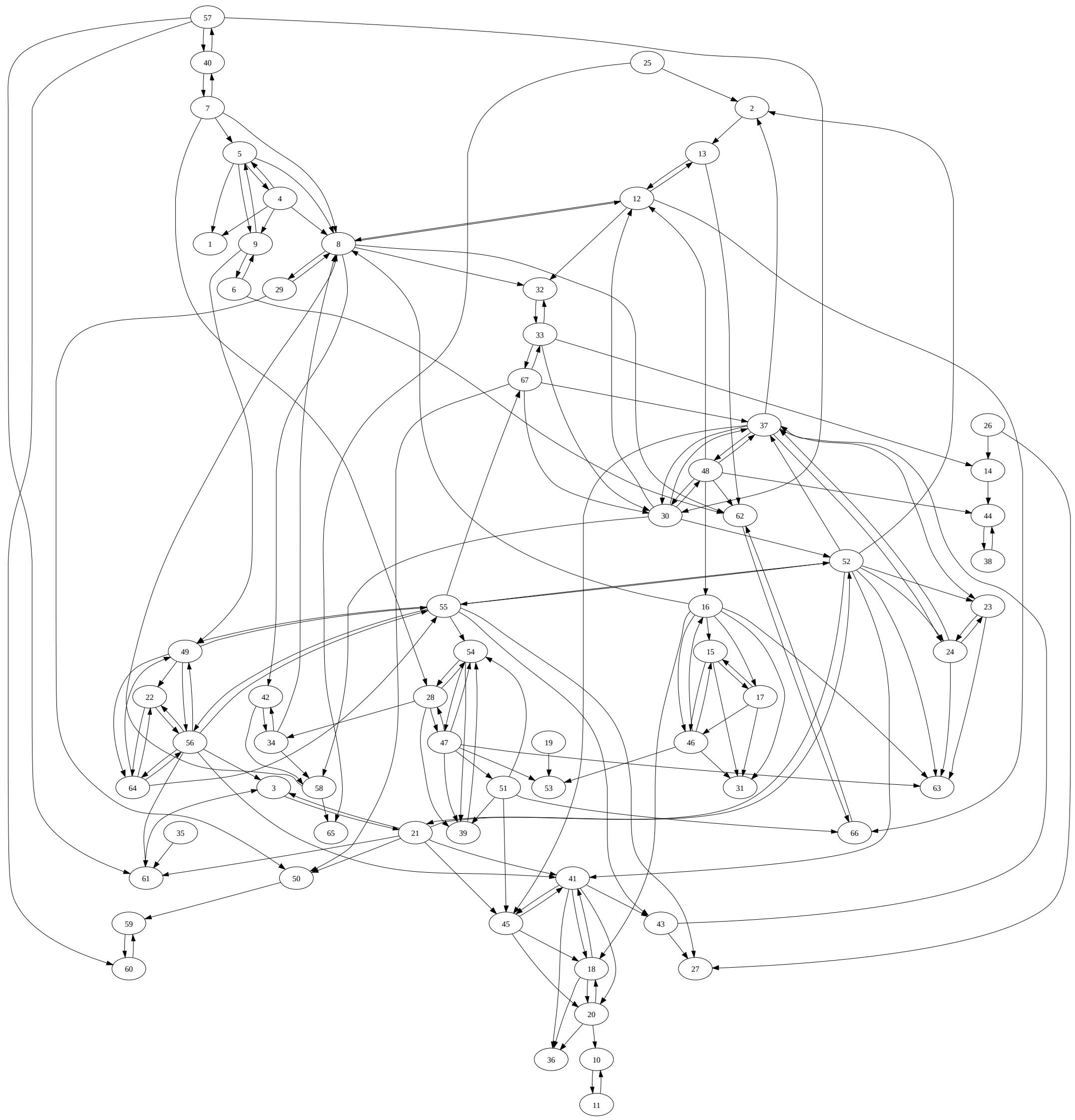
Mining frequent tree-conjunctive queries in large graphs

Jan Van den Bussche, Limburgs Universitair Centrum
joint with
Bart Goethals, University Antwerpen
Eveline Hoekx, Limburgs Universitair Centrum

Graph data

A (directed) graph over a set of nodes N is a set G of edges: ordered pairs (i, j) with $i, j \in N$.


