Answering queries using views

Paper by Alon Halevy

Presentation by Jeffrey Niu
adapted from Rachel Pottinger

Discussion by Nalin Munshi

Background

- A view is a stored query
- e.g. in SQL:

Product(Name, Price, Category, Manufacturer)

Company(Cname, StockPrice, Country)

```
CREATE VIEW JapaneseProducts AS
SELECT Name, Price, Category, Manufacturer
FROM Product, Company
WHERE Product.Manufacturer=Company.Cname AND
Company.Country = 'Japan'
```

Background

- Datalog query example:

```
q(code) :- Airport(code, city),
           Feature(city, "Beach")
```

  Find all airport codes of cities that have beaches

Answering queries using views – basic definition

- Answer a query using views rather than using the underlying relations
- Query: q(code) :- Airport(code, city),
  Feature(city, POI)
- View:
  feature-code(code, POI) :- Airport(code, city),
  Feature(city, POI)
- Rewriting using views:
  q(code) :- feature-code(code, POI)

AQUV – two problems

- Query optimization
- Data integration
- Physical data independence

Query optimization goals

- Use views alongside base relations to answer query
- Optimize query speed
- Query rewrite with views needs to provide exact same answers
  - Sound and complete
- i.e. an *equivalent* rewriting

Query Optimization using Views: Discussion

- What are the advantages and disadvantages of using views for query optimization? Is it only for certain kinds of queries?

Closed world assumption

- Views are sound and complete – all valid answers to the view query are present, no extraneous answers
- Like "if and only if"
- feature-code(code, POI) :- Airport(code, city)
  Feature(city, "Beach")
  Retrieves *all* airport codes for cities w/ beaches
- Cannot tell whether this assumption holds from the view definition

Equivalent rewritings

- Equivalent example:
  Query: q(code) :- Airport(code, city), Feature(city, POI)
  View: feature-code(code, POI) :- Airport(code, city),
  Feature(city, POI)
  Equivalent rewriting: q(code) :- feature-code(code, POI)
- Non-equivalent example:
  Same query
  View: Beach-code(code) :- Airport(code, city),
  Feature(city, "Beach")
  Non-equivalent rewriting:
  q(code) :- Beach-code(code)

