Desiderada from source descriptions

Distinguish between sources with closely related data: so we can prune access to irrelevant sources.

Enable easy addition of new information sources: because sources are dynamically being added and removed.

Be able to find sources relevant to a query: reformulate queries such that we obtain guarantees on which sources we access.

Query Reformulation Problem

Problem: reformulate the user query referring to the mediated schema onto the local schemas.

Given

- a query Q in terms of the mediated-schema relations,
- descriptions of the data sources,

Find a query Q' that uses only the data source relations, such that

- $Q' \models Q$ (i.e., answers are correct) and
- Q' provides all the possible answers to Q using the sources.

Approches to Specification of Source Descriptions

Mediated schema relations defined as views over the source relations: TSIMMIS (Stanford), HERMES (Maryland).

Source relations defined as views over mediated-schema relations: Information Manifold (AT&T), Occam, Razor, Tukwila (UW), InfoMaster (Stanford).

Sources described as concepts in a description logic: SIMS (ISI/USC), Catarci and Lenzerini (Rome).

The Global As View Approach (GAV)

The mediated-schema relations are described in terms of the source relations.

Movies and their years can be obtained from either DB_1 or DB_2:

$$MovieYear(title, year) :- DB_1(title, director, year)$$
$$MovieYear(title, year) :- DB_2(title, director, year)$$

Movie reviews can be obtaind by joining DB_1 and DB_3:

$$\begin{aligned} MovieRev(title, director, review) :- \; & DB_1(title, director, year)\,\& \\ & DB_3(title, review). \end{aligned}$$

Query Reformulation in GAV

Query reformulation is done by rule unfolding.

Query: Find reviews for 1997 movies:

$$q(title, review) :- MovieYear(title, 1997) \,\&\\ MovieRev(title, director, review).$$

Reformulated query on the sources:

$$q(title, review) :- DB_1(title, director, year) \,\&\, DB_3(title, review)$$
$$q(title, review) :- DB_1(title, director, year) \,\&\\ DB_2(title, director, year) \,\&\, DB_3(title, review)$$

A containment check shows that the second rule is redundant.

The Local As View Approach (LAV)

Every data source is described as a query expression over mediated-schema relations:

$$
\begin{aligned}
S_1 : V_1(title, year, director) \subseteq\ & Movie(title, year, director, genre) \& \\
& American(director) \& \\
& year \geq 1960 \& genre = Comedy.
\end{aligned}
$$

$$
\begin{aligned}
S_2 : V_2(title, review) \subseteq\ & Movie(title, year, director, genre) \\
& year \geq 1990 \& Review(title, review).
\end{aligned}
$$

Query Reformulation

Find reviews for comedies produced after 1950:

$$q(title, review) :- Movie(title, year, director, Comedy)\ \&\ year \geq 1950\ \&\ Review(title, review).$$

$$V_1(title, year, director) \subseteq Movie(title, year, director, genre)\ \&\ American(director)\ \&\ year \geq 1960\ \&\ genre = Comedy.$$

$$V_2(title, review) \subseteq Movie(title, year, director, genre)\ year \geq 1990\ \&\ Review(title, review).$$

The reformulated query on the sources:

$$q'(title, review) :- V_1(title, year, director)\ \&\ V_2(title, review).$$

Reformulation Algorithm

How do we translate the query on the mediated-schema to a query on the sources?

Translate the query:

$$q(title, review) :- Movie(title, year, director, Comedy)\ \&\ year \geq 1950\ \&\ Review(title, review).$$

into a query on the sources:

$$q'(title, review) :- V_1(title, year, director)\ \&\ V_2(title, review).$$

Conceptually: we have a set of precomputed queries and we want to use them to answer a new query.

Comparison of the Approaches

See [Ullman, ICDT-97] for a detailed comparison.

Local as View approach:

Easier to add sources: specify the query expression.

Easier to specify constraints on contents of the sources: they're part of the query expression describing them.

Global as view:

Query reformulation is straightforward.

GLAV: the best of both worlds (Friedman & Millstein).

Local Completeness Information

Incomplete sources require that we look at all relevant sources.

Often, sources are complete, or locally complete:

`Movie(title, director, year)`
is complete for year $\geq$ 1960.
`Book(title, publisher, author)`
is complete for `American(publisher)`.

By exploiting source completeness we can avoid useless accesses to sources.

Answer completeness problem: is the answer to a query complete given the local completeness of the sources?

Example # 1: Incomplete Query Answer

Relations:

`Movie(title, director, year)` (incomplete before 1960)
`Show(title, theater, city, hour)`

Query: find movies (and their directors) playing in Seattle:

(Q_1): **SELECT** m.TITLE, m.DIRECTOR
FROM Movie m, Show s
WHERE m.TITLE = s.TITLE **AND** city = Seattle

Additions to `Movie` may change the answer to Q_1.