7
5
4
1
9
8
6
29


57
40
25
7
2
5
13
4
12
1
9
8
6
29
32
33
67
37
26
48
14
30
62
44
52
38
55
16
23
49
54
15
24
22
42
28
17
56
34
47
19
46
64
3
58
51
53
31
63
35
21
39
65
66
61
50
41
59
45
43
18
60
27
20
36
10
11

Graphs are everywhere!

- data structures
- hypertext documents
- social networks
- protein structures
- transportation networks
- World Wide Web
- food webs
- ...

Mining for patterns in graphs

Q1. Given a class $\mathcal{C}$ of graphs, which patterns typically occur frequently in graphs in $\mathcal{C}$?

Q1 has become a very hot topic over the past years (Science, Nature)

To do Q1 well we must at least be able to do:

Q2. Given a graph G, which patterns occur frequently in G?

This can be interesting in itself. We will focus on Q2.

Q3. Given a collection $\mathcal{C}$ of graphs, which patterns frequently occur in graphs in $\mathcal{C}$?

Mining for patterns in graphs

Q1. Given a class $\mathcal{C}$ of graphs, which patterns typically occur frequently in graphs in $\mathcal{C}$?

Q1 has become a very hot topic over the past years (Science, Nature)

To do Q1 well we must at least be able to do:

Q2. Given a graph G, which patterns occur frequently in G?

This can be interesting in itself. We will focus on Q2.

Q3. Given a collection $\mathcal{C}$ of graphs, which patterns frequently occur in graphs in $\mathcal{C}$?

Examples of patterns

$$\begin{array}{c} x \\ \downarrow \\ 8 \end{array}$$

frequency: $\#\{x \mid (x, 8) \in G\}$

Examples of patterns

$$\begin{array}{c} 0 \\ \downarrow \\ x \end{array}$$

frequency: $\#\{x \mid (0, x) \in G\}$

Examples of patterns

$$\begin{array}{c} x \\ \downarrow \\ 8 \\ \downarrow \\ y \end{array}$$

frequency: $\#\{(x,y) \mid (x,8) \in G \wedge (8,y) \in G\}$

Existential nodes in patterns

$$\begin{tikzcd} & \exists \arrow[dl] \arrow[dr] & \\ x & & 8 \end{tikzcd}$$

frequency: $\#\{x \mid \exists z : (z, x) \in G \wedge (z, 8) \in G\}$

Existential nodes in patterns

$$0 \longrightarrow \exists \longrightarrow \exists \longrightarrow x$$

frequency:

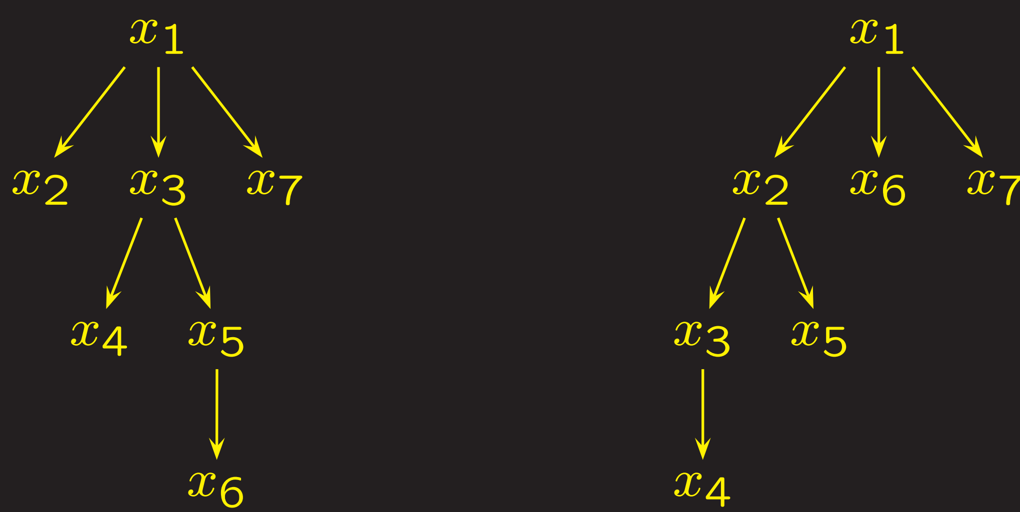
$$\#\{x \mid \exists z_1, z_2 : (0, z_1) \in G \wedge (z_1, z_2) \in G \wedge (z_2, x) \in G\}$$