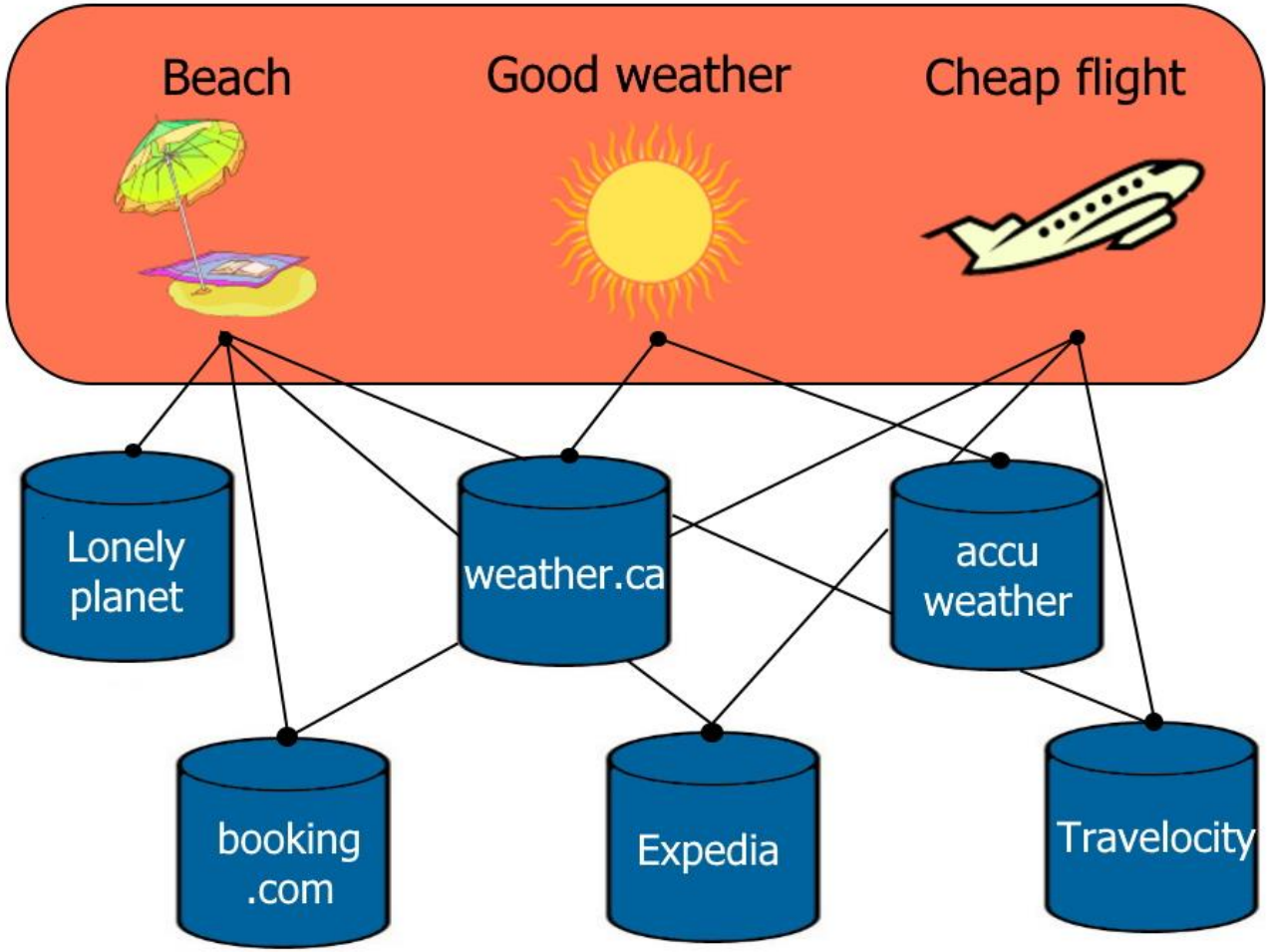
General algorithm

- Fold views into System-R style optimizer
- Views are another access path
  - Filter for views relevant to query
    - Table name in view from clause also present in query
    - Apply same join & selection predicates or apply logically weaker selection
    - Not project out any attributes needed in selection
- Optimal plan need not use the views
  - Consider indices available on views & base relations