Example # 2: Complete Query Answer

Relations:

`Movie(title, director, year)`

`Oscar(title, year)`

Query: directors whose movies won Oscars after 1965:

(Q_2): $\texttt{SELECT m}.\mathrm{DIRECTOR}$
$\texttt{FROM Movie m, Oscar o}$
$\texttt{WHERE m}.\mathrm{TITLE} = \texttt{o}.\mathrm{TITLE}\ \texttt{AND}$
$\texttt{m}.\mathrm{YEAR} = \texttt{o}.\mathrm{YEAR}\ \texttt{AND}$
$\texttt{o}.\mathrm{YEAR} \geq 1965.$

NO, completeness is not simply disjointness of the query with the incomplete parts of the database.

The Query Optimization Problem

(artificial break from the reformulation problem)

The goal of a query optimizer:

> Translate a declarative query into an equivalent imperative program of *minimal* cost.

The imperative program is a query execution plan (QEP): an operator tree in some algebra.

Basic notions in optimization:

- search space, search strategy, cost model.

Similarities of Data Integration with Optimization in DDBMS

A distributed database:

- Queries execution can be distributed over multiple sites.
- Communication costs are significant.

Consequences for Query Optimization

- The optimizer needs to decide on operation locality.
- Plans should exploit independent parallelism.
- Plans should include new operations for reducing communication overhead.
- Query caching can become a significant factor.

Differences from DDBMS

Capabilities of data sources: Data sources do not have the same role as a storage manager in a traditional DBMS:

- May provide only limited access patterns to the data.
- May have additional query processing capabilities.

Information about sources and network are missing:

- cost of answering queries is unknown,
- statistics about the data are harder to estimate.
- transfer rates are unpredictable.

In DDBMS the data is distributed over sites according to some precise rules.

Modeling Source Capabilities

Negative capabilities:

- A web-site may require certain inputs (in an HTML form) to answer a query
- Need to consider only valid query execution plans.

Positive capabilities:

- A source may be an ODBC compliant database system.
- Need to decide the placement of operations according to capabilities.

Problem: how do describe and exploit source capabilities.

Negative Capabilities

We model limited access limitations by binding pattern limitations:

$CitationDB^{bf}(X, Y) \subseteq Cites(X, Y)$
$CitingPapers^{f}(X) \subseteq Cites(X, Y)$
Query: $Q(X) : -Cites(X, a)$

Need to consider only valid plans:

$q(X) : -CitingPapers(Y) \,\&\, CitationDB(Y, a).$

Contained Rewritings for Views with Binding Patterns

In some cases, we're in real trouble: [Example from Kwok & Weld, 96]:

$UWDB(X) \subseteq UWPapers(X)$
$CitationDB^{bf}(X,Y) \subseteq Cites(X,Y)$
$AwardDB^{b}(X) \subseteq AwardPaper(X)$
Query: $Q(X) : -AwardPaper(X)$

There is no bound on the size of the reformulated query:
$Q'(X) : -UWDB(X) \,\&\, AwardDB(X)$
$Q'(X) : -UWDB(V) \,\&\, CitationDB(V, X_1) \ldots$
$CitationDB(X_n, X) \,\&\, AwardDB(X)$

Recursive Rewritings

Not to worry: invent recursive rewritings [Duschka & Levy, 97]

$$papers(X) :- UWDB(X)$$
$$papers(X) :- papers(Y) \,\&\, CitationDB(Y, X)$$
$$Q'(X) :- papers(X) \,\&\, AwardDB(X).$$

$$UWDB(X) \subseteq UWPapers(X)$$
$$CitationDB^{bf}(X, Y) \subseteq Cites(X, Y)$$
$$AwardDB^{b}(X) \subseteq AwardPaper(X)$$

Query: $Q(X) :- AwardPaper(X)$

Optimization in the Presence of Positive Capabilities

Characterization of positive capabilities:

- Schema dependent vs. schema independent:
  Source able to perform joins, selections or specifically $R \bowtie S$.

Describing and using positive capabilities:

- Positive-capabilities testing module (PCTM): how to decide that a plan is valid?
- Level of specification: declarative query vs. logical query execution plan.
- Interaction between optimizer and PCTM.

Query Processing in GARLIC

Extends traditional System-R and STAR framework to consider positive capabilities.

