4. Virtual Data Integration

- Virtual Data Integration

Problems:
- How to create mappings?
  - Discussed in previous part of the course
- How to compute query Q
  - This is the main focus of this part

4. Query Answering with Views

- How to compute query Q over global schema based on source schemas only?
  - What language is used to express mappings?
  - What language do we allow for Q?
  - What language(s) can we use to query local sources?
  - What language can we use to compute Q from query results returned by local sources?
  - How to deal with incompleteness?
4. Query Answering with Views

4.1 Query Answering with Views

Example: Solutions

Local Schema | Global Schema
---|---
Name | Name
Address | Address
City | City
Home-phone | Home-phone
Office-contact | Office-contact
Office-address | Office-address
Office-phone | Office-phone
Values of home-phone are not available in the source

Example: Solutions

Local Schema | Global Schema
---|---
Name | Name
Address | Address
City | City
Home-phone | Home-phone
Office-contact | Office-contact
Office-address | Office-address
Office-phone | Office-phone

Motivating Example (Part 1)

Movie(ID,title,year,genre)
Director(ID,director)
Actor(ID,actor)

Q(T,Y,D) : ¬Movie(T,Y,G), Y ≥ 1950, G = "comedy"
Director(D),Actor(ID)

V_1(T,Y,D) : ¬Movie(T,Y,G), Y ≥ 1940, G = "comedy"
Director(D),Actor(ID)

V_2(T,Y,D) : Movie(T,Y,G), Y ≥ 1950, G = "comedy"
Director(D),Actor(ID)

Containment is enough to show that V_2 can be used to answer Q.

Motivating Example (Part 2)

Q(T,Y,D) : Movie(T,Y,G), Y ≥ 1950, G = "comedy"
Director(D),Actor(ID)

V_1(T,Y,D) : Movie(T,Y,G), Y ≥ 1940, G = "comedy"
Director(D),Actor(ID)

V_2(T,Y,D) : ¬Director(D),Actor(ID)

Containment does not hold, but intuitively, V_2 and V_3 are useful for answering Q.

Q(T,Y,D) : V_2(T,Y,D), V_3(T,Y,D)

How do we express that intuition?

Answering queries using views!

Problem Definition

Input: Query Q
View definitions: V_1, ..., V_n

A rewriting: a query Q' that refers only to the views and interpreted predicates (comparisons)

An equivalent rewriting of Q using V_2,...,V_n:
A rewriting Q', such that Q' ⇔ Q
Naïve approach

- **Given Q and views**
  - Randomly combine views into a query Q'
  - Check equivalence of Q' and Q
  - If Q' is equivalent we are done
  - Else repeat

- **Why is this not good?**
  - There are infinitely many ways of combining views
    - E.g., V, V x V, V x V x V, ...
  - We are not using any information in the query

Motivating Example (Part 3)

Movie(ID,title,year,genre)
Director(ID,director)
Actor(ID,actor)

\[ Q(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[ \text{Director}(I,D), \text{Actor}(I,D) \]

\[ V_1(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1960, G = \text{"comedy"} \]
\[ V_2(I,D) : \neg \text{Director}(I,D), \text{Actor}(I,D) \]

\[ Q'(T,Y,D) : V_1(I,T,Y), V_2(I,D) \]

Maximally-Contained Rewritings

**Input:** Query Q

**Rewriting query language** \( L \)

**View definitions:** \( V_1, \ldots, V_n \)

\( Q' \) is a maximally-contained rewriting of Q given \( V_1, \ldots, V_n \) and \( L \) if:

1. \( Q' \in L \),
2. \( Q' \subseteq Q \), and
3. there is no \( Q'' \) in \( L \) such that \( Q'' \subseteq Q \) and \( Q \subset Q'' \)

Why again?