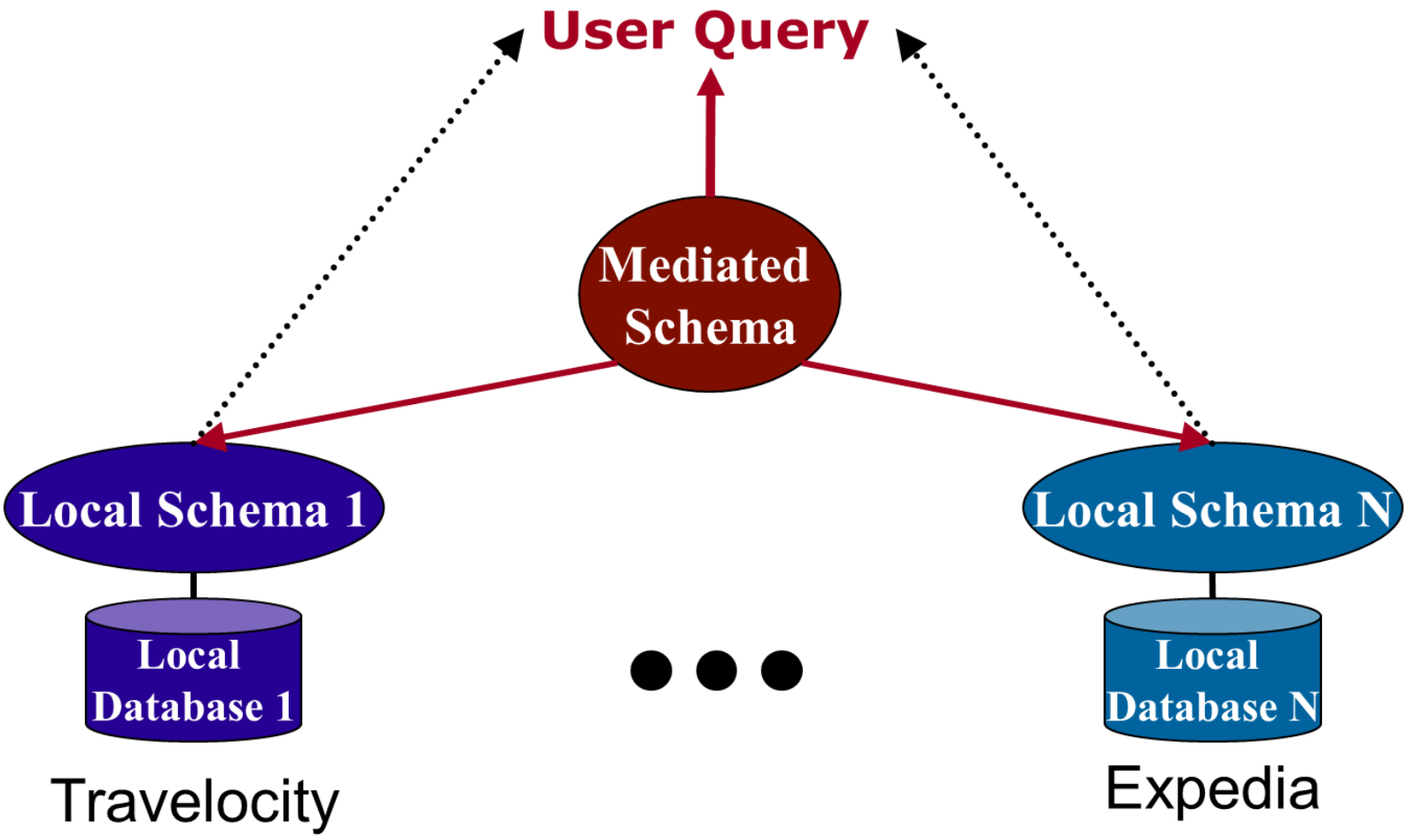
Data integration

- Goal: "to provide a uniform query interface to a multitude of autonomous data sources, which may reside within an enterprise or the World-Wide Web"

Data integration
Example: planning a beach vacation



Data integration architecture: Local-As-View (LAV)



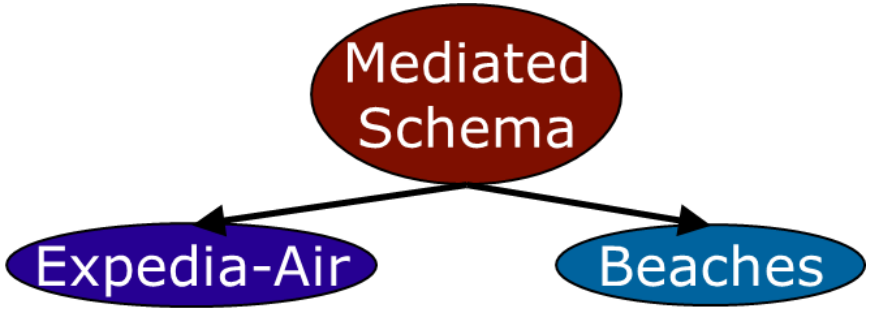
Local sources are views on mediated schema

Local-As-View (LAV)

- LAV: local source is *materialized view* over mediated schema

Mediated schema:

Airport(code, city)
Feature(city, attraction)



