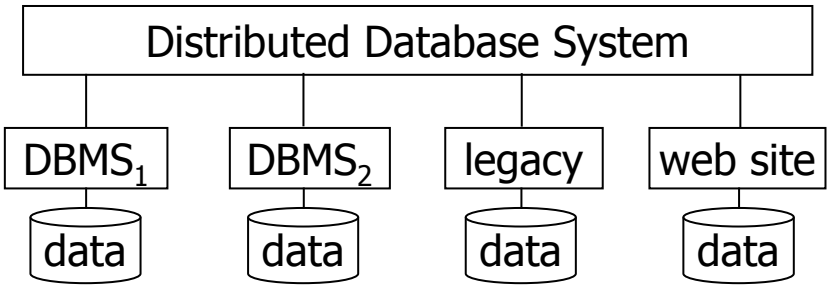
Mediator Systems & Their Capabilities

Adapted from slides by Hector Garcia-Molina and Sudarshan

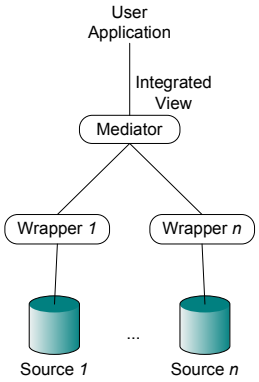
Heterogeneous Databases



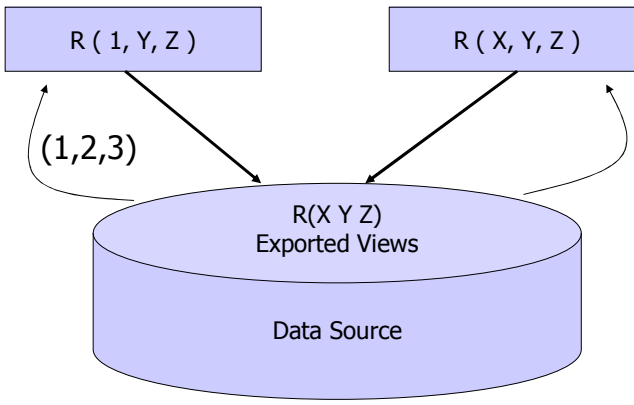
Mediation Architecture

Mediators

- Define integrated views based on the data provided by the sources
- Translate user queries on integrated views into source queries
- Perform postprocessing operations on the source query results

