Query Minimization

**Definition** A conjunctive query $q$ is minimal if for every other conjunctive query $q'$ s.t. $q \equiv q'$, $q'$ has at least as many predicates (‘subgoals’) as $q$.

Are these queries minimal?

$$q(x) : R(x,y), R(y,z), R(x,x)$$

$$q(x) : R(x,y), R(y,z), R(x,Alice)$$

Query Minimization

- Query minimization algorithm
  - Choose a subgoal $g$ of $q$
  - Remove $g$: let $q'$ be the new query
  - We already know $q \subseteq q'$ (why?)
  - If $q' \subseteq q$ then permanently remove $g$
- Notice: the order in which we inspect subgoals doesn’t matter

Query Minimization In Practice

- No database system today performs minimization !!!
- Reason:
  - It’s hard (NP-complete)
  - Users don’t write non-minimal queries
- However, non-minimal queries arise when using views intensively

Query Minimization for Views

```sql
CREATE VIEW HappyBoaters
SELECT DISTINCT E1.name, E1.manager
FROM Employee E1, Employee E2
WHERE E1.manager = E2.name
and E1.bouter='YES'
and E2.bouter='YES'
```

This query is minimal

Query Minimization for Views

Now compute the Very-Happy-Boaters

```sql
SELECT DISTINCT H1.name
FROM HappyBoaters H1, HappyBoaters H2
WHERE H1.manager = H2.name
```

This query is also minimal

What happens in SQL when we run a query on a view?
**Query Minimization for Views**

**View Expansion**

```
SELECT DISTINCT E1.name
FROM Employee E1, Employee E2, Employee E3, Employee E4
WHERE E1.manager = E2.name and E1.boater = 'YES' and E2.boater = 'YES'
and E3.manager = E4.name and E3.boater = 'YES' and E4.boater = 'YES'
and E1.manager = E3.name
```

This query is no longer minimal!

```
E1 -- E3
|   |   |
|   |   |
|   |   |
|   |   |
|   |   |
E2 -- E4
```

E2 is redundant

**Monotone Queries**

**Definition** A query \( q \) is monotone if:

For every two databases \( D, D' \)
if \( D \subseteq D' \) then \( q(D) \subseteq q(D') \)

Which queries below are monotone?

\[
\phi = \exists x. R(x, x)
\]

\[
\phi = \exists x \exists y \exists z (R(x, y) \land R(y, z) \land R(z, u))
\]

\[
\phi = \exists x \exists y. R(x, y)
\]

**How To Impress Your Students Or Your Boss**

- Find all drinkers that like some beer that is not served by the bar “Black Cat”

```
SELECT L.drinker
FROM Likes L
WHERE L.beer not in (SELECT S.beer
                     FROM Serves S
                     WHERE S.bar = 'Black Cat')
```

- Can you write as a simple SELECT-FROM-WHERE (i.e. without a subquery) ?

**Expressive Power of FO**

- The following queries cannot be expressed in FO:

  - Transitive closure:
    \[
    \forall x, y, \exists x_1, \ldots, x_n, s.t.
    R(x, x_1) \land R(x_1, x_2) \land \ldots \land R(x_{n-1}, x_n) \land R(x_n, y)
    \]
  - Parity: the number of edges in \( R \) is even

**Datalog**

- Adds recursion, so we can compute transitive closure
- A datalog program (query) consists of several datalog rules:

\[
P_1(t_1) \leftarrow \text{body}_1
\]

\[
P_2(t_2) \leftarrow \text{body}_2
\]

\[
\ldots
\]

\[
P_n(t_n) \leftarrow \text{body}_n
\]
Datalog

Terminology:
• EDB = extensional database predicates
  – The database predicates
• IDB = intentional database predicates
  – The new predicates constructed by the program

Unfolding non-recursive rules

Graph: R(x,y)

P(x,y) :- R(x,u), R(u,y)
P(x,y) :- R(x,y)

Now the unfolding has a union:
A(x,y) :- R(x,y) \lor \exists u (R(x,u) \land R(u,y))

Recursion in Datalog

Graph: R(x,y)

Transitive closure:
P(x,y) :- R(x,u), R(u,y)
P(x,y) :- R(x,y)

Transitive closure:
P(x,y) :- R(x,u), R(u,y)
P(x,y) :- R(x,y)
Recursion in Datalog

Boolean trees:
Leaf0(x), Leaf1(x),
AND(x, y1, y2), OR(x, y1, y2),
Root(x)

• Write a program that computes:
  Answer() :- true if the root node is 1

Exercise

Boolean trees:
Leaf0(x), Leaf1(x),
AND(x, y1, y2), OR(x, y1, y2), Not(x,y),
Root(x)

• Hint: compute both One(x) and Zero(x)
  here you need to use Leaf0

Non-recursive Datalog

• Union of Conjunctive Queries = UCQ
  – Containment is decidable, and NP-complete

• Non-recursive Datalog
  – Is equivalent to UCQ
  – Hence containment is decidable here too
  – Is it still NP-complete?

Non-recursive Datalog

• A non-recursive database:
  \[ T_0(x,y) \rightarrow R(x,u), R(u,y) \]
  \[ T_1(x,y) \rightarrow T_0(x,u), T_1(u,y) \]
  \[ \ldots \]
  \[ T_n(x,y) \rightarrow T_{n-1}(x,u), T_{n-1}(u,y) \]
  Answer(x,y) :- T_n(x,y)

• Its unfolding as a CQ:
  \[ Answer(x,y) \rightarrow R(x,u_1), R(u_1,u_2), R(u_2,u_3), \ldots R(u_n,y) \]

• How big is this query?

Recursion in Datalog

One(x) :- Leaf1(x)
One(x) :- AND(x, y1, y2), One(y1), One(y2)
One(x) :- OR(x, y1, y2), One(y1)
One(x) :- OR(x, y1, y2), One(y2)
Answer() :- Root(x), One(x)
Query Complexity

• Given a query $\varphi$ in $\text{FO}$
• And given a model $D = (D, R_1^D, \ldots, R_k^D)$
• What is the complexity of computing the answer $\varphi(D)$

Vardi’s classification:

Data Complexity:
• Fix $\varphi$. Compute $\varphi(D)$ as a function of $|D|$

Query Complexity:
• Fix $D$. Compute $\varphi(D)$ as a function of $|\varphi|$

Combined Complexity:
• Compute $\varphi(D)$ as a function of $|D|$ and $|\varphi|$

Which is the most important in databases?

Example

$$\varphi(x) = \exists u. (R(u,x) \land \forall y. (\exists v. S(y,v) \Rightarrow \neg R(x,y)))$$

How do we proceed?

| R = | \begin{array}{c|c|c|c|c|c|c|c|c|c} \\
| 1 & 4 & 2 & 5 & 7 & 3 & 6 & 8 & 9 & 10 \\\n| 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\\n| \end{array} | \quad | S = \begin{array}{c|c|c|c|c|c|c|c|c|c} \\
| 1 & 4 & 2 & 5 & 7 & 3 & 6 & 8 & 9 & 10 \\\n| 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\\n| \end{array} |

General Evaluation Algorithm

for every subexpression $\varphi_i$ of $\varphi$, ($i = 1, \ldots, m$)
compute the answer to $\varphi_i$ as a table $T_i(x_1, \ldots, x_n)$
return $T_m$

Theorem. If $\varphi$ has $k$ variables then one can compute $\varphi(D)$ in time $O(|\varphi|^k|D|^k)$

Data Complexity = $O(|D|^k)$ = in PTIME
Query Complexity = $O(|\varphi|^k|D|^k)$ = in EXPTIME

General Evaluation Algorithm

Example:

$$\varphi(x) = \exists u. (R(u,x) \land \forall y. (\exists v. S(y,v) \Rightarrow \neg R(x,y)))$$

\begin{align*}
\varphi(u,x) &= R(u,x) \\
\varphi(y,v) &= S(y,v) \\
\varphi(x,y) &= \neg R(x,y) \\
\varphi_i(y) &= \exists v. \varphi(y,v) \\
\varphi_i(x,y) &= \varphi_i(y) \Rightarrow \varphi_i(x,y) \\
\varphi_i(x) &= \forall y. \varphi_i(x,y) \\
\varphi_i(u,x) &= \varphi_i(u,x) \land \varphi_i(x) \\
\varphi_i(x) &= \exists u. \varphi_i(u,x) = \varphi(x) \\
\end{align*}

Complexity

Theorem. If $\varphi$ has $k$ variables then one can compute $\varphi(D)$ in time $O(|\varphi|^k|D|^k)$

Remark. The number of variables matters!
Paying Attention to Variables

• Compute all chains of length m

  Chain_m(x, y) :- R(x, u_1), R(u_1, u_2), R(u_2, u_3), ... R(u_{m-1}, y)

• We used m+1 variables
• Can you rewrite it with fewer variables?

Counting Variables

• \( \text{FO}^k = \text{FO} \text{ restricted to variables } x_1, ..., x_k \)

  Write \( \text{Chain}_m \) in \( \text{FO}^3 \):

  Chain_m(x, y) :- \exists u. R(x, u) \land (\exists x. R(u, x) \land \exists u. R(x, u) ... \land (\exists a. R(u, y) ...))

Query Complexity

• Note: it suffices to investigate boolean queries only
  – If non-boolean, do this:

  for \( a_1 \) in D, ..., \( a_k \) in D
  if \( (a_1, ..., a_k) \) in \( \varphi(D) \) /* this is a boolean query */
  then output \( (a_1, ..., a_k) \)

Computational Complexity Classes

Recall computational complexity classes:
• \( \text{AC}^0 \)
• \( \text{LOGSPACE} \)
• \( \text{NL} \)
• \( \text{P} \)
• \( \text{NP} \)
• \( \text{PSPACE} \)
• \( \text{EXPSPACE} \)
• (Kalmar) Elementary Functions
• Turing Computable Functions

Data Complexity of Query Languages

Paper: On the Unusual Effectiveness of Logic in Computer Science

Important: the more complex a QL, the harder it is to optimize

Views

Employee(x), ManagedBy(x,y), Manager(y)

Views:

L(x,y) :- ManagedBy(x,u), ManagedBy(u,y)
E(x,y) :- ManagedBy(x,y), Employee(y)
Q(x,y) :- ManagedBy(x,u), ManagedBy(u,v), ManagedBy(v,w), ManagedBy(w,y), Employee(y)

How can we answer Q if we only have L and E?
Views

• Query rewriting using views (when possible):
  \[ Q(x,y) \Leftarrow L(x,a), L(u,y), E(v,y) \]

• Query answering:
  – Sometimes we cannot express it in CQ or FO, but we can still answer it

Applications:
• Using advanced indexes
• Using replicated data
• Data integration [Ullman’99]