Exercise: which of these views can be used to answer \( Q \)?

\[ Q(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[ \text{Director}(I,D), \text{Actor}(I,D) \]
\[ V_1(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[ V_2(I,D) : \neg \text{Director}(I,D), \text{Actor}(I,D) \]
\[ V_3(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]
\[ V_4(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, \]
\[ G = \text{"comedy"}, \text{Award}(I,W) \]
\[ V_5(I,T) : \neg \text{Movie}(I,T,Y,G), Y \geq 1940, G = \text{"comedy"} \]
Algorithms for answering queries using views

- **Step 1**: we’ll bound the space of possible query rewritings we need to consider (no comparisons)
- **Step 2**: we’ll find efficient methods for searching the space of rewritings
  - Bucket Algorithm, MiniCon Algorithm
- **Step 2b**: we consider “logical approaches” to the problem:
  - The Inverse-Rules Algorithm

Bounding the Rewriting Length

**Theorem**: if there is an equivalent rewriting, there is one with at most $n$ subgoals.

Query: $Q(X): p_1(X_1), \ldots, p_n(X_n)$

Rewriting: $Q'(X) : V(X_1), \ldots, V(X_m)$

Expansion: $Q''(X) : g_1(X), \ldots, g_k(X), \ldots, g_j(X)$

Proof: Only $n$ subgoals in $Q$ can contribute to the image of the containment mapping $\phi$

Complexity Result
[LMSS, 1995]

- Applies to queries with no interpreted predicates.
- Finding an equivalent rewriting of a query using views is NP-complete
  - Need only consider rewritings of query length or less.
- Maximally-contained rewriting:
  - Union of all conjunctive rewritings of length $n$ or less.

The Bucket Algorithm

**Key idea**:
- Create a bucket for each subgoal $g$ in the query.
- The bucket contains views that contribute to $g$.
- Create rewritings from the Cartesian product of the buckets (select one view for each goal)

- **Step 1**: assign views with renamed vars to buckets
- **Step 2**: create rewritings, refine them, until equivalent/all contained rewriting(s) are found

The Bucket Algorithm

**Step 1 Intuition**
- A view can only be used to provide information about a goal $R(X)$ if it has a goal $R(Y)$
  - $Q(X) : R(X, Y)$
  - $V(X) : S(X, Y)$
- If the query goal contains variables that are in the head of the query, then the view is only useful if it gives access to these values (they are in the head)
  - $Q(X) : R(X, Y)$
  - $V(X) : S(X, Y), R(Y, Z)$
Bucket Algorithm in Action

\( Q(ID, Dir) \rightarrow \text{Movie}(ID, title, year, genre), \text{Revenues}(ID, amount), \text{Director}(ID, dir) \)

- View atoms that can contribute to \( \text{Movie} \):
  - \( V_1(ID, year) \)
  - \( V_2(ID, A') \)
  - \( V_3(ID, D', year) \)

Next Candidate Rewriting

\[
\begin{align*}
V_1(ID, year) & \rightarrow V_1(ID, Y') & V_4(ID, Dir, Y') \\
V_2(ID, A') & \rightarrow V_3(ID, amount) \\
V_3(ID, D', year) & \rightarrow V_6(ID, amount), V_6(ID, dir, y'), \text{amount} \geq 100M
\end{align*}
\]

\( q_1'(ID, dir) \rightarrow V_1(ID, A'), V_2(ID, amount), V_3(ID, dir, y') \)

\( q_2'(ID, dir) \rightarrow V_1(ID, amount), V_6(ID, dir, y'), \text{amount} \geq 100M \)

The Bucket Algorithm

Step 2:
- For each combination of one element of each bucket:
  - Create query \( Q' \) with \( Q \)'s head and list all these view atoms in the body
  - If \( Q' \) equivalent to \( Q \) (or contained in \( Q \))
    - Done (equivalent)
    - Add to union of CQs for contained case
  - If not try to add comparisons

The Bucket Algorithm: Summary

- Cuts down the number of rewriting that need to be considered, especially if views apply many interpreted predicates.
- The search space can still be large because the algorithm does not consider the interactions between different subgoals.
  - See next example.

The MiniCon Algorithm

\[
Q(title, year, dir) \rightarrow \text{Movie}(ID, title, year, genre), \text{Director}(ID, dir), \text{Actor}(ID, dir)
\]

\( V_5(D, A) \rightarrow \text{Director}(I, D), \text{Actor}(I, A) \)

**Intuition**: The variable \( I \) is not in the head of \( V_5 \); hence \( V_5 \) cannot be used in a rewriting. **MiniCon** discards this option early on, while the Bucket algorithm does not notice the interaction.
MinCon Algorithm Steps

1) Create MiniCon descriptions (MCDs):
   - Homomorphism on view heads
   - Each MCD covers a set of subgoals in the query with a set of subgoals in a view

2) Combination step:
   - Any set of MCDs that covers the query subgoals (without overlap) is a rewriting
   - No need for an additional containment check!