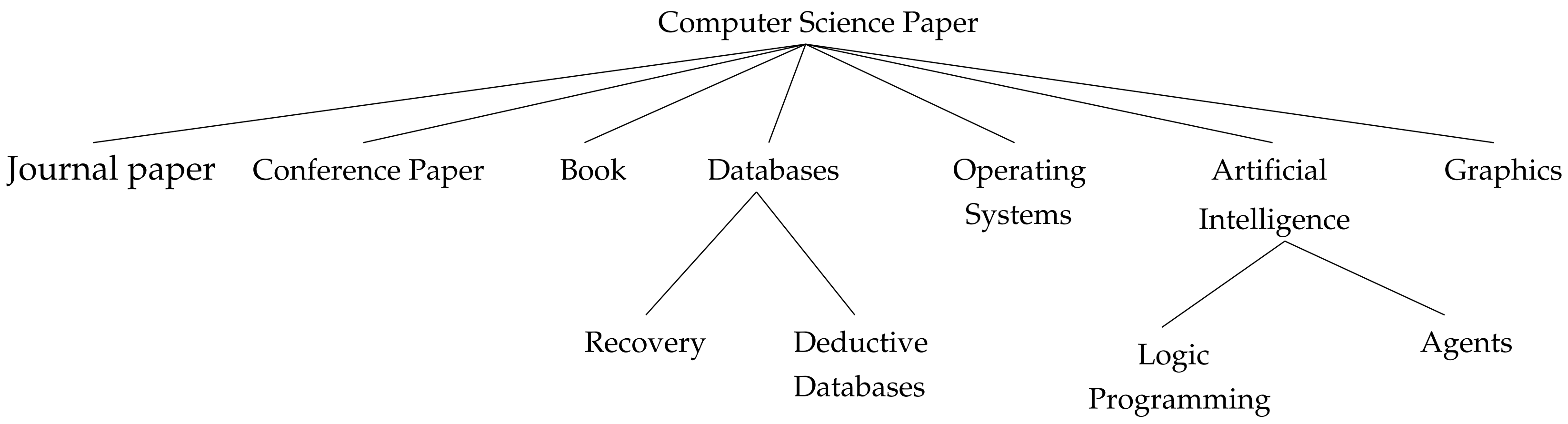
Specialized STARs detect opportunities to push processing to sources.

Given a query, the specialized STARs return a set of maximal(?) subqueries executable by the sources.

These subqueries are used as building blocks for plans in the bottom-up generation of query execution plans.

GARLIC emphasizes the interface between the optimizer and the PCTM, but leaves the implementation open.

Probabilistic Information in Data Integration



Attributes:

author
title
type
topics
year
journal
conference

Query: find papers journal papers on deductive databases.

Relevance reasoning: still leaves all computer science repositories relevant.

Need: information about overlaps between collections.

Fusion Queries

Specific data integration queries of the form:

SELECT $u_1.M$
FROM U u_1, ..., U u_n
WHERE $u_1.M = \ldots = u_n.M$ **and** $c_1, \ldots, c_l$

U: result of a union over a large number of sources.

Problem: impossible to search space of plans using traditional techniques (Yerneni et al, EDBT-98)

Exhaustive search for 3 different families of query execution plans:

filter plans, semi-join plans, and semi-join adaptive plans.

Efficiently Matching Objects Across Sources

Problem: how to decide that objects in two sources refer to the same object in the real world (recall: no uniform keys across data sources).

Cohen, SIGMOD-98: introduce IR techniques for similarity joins.

New (optimization) problem: plan contains only cartesian products.

Cohen's solution: novel query execution model:

- does not follow a query execution tree as usual,
- based on an A* search strategy for solution tuples.

Dealing with Unexpected Data Transfer Delays

Problem: even the best query execution plan can behave arbitrarily badly in the presence of unexpected data-transfer delays.

Query scrambling [Urhan et al, SIGMOD-98]: a set of run-time techniques to adapt to initial delays.

Query scrambling reacts in two ways:

1. Rescheduling the query execution plan: leave the plan unchanged, but evaluate different operators.
2. Operator synthesis: modify the tree by removing, adding or rearranging operators.

Adaptive Query Processing

Tukwila: a more general framework for query processing in data integration.

Because of lack of stats and unexpected network delays, we must interleave query optimization and execution.

- Execute plan fragments; reoptimize.
- System can decide to reoptimize even if the next fragment is planned out.
- Adaptive operators for data integration: collectors and double-pipelined joins.
- Rule-based mechanism for coordinating execution and optimization.

Final Conclusions

Data integration at a crossroads:

- Bulk of research moving to optimization and execution.
- XML!!??
- We need a more refined classification of the field.
- Test current systems on real applications.
- Benchmarks!
- Industry: service only, or also products?