Our work

Efficiently mine all frequent tree-shaped patterns in a large graph

- Incremental in size of patterns
- Tree-shaped only, but with existential nodes
- Database approach: on top of SQL
- Mining results stay in database
- Provable optimality properties
- Underlying theory of conjunctive queries

Avoiding isomorphic trees



$\Rightarrow$ Generate only canonical trees: “left-deep”

Generating all canonical trees

A. If T is canonical and n is its last node, then $T - n$ is also canonical.

$\Rightarrow$ Generate canonical trees incrementally by size

B. All canonical extensions of a given canonical tree can be generated efficiently.

- All this is known for a long time!

- For general graph shapes, no such efficient canonization is known.

Generating all canonical trees

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \end{tikzcd}$$

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[r] & x_3 \end{tikzcd}$$

$$\begin{tikzcd} & x_3 \\ x_1 \arrow[ur] \arrow[dr] & \\ & x_2 \end{tikzcd}$$

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[r] & x_4 \end{tikzcd}$$

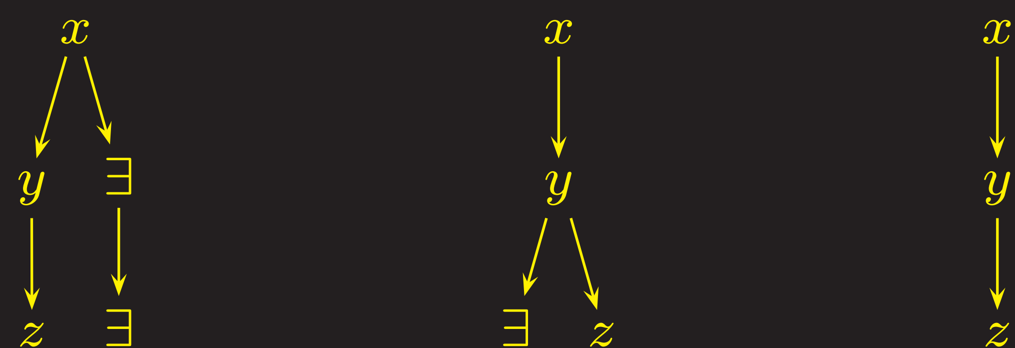
$$\begin{tikzcd} & & x_4 \\ x_1 \arrow[r] & x_2 \arrow[ur] \arrow[dr] & \\ & & x_3 \end{tikzcd}$$

$$\begin{tikzcd} & x_4 & \\ x_1 \arrow[ur] \arrow[dr] & & \\ & x_2 \arrow[r] & x_3 \end{tikzcd}$$

$$\begin{tikzcd} & x_4 \\ x_1 \arrow[ur] \arrow[r] \arrow[dr] & x_3 \\ & x_2 \end{tikzcd}$$

$\cdots$

Equivalent patterns



- Two patterns are equivalent if they become identical after removal of redundancies.

$\Rightarrow$ Efficient redundancy check needed

Redundancy characterization

A pattern has a redundancy if and only if contains the following pattern:

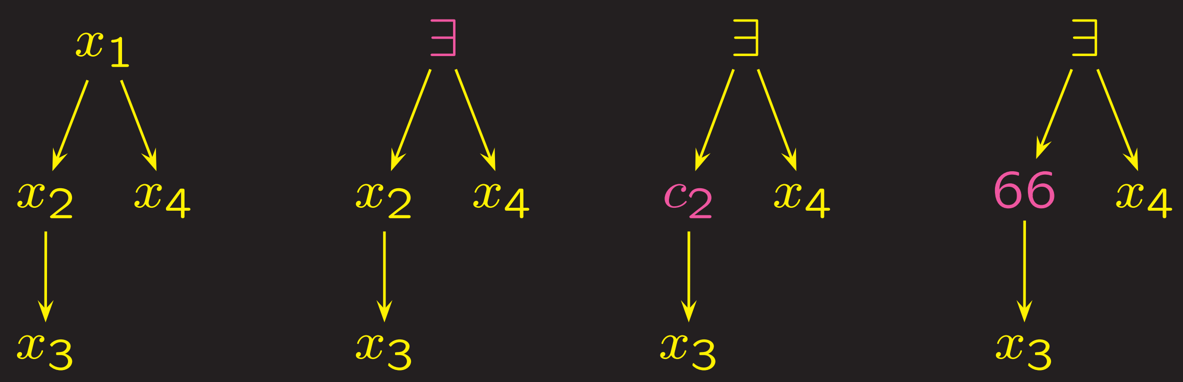
$$
\begin{tikzcd}
 & x \arrow[dl] \arrow[dr] & & & \\
T & & \exists \arrow[dr] & & \\
 & & & \cdots \arrow[dr] & \\
 & & & & \exists
\end{tikzcd}
$$

where subtree T is at least as deep as the $\exists$-path.

- Efficiently checkable
- For general graph patterns, redundancy checking is NP-complete

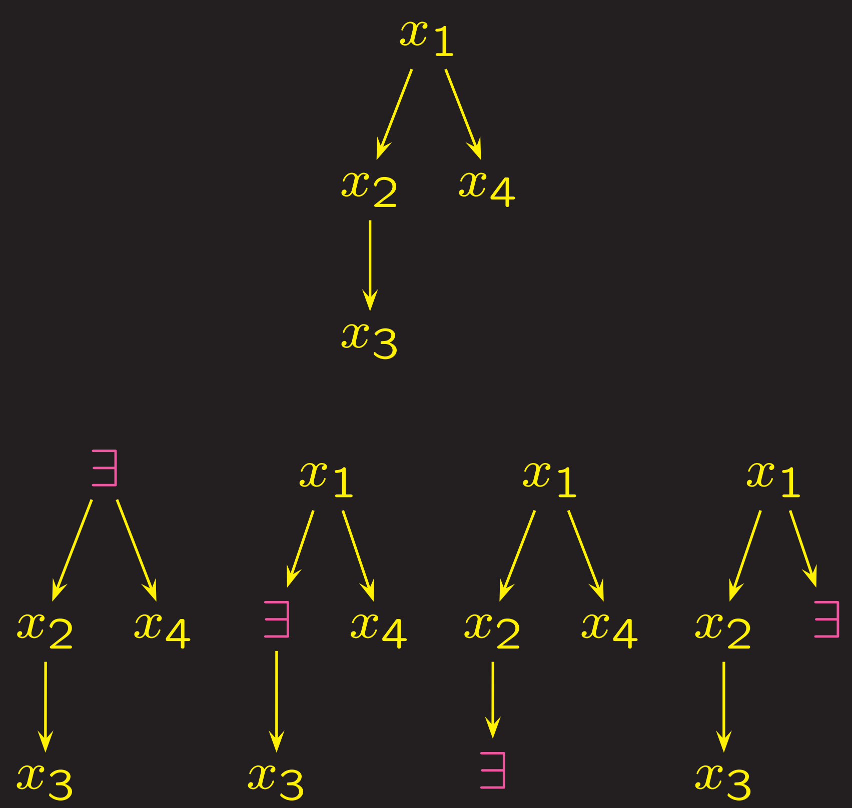
Overall approach

1. Generate canonical trees of increasing size
2. Generate (non-redundant) projections
3. Generate selections
4. Count all instantiations with one SQL expression



canon. tree projection selection instantiation

Levelwise generation of projections



Levelwise generation of projections

$$
\begin{tikzcd}[column sep=small]
 & x_1 \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
$$