Local sources/views:

Expedia-Air(code, city) :- Airport(code, city)
Beaches(code) :- Airport(code, city),
Feature(city, "Beach")

- Adding new sources is easy
- Rewriting queries is NP-complete

Data integration assumptions

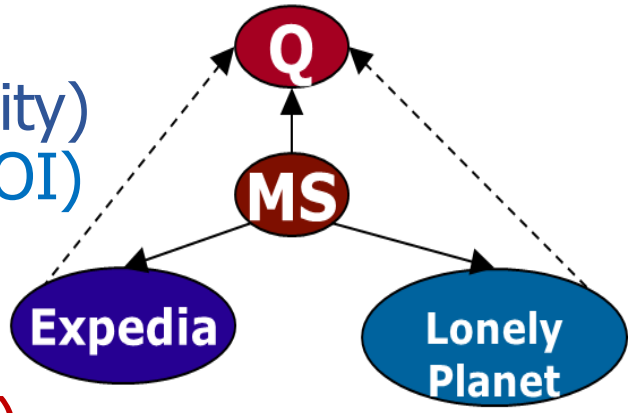
- Open world assumption:
  - Each source only has *some* of the tuples
  - Like "if → then"
  - LonelyPlanet(city, POI) :- Feature(city, POI)
    LonelyPlanet has *some* Features
  - This is an assumption – can't tell from view definition
- Can't access base relations
  - May not be able to find an equivalent rewriting

Open-world vs Closed-world assumption: Discussion

- Jianhao - Are there applications where it is more suitable to apply the open-world assumption, and the same for closed-world assumption?

Maximally contained rewritings

- Query:
  Dest(code) :- Airport(code, city), Feature(city, "Beach")
- Sources/Views:
  Expedia-Air(code, city) :- Airport(code, city)
  LonelyPlanet(city, POI) :- Feature(city, POI)
- Rewriting:
  Dest(code) :- Expedia-Air(code, city),
  LonelyPlanet(city, "Beach")



- Maximally contained rewriting: all answers to Query are a subset of those of Rewriting, and Rewriting contains all possible answers given local sources

Maximally contained rewritings

- New source Sun-Surf(city) :- Feature(city, "Beach") was added
- Sources/Views:
  Expedia-Air(code, city) :- Airport(code, city)
  LonelyPlanet(city, POI) :- Feature(city, POI)
  Sun-Surf(city) :- Feature(city, "Beach")
- Rewriting:
  Dest(code) :- Expedia-Air(code, city), LonelyPlanet(code, city)
  ∪
  Dest(code) :- Expedia-Air(code, city), Sun-Surf(city)
- This extends to taking the Cartesian product of all ways of covering view subgoals

Maximally contained rewritings: Discussion

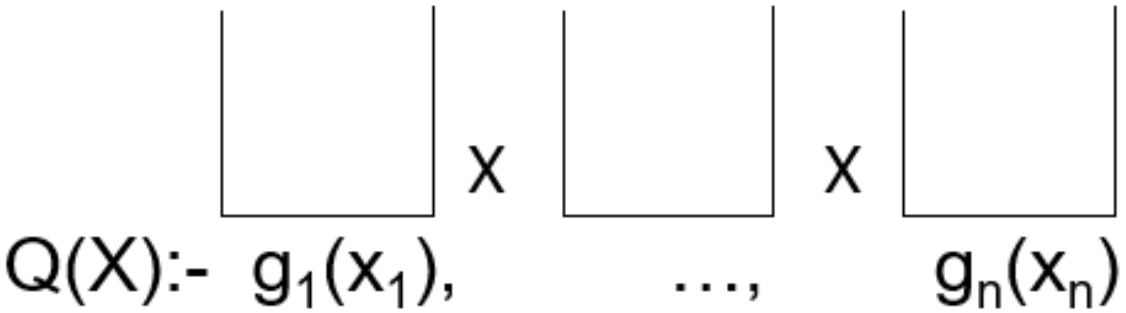
- What factors can influence the effectiveness and efficiency of maximally contained rewritings?
- What are some other use cases of maximally contained rewritings apart from data integration?