MiniCon Descriptions (MCDs)
An atomic fragment of the ultimate containment mapping

Q(title,act,dir) :- Movie(ID,title,year,genre),
                  Director(ID,dir),Actor(ID,act)

V(I,D,A) :- Director(I,D),Actor(I,A)

MCD:
      ID -> I
      dir -> D
      act -> A

covered subgoals of Q: {2,3}

MCDs: Detail 1

Q(title,year,dir) :- Movie(ID,title,year,genre),
                  Director(ID,dir),Actor(ID,dir)

V(I,D,A) :- Director(I,D),Actor(I,A)

Need to specialize the view first:

V'(I,D,D) :- Director(I,D),Actor(I,D)

MCD:
      ID -> I
      dir -> D

covered subgoals of Q: {2,3}

MCDs: Detail 2

Q(title,year,dir) :- Movie(ID,title,year,genre),
                  Director(ID,dir),Actor(ID,dir)

V(I,D,D) :- Director(I,D),Actor(I,D),
           Movie(I,T,Y,G)

Note: the third subgoal of the view is not included in the MCD.

MCD:
      ID -> I
      dir -> D

covered subgoals of Q still: {2,3}

Inverse-Rules Algorithm

• A "logical" approach to AQUV
• Produces maximally-contained rewriting in polynomial time
  - To check whether the rewriting is equivalent to the query, you still need a containment check.
• Conceptually simple and elegant
  - Depending on your comfort with Skolem functions...

Inverse Rules by Example

Given the following view:

V_v(I,T,Y,G) :- Movie(I,T,Y,G),Director(I,D),Actor(I,D)

And the following tuple in V_v:

V_v(79,Manhattan,1979,Comedy)

Then we can infer the tuple:

Movie(79,Manhattan,1979,Comedy)

Hence, the following ‘rule’ is sound:

IN_v : Movie(I,T,Y,G) :- V_v(I,T,Y,G)
Skolem Functions

\[ V_7(I,T,Y,G) : \neg \text{Movie}(I,T,Y,G), \text{Director}(I,D), \text{Actor}(I,D) \]

Now suppose we have the tuple
\[ V_7(79, \text{Manhattan}, 1979, \text{Comedy}) \]

Then we can infer that there exists some director. Hence, the following rules hold (note that they both use the same Skolem function):

\[ \text{IN}_2: \text{Director}(I, f_1(I,T,Y,G)) \rightarrow V_7(I,T,Y,G) \]
\[ \text{IN}_3: \text{Actor}(I, f_1(I,T,Y,G)) \rightarrow V_7(I,T,Y,G) \]

Inverse Rules in General

Rewriting = Inverse Rules + Query

Given \( Q_2 \), the rewriting would include:
\[ \text{IN}_1, \text{IN}_2, \text{IN}_3, Q_2. \]

Given input: \( V_7(79, \text{Manhattan}, 1979, \text{Comedy}) \)
Inverse rules produce:

\[ \text{Movie}(79, \text{Manhattan}, 1979, \text{Comedy}) \]
\[ \text{Director}(79, f_1(79, \text{Manhattan}, 1979, \text{Comedy})) \]
\[ \text{Actor}(79, f_1(79, \text{Manhattan}, 1979, \text{Comedy})) \]

(Exercise: apply \( Q_2 \) to the last tuple.)