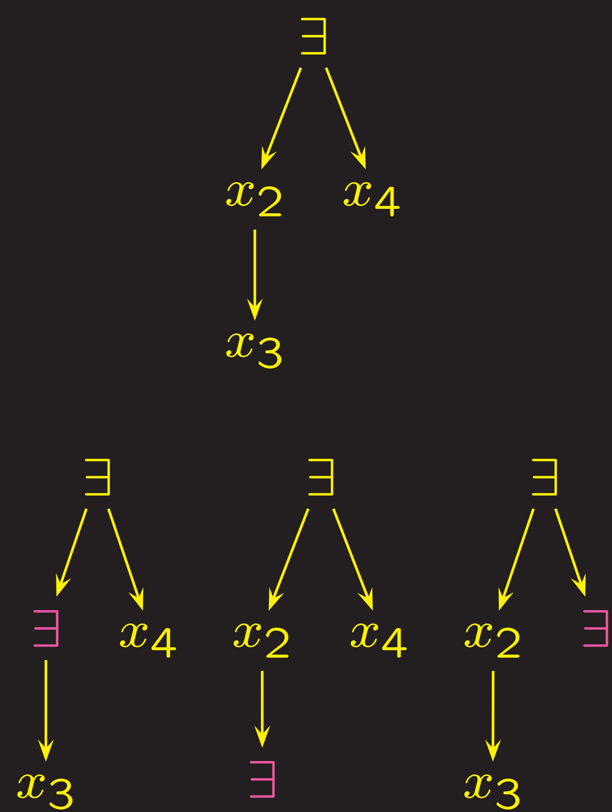
$$
\begin{tikzcd}[column sep=small]
 & \exists \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
\quad
\begin{tikzcd}[column sep=small]
 & x_1 \arrow[dl] \arrow[dr] & \\
\exists \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
\quad
\begin{tikzcd}[column sep=small]
 & x_1 \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
\exists & &
\end{tikzcd}
\quad
\begin{tikzcd}[column sep=small]
 & x_1 \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & \exists \\
x_3 & &
\end{tikzcd}
$$

Levelwise generation of projections

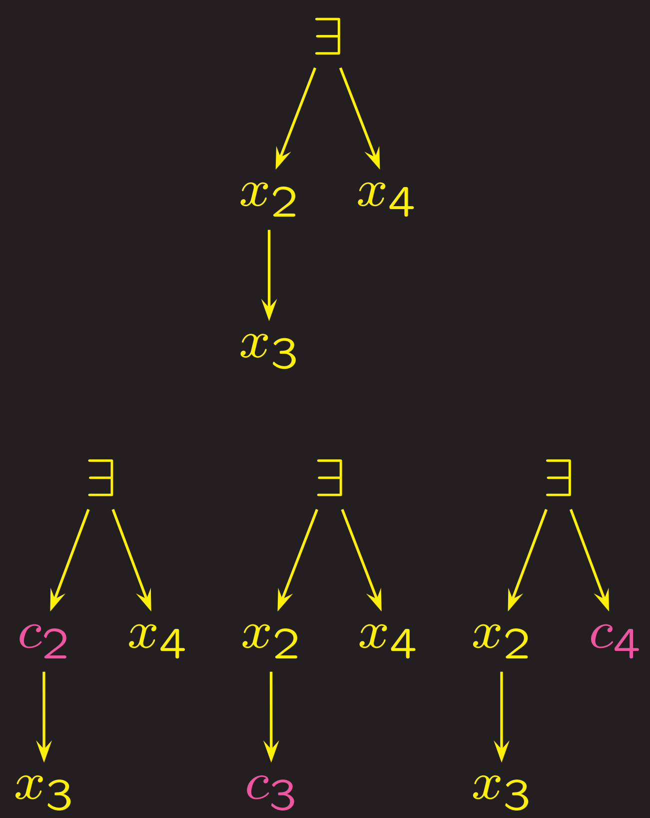
$$
\begin{tikzcd}
& x_1 \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
$$

$$
\begin{tikzcd}
& \exists \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
\qquad
\begin{tikzcd}
& x_1 \arrow[dl] \arrow[dr] & \\
\exists \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
\qquad
\begin{tikzcd}
& x_1 \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
\exists & &
\end{tikzcd}
$$

Levelwise generation of projections



Levelwise generation of selections



Levelwise generation of selections

$$
\begin{tikzcd}
& \exists \arrow[dl] \arrow[dr] & \\
x_2 \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
$$

$$
\begin{tikzcd}
& \exists \arrow[dl] \arrow[dr] & & & \exists \arrow[dl] \arrow[dr] & & & \exists \arrow[dl] \arrow[dr] & \\
c_2 \arrow[d] & & x_4 & x_2 \arrow[d] & & x_4 & x_2 \arrow[d] & & c_4 \\
x_3 & & & c_3 & & & x_3 & &
\end{tikzcd}
$$

Levelwise generation of selections

$$
\begin{tikzcd}
 & \exists \arrow[dl] \arrow[dr] & \\
c_2 \arrow[d] & & x_4 \\
x_3 & &
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & \exists \arrow[dl] \arrow[dr] & & & \exists \arrow[dl] \arrow[dr] & \\
c_2 \arrow[d] & & x_4 & c_2 \arrow[d] & & c_4 \\
c_3 & & & x_3 & &
\end{tikzcd}
$$

Pattern tables

$$
\begin{tikzcd}
 & \exists \arrow[dl] \arrow[dr] & \\
c_2 \arrow[d] & & c_4 \\
x_3 & &
\end{tikzcd}
$$

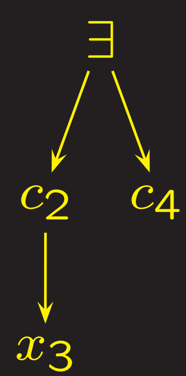
c_2	c_4	count
66	77	20
66	78	24
	$\vdots$	

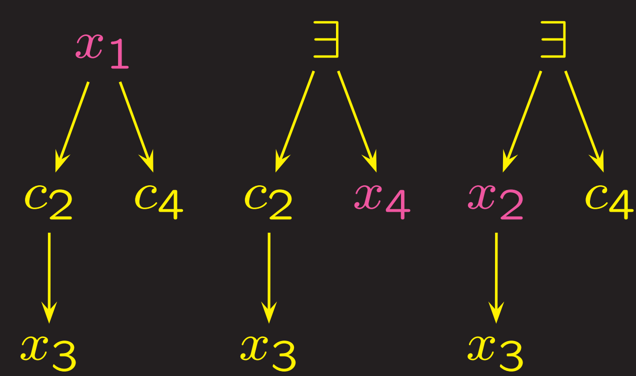
In each row of the table,

$$\text{count} = \#\{x_3 \mid \exists x_1 : (x_1, c_2) \in G \wedge (c_2, x_3) \in G \wedge (x_1, c_4) \in G\}$$

Computing the pattern table in SQL

1. Initalize with natural join of parent pattern tables

parent patterns of 

are 

2. Compute counts with one SQL expression

SQL expression

Graph G stored in table `G(from,to)`

$$
\begin{tikzcd}
 & c_4 & \\
\exists \arrow[ur] \arrow[dr] & & \\
 & c_2 \arrow[r] & x_3
\end{tikzcd}
$$

```
select tab.c2, tab.c3, count(*)
from (select table.c2, table.c3, G3.to
      from G G2, G G3, G G4, table
      where G2.from=G4.from and G2.to=G3.from
        and G2.to=table.c2 and G4.to=table.c3)
```

Optimality properties

1. We never investigate distinct but equivalent patterns

2. We never investigate a pattern subsumed by another pattern that we already know to be infrequent

- Incremental and levelwise approach

- Subsumption for general graph patterns is NP-complete

Current work

- Database performance tuning
- Apply to real-world graph data
- Pattern browsing
- Association rules

$$\text{pattern} \quad \begin{array}{c} x \\ \downarrow \\ 8 \end{array} \quad \text{versus} \quad \text{rule} \quad \begin{array}{c} x \\ \downarrow \\ \exists \end{array} \Rightarrow \begin{array}{c} x \\ \downarrow \\ 8 \end{array}$$