How to find maximally contained rewritings

- Bucket algorithm
- Minicon
- Inverse rules algorithm

Naïve solution: bucket algorithm

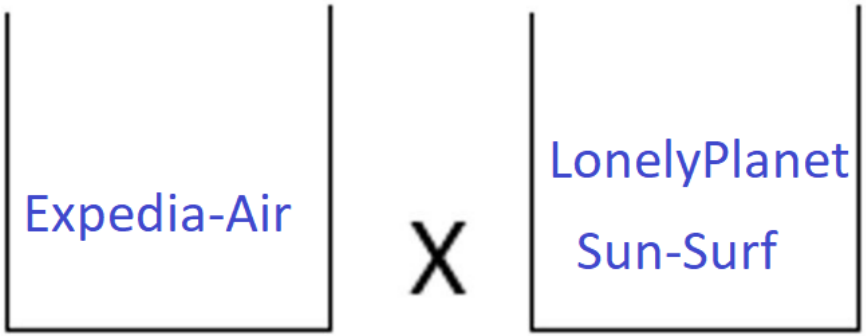
- Create a bucket for each query subgoal, place all relevant views into the bucket



- For each element in cross-product of the buckets, check for containment (check that answers contained in original query)
  - Containment check is Π_2^p - complete

Naïve solution: bucket algorithm

- Query:



Dest(code) :- Airport(code, city), Feature(city, "Beach")

- Sources/Views:
  Expedia-Air(code, city) :- Airport(code, city)
  LonelyPlanet(city, POI) :- Feature(city, POI)
  Sun-Surf(city) :- Feature(city, "Beach")

Subgoal interaction

- Bucket algorithm doesn't recognize interactions
- Query:
  Dest(code) :- Airport(code, city), Feature(city, "Beach")
- Sources/Views:
  Travelocity(code) :- Airport(code, city)
  Beaches(code) :- Airport(code, city), Feature(city, "Beach")
  Frommers(city, POI) :- Feature(city, POI)
- Bucket would check:
  Dest'(code) :- Travelocity(code), Frommers(city, "Beach")
  equivalent to:
  Dest'(code) :- Airport(code, __), Frommers(city, "Beach")
- Dest' not contained in Dest

MiniCon phase one

- Query:
  Dest(code) :- Airport(code, city), Feature(city, "Beach")
- Source/Views:
  Travelocity(code) :- Airport(code, city)
  Beaches(code) :- Airport(code, city), Feature(city, "Beach")
- Rewriting:
  Dest(code) :- Beaches(code)

Create MiniCon Descriptions (MCD): view subgoals linked by existential variables *must* be mapped together

MiniCon phase two

- Combine MCDs with non-overlapping subgoals
- Query:
  Dest(code) :- Airport(code, city), Feature(city, "Beach"),
  Flight("YVR", code, airline, number)
- Sources/Views:
  Travelocity(code) :- Airport(code, city)
  Beaches(code) :- Airport(code, city), Feature(city, "Beach")
  Expedia(orig, dest) :- Flight(orig, dest, airline, number)
- Rewriting:
  Dest(code) :- Beaches(code), Expedia("YVR", code)

MiniCon advantages

- Fewer combinations to perform Cartesian product
- No explicit containment check
  - Careful construction of MCDs and only combining MCDs covering disjoint sets of subgoals avoids check

Maximally contained Rewriting Algorithms: Discussion

- Rank the three algorithms – Bucket and MiniCon on the basis of the following parameters (1 being the best):

Algorithm	Compute	Memory	Parallelism
Bucket			
MiniCon			

- While solving an optimization problem what trade-offs should be kept in, for example, speed vs optimality, heuristic vs algorithm vs ML etc?