Comparing Algorithms

- Bucket algorithm:
  - Good if there are many interpreted predicates
  - Requires containment check. Cartesian product can be big
- MiniCon:
  - Good at detecting interactions between subgoals

Algorithm Comparison (Continued)

- Inverse-rules algorithm:
  - Conceptually clean
  - Can be used in other contexts (see later)
  - But may produce inefficient rewritings because it “undoes” the joins in the views (see next slide)
- Experiments show MiniCon is most efficient.
- Even faster:
  Konstantinidis, G. and Ambite, J.L., Scalable query rewriting: a graph-based approach. SIGMOD ’11

Inverse Rules Inefficiency

Example

\[ Q(X,Y) : \neg e_1(X,Z) e_2(Z,Y) \]
\[ V(A,B) : \neg e_1(A,C) e_2(C,B) \]

Inverse rules:
\[ e_1(A, f(A,B)) : \neg V(A,B) \]
\[ e_2(f(A,B), B) : \neg V(A,B) \]

Now we need to re-compute the join…

View-Based Query Answering

- Maximally-contained rewritings are parameterized by query language.
- More general question:
  - Given a set of view definitions, view instances and a query, what are all the answers we can find?
- We introduce certain answers as a mechanism for providing a formal answer.
View Instances = Possible DB’s

Consider the two views:

\[ V_8(\text{dir}) : \neg \text{Movie}(\text{ID}, \text{dir}, \text{actor}) \]
\[ V_9(\text{actor}) : \neg \text{Movie}(\text{ID}, \text{dir}, \text{actor}) \]

And suppose the extensions of the views are:

\[ V_8 : \{\text{Allen, Copolla}\} \]
\[ V_9 : \{\text{Keaton, Pacino}\} \]

Possible Databases

There are multiple databases that satisfy the above view definitions: (we ignore the first argument of Movie below)

DB1. {([Allen, Keaton], [Coppola, Pacino])}
DB2. {([Allen, Pacino], [Coppola, Keaton])}

If we ask whether Allen directed a movie in which Keaton acted, we can’t be sure.

Certain answers are those true in all databases that are consistent with the views and their extensions.

Certain Answers: Formal Definition

[Open-world Assumption]

- Given:
  - Views: \( V_1, \ldots, V_n \)
  - View extensions \( v_1, \ldots, v_n \)
  - A query \( Q \)
- A tuple \( t \) is a certain answer to \( Q \) under the open-world assumption if \( t \in Q(D) \) for all databases \( D \) such that:
  - \( V_i(D) \subseteq v_i \) for all \( i \).

[Closed-world Assumption]

- Given:
  - Views: \( V_1, \ldots, V_n \)
  - View extensions \( v_1, \ldots, v_n \)
  - A query \( Q \)
- A tuple \( t \) is a certain answer to \( Q \) under the open-world assumption if \( t \in Q(D) \) for all databases \( D \) such that:
  - \( V_i(D) = v_i \) for all \( i \).

Certain Answers: Example

\[ V_8(\text{dir}) : \neg \text{Director}(\text{ID}, \text{dir}) \]
\[ V_9(\text{actor}) : \neg \text{Actor}(\text{ID}, \text{actor}) \]
\[ Q(\text{dir}, \text{actor}) : \neg \text{Director}(\text{ID}, \text{dir}), \text{Actor}(\text{ID}, \text{actor}) \]

Under closed-world assumption:
- single DB possible \( \Rightarrow \) ([Allen, Keaton])

Under open-world assumption:
- no certain answers.

The Good News

- The MiniCon and Inverse-rules algorithms produce all certain answers
  - Assuming no interpreted predicates in the query (ok to have them in the views)
  - Under open-world assumption
  - Corollary: they produce a maximally-contained rewriting
In Other News...

- Under closed-world assumption finding all certain answers is co-NP hard!

**Proof:** encode a graph - $G = (V,E)$

$v_1(X): \neg \text{color}(X,Y)$ \hspace{1cm} $H(V_1) = V$
$v_2(Y): \neg \text{color}(X,Y)$ \hspace{1cm} $H(V_2) = \{\text{red, green, blue}\}$
$v_3(X,Y): \neg \text{edge}(X,Y)$ \hspace{1cm} $H(V_3) = E$

$q() : \neg \text{edge}(X,Y), \text{color}(X,Z), \text{color}(Y,Z)$

$q$ has a certain tuple iff $G$ is not 3-colorable

Interpreted Predicates

- In the views: no problem (all results hold)
- In the query $Q$:
  - If the query contains interpreted predicates, finding all certain answers is co-NP-hard even under open-world assumption
  - Proof: reduction to CNF.

Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance