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
A (directed) graph over a set of nodes $N$ is a set $G$ of edges: ordered pairs $(i, j)$ with $i, j \in N$. 
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 $C$ of graphs, which patterns typically occur frequently in graphs in $C$?

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 $C$ of graphs, which patterns frequently occur in graphs in $C$?
Mining for patterns in graphs

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

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 $C$ of graphs, which patterns frequently occur in graphs in $C$?
Examples of patterns

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

\[
\# \{x \mid (0, x) \in G\}
\]
Examples of patterns

frequency:  \#\{(x, y) \mid (x, 8) \in G \land (8, y) \in G\}
Existential nodes in patterns

\[
\exists \bigg\{ x \mid \exists z : (z, x) \in G \land (z, 8) \in G \bigg\}
\]
Existential nodes in patterns

\[ 0 \rightarrow \exists \rightarrow \exists \rightarrow x \]

frequency:

\[ \#\{ x \mid \exists z_1, z_2 : (0, z_1) \in G \land (z_1, z_2) \in G \land (z_2, x) \in G \} \]
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

⇒ 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

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \]

\[ x_1 \rightarrow x_2 \leftarrow x_4 \]

\[ x_1 \leftarrow x_2 \rightarrow x_3 \rightarrow x_4 \]

\[ x_1 \leftarrow x_2 \leftarrow x_3 \rightarrow x_4 \]

\[ x_1 \leftarrow x_2 \leftarrow x_3 \rightarrow x_4 \]

\[ x_1 \leftarrow x_2 \leftarrow x_3 \leftarrow x_4 \]

\[ x_1 \leftarrow x_2 \leftarrow x_3 \leftarrow x_4 \]
Equivalent patterns

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

⇒ Efficient redundancy check needed
Redundancy characterization

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

\[
\begin{align*}
\exists \to \Bigarrowdown & \ x \\
\to & \ T \\
\to & \ \exists \to \ldots \to \exists
\end{align*}
\]

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
Levelwise generation of projections
Levelwise generation of projections
Levelwise generation of projections
Levelwise generation of projections

\[
\exists \ x_2 \ x_4 \\
\exists \ x_3 \\
\exists \ x_4 \ x_2 \ x_4 \ x_2 \\
\exists \ x_3 \ x_3
\]
Levelwise generation of selections

∃

x_2 \quad x_4

∃

x_3

∃

c_2 \quad x_4

x_2 \quad x_4

x_2 \quad c_4

c_3

x_3

x_3
Levelwise generation of selections

∃

x_2  x_4

∃

x_3

∃

c_2  x_4

x_3

∃

x_2  x_4  x_2  c_4

c_3  x_3  x_3
Levelwise generation of selections
In each row of the table,

\[
\text{count} = \# \{ x_3 \mid \exists x_1 : (x_1, c_2) \in G \land (c_2, x_3) \in G \land (x_1, c_4) \in G \}.
\]
Computing the pattern table in SQL

1. Initialize with natural join of parent pattern tables

parent patterns of \( \exists c_2 \times x_3 \times c_4 \) are

\( \exists c_2 \times x_3 \times c_4 \times x_1 \)

\( \exists c_2 \times x_3 \times c_4 \times x_2 \)

\( \exists c_2 \times x_3 \times c_4 \times x_3 \)

2. Compute counts with one SQL expression
Graph $G$ stored in table $G$(from,to)

\[ \exists \quad c_4 \quad c_2 \quad x_3 \]

```
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

\[
\begin{align*}
\text{pattern} & \quad x \quad \text{versus rule} \quad x \\
8 & \quad \exists \quad 8
\end{align*}
\]