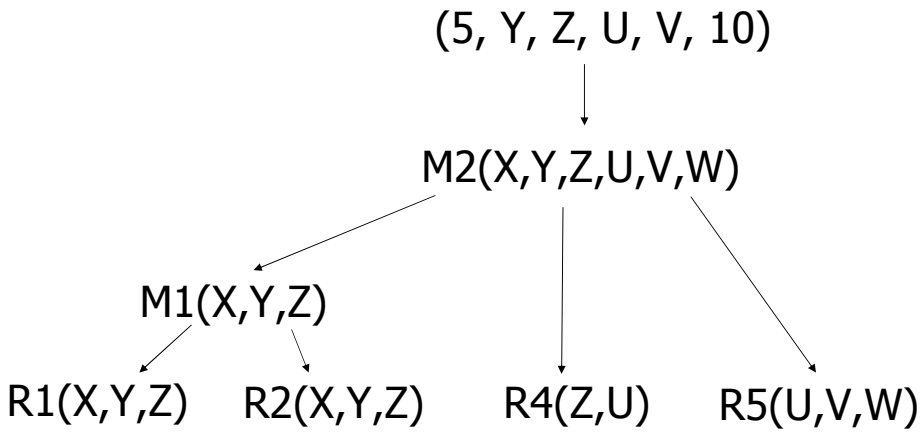


Traditional Unlimited Capabilities

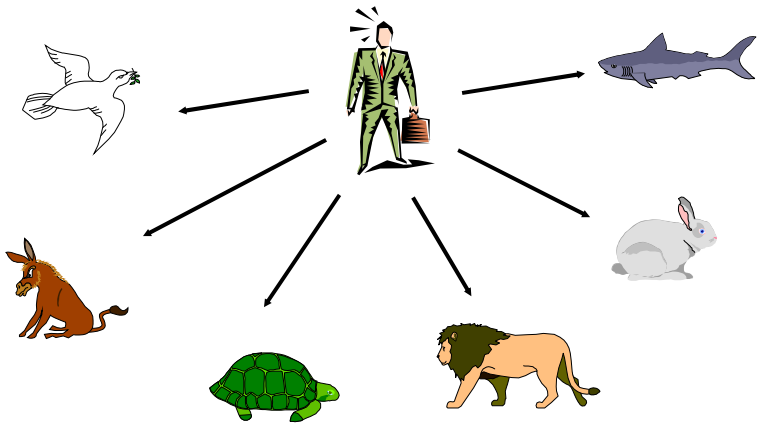


X	Y	Z
1	2	3
4	5	6
7	8	9

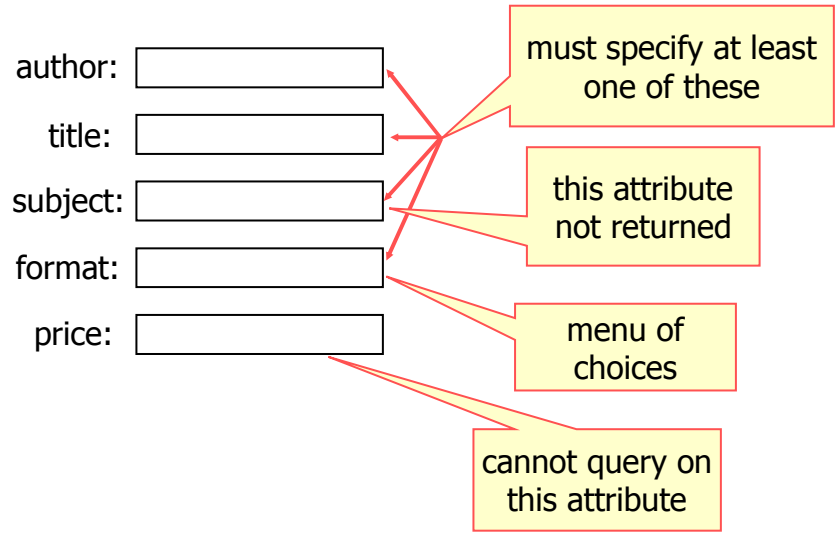
Traditional Unlimited Capabilities



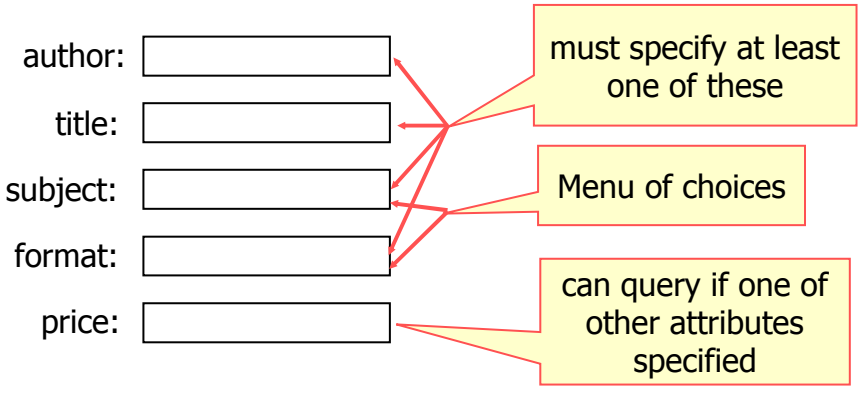
Limited Capabilities



Example: Amazon.com



Example: BarnesAndNoble.com



Why Limited Capabilities?

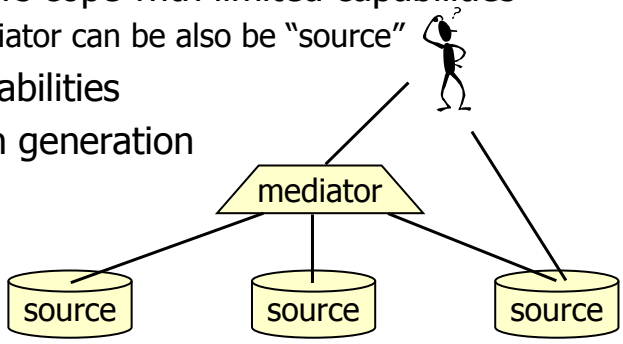
- Search forms
- Security
- Indexes
- Legacy

Capability vs. Content

- Capability description
  - Can only search for subject = "art," "history," "science"
- Content description
  - Source only contains subject = "art," "history," "science"

Issues

- Describing source capabilities
- Extending source capabilities
  - Beyond what source can do
- How mediators cope with limited capabilities
  - So that mediator can be also be "source"
- Mediator capabilities
- Dynamic plan generation



"Computing Capabilities of Mediators"
(Paper I)

Describing Query/Source Capabilities

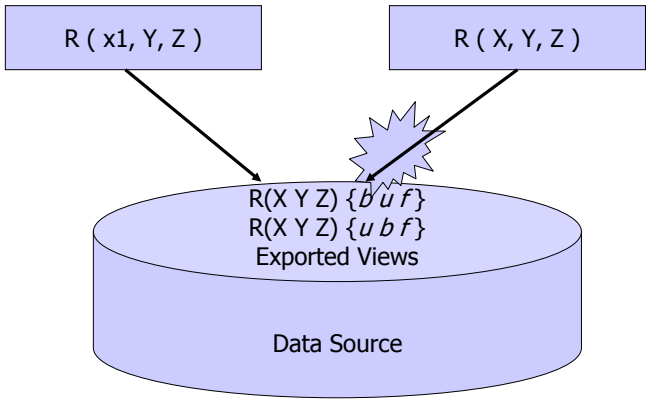
R(X, Y, … Z)

Adornments:
- **f**: may or may not specify
- **u**: cannot be specified
- **b**: must be specified
- **c[S]**: specified from list S
- **o[S]**: optional, chose from S

With output restriction
- **f'**
- **u'**
- **b'**
- **c'[S]**
- **o'[S]**

Capabilities of Data Sources

Some queries cannot be processed



Simple Mediators

- Simple Mediator Properties
  - No post processing
  - Performs joins blindly
- Union Views
  - Variable Adornments in Union View are computed from base view adornments
  - Mapping Function

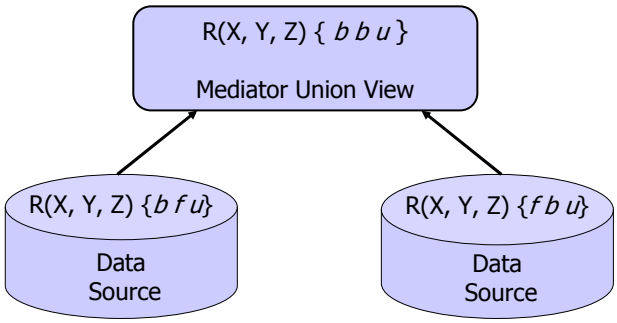
Simple Mediators

- Mapping Function:

	f	**o[s3]**	**b**	**c[s4]**	**u**
f	f	o[s3]	b	c[s4]	u
o[s1]	o[s1]	o[s1∩s3]	c[s1]	c[s1∩s4]	u
b	b	c[s3]	b	c[s4]	-
c[s2]	c[s2]	c[s2∩s3]	c[s2]	c[s2∩s4]	-
u	u	u	-	-	u

Simple Mediators

- Union of two views
  - Example

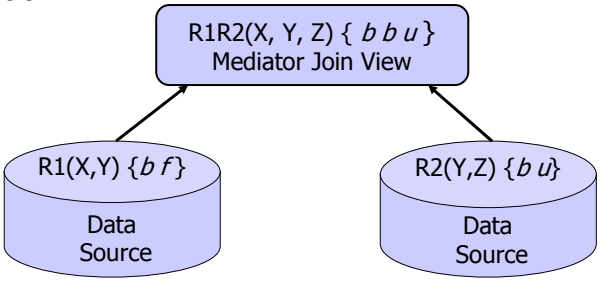


Simple Mediators

- Union of two base views with multiple templates
  - Cross product of two template sets
  - For each pair, apply mapping function
  - E.g., R1(X,Y,Z), has 2 templates *bff* & *ffb*; R2(X,Y,Z) has 1 template *fbf*; and R3(X,Y,Z) h as two template *ffc[s1]* & *c[s2]ff*. We'll end up with M(X,Y,Z) and 4 templates: *bbc[s1], fbc[s1], c[s2]bf* & *c[s2]bb*
- Union of multiple base views with multiple templates
  - Process two base views at a time
  - Repeat (n-1) times

Simple Mediators

- Join Views
  - Non-join attribute adornments are copied as they are in join view
  - Join attributes are computed using mapping function

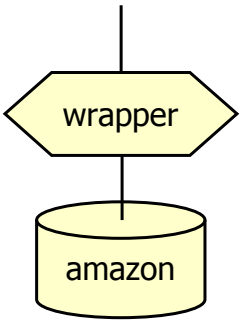


Simple Mediators

- Selection Views
  - Pass the query to underlying data source as it is if it fits the template
  - Filter the results according to selection predicates
- Projection Views
  - Hidden attributes are left unspecified
  - Works only if hidden attribute adornments are *f, u, o*

Extending Source Capabilities

Query: author="Freud" AND
price > 10

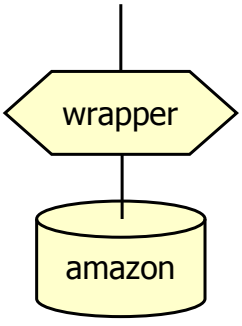


Source: R(author, price, ...)
Template: b, u, ...

Extending Source Capabilities

Query: author="Freud" AND
price > 10

Wrapper Filter: price > 10

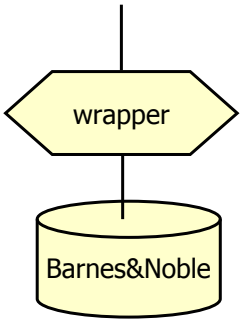


Source Query: author="Freud"

Source: R(author, price, ...)
Template: b, u, ...

Another Example

Query: (author = "Freud" OR author = "Jung")
AND price < 10

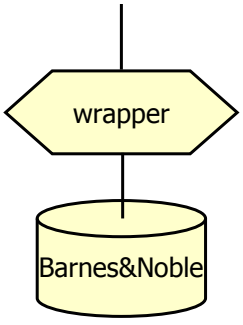


R(author, price, ...)
No disjunctive conditions;
Price can only be specified with author

Another Example

Query: (author = "Freud" OR author = "Jung")
AND price < 10

Union Operation



Q1: author = "Freud" AND price < 10
Q2: author = "Jung" AND price < 10

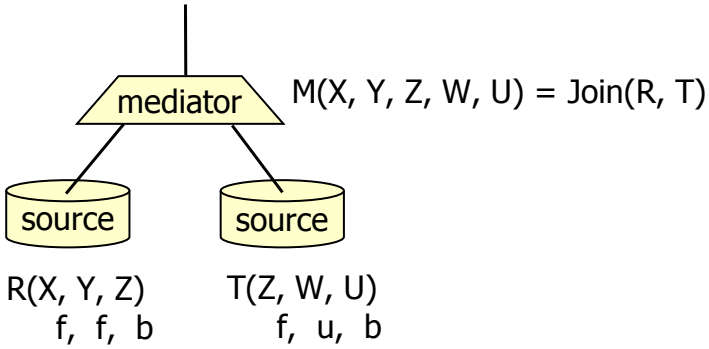
R(author, price, …)
No disjunctive conditions;
Price can only be specified with author

Extending Source Capabilities

- General scheme:
  - try many query rewritings
  - check if query fragments supported by source
  - check if wrapper can combine answer fragments

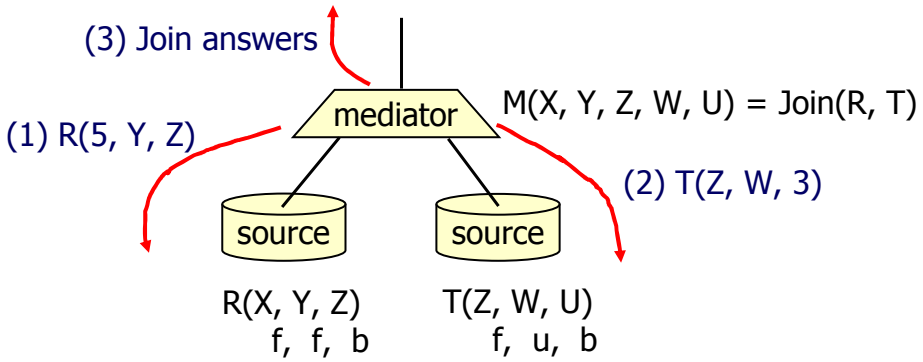
Mediator Processing

Query: **M(5, Y, Z, W, 3)**



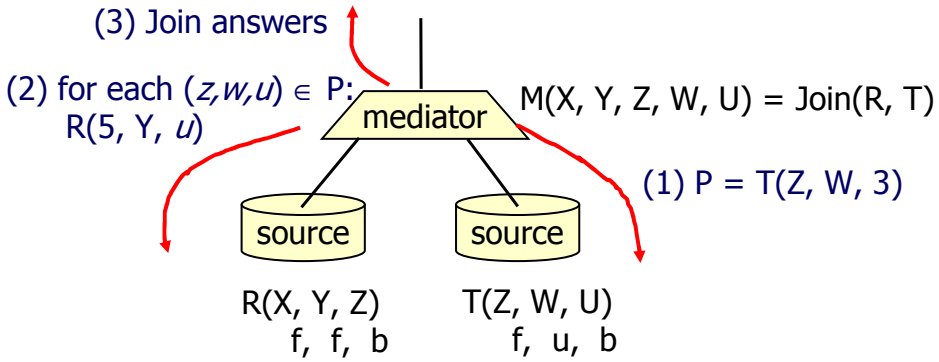
Plan 1

Query: **M(5, Y, Z, W, 3)**



Plan 2

Query: **M(5, Y, Z, W, 3)**

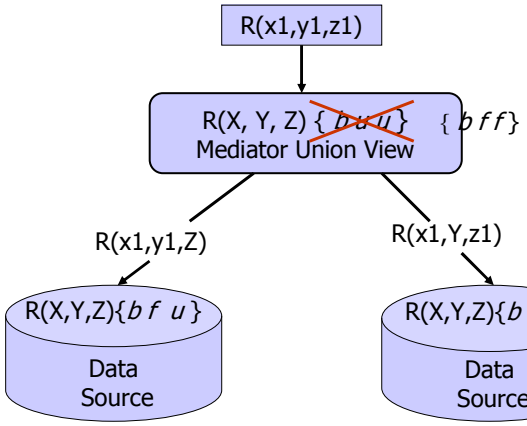


Advance Techniques in Mediators

- Union Views
  - Post processing
    - Filter the input query to fit source views
    - Post process the results to match user requirements
  - Union view adornments are computed using Mapping function

Advance Techniques in Mediators

- Union View Example



Union with postprocessing

Mapping function for union with postprocessing

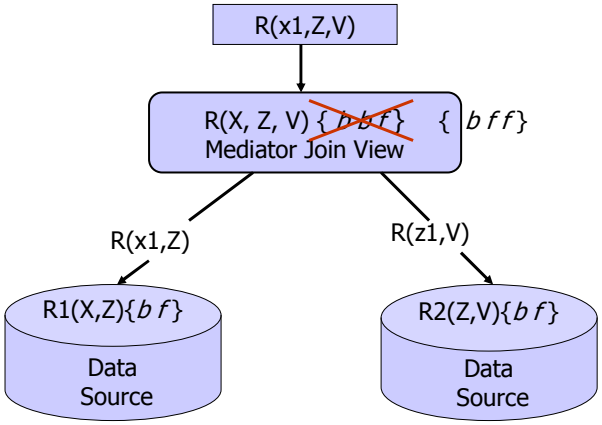
	f	**o[s3]**	**b**	**c[s4]**	**u**
f	f	f	b	c[s4]	f
o[s1]	f	f	b	c[s4]	f
b	b	b	b	c[s4]	b
c[s2]	c[s2]	c[s2]	c[s2]	c[s2∩s4]	c[s2]
u	f	f	b	c[s4]	f

Joins with Binding Passing

- Join Views
  - Non-join attribute adornments are copied as they are
  - Passing join attribute bindings from left to right view
  - Join attribute adornments are computed using new mapping function

Joins with Binding Passing: Example

- Join views: passing bindings



Join with Binding Passing + Postprocessing

- Aka "Dependent join"
- Mapping function (left relation -> rows)

Left \ Rt.	**f**	**o[s3]**	**b**	**c[s4]**	**u**
f	f	f	f	c[s4]	f
o[s1]	f	f	f	c[s4]	f
b	b	b	b	c[s4]	b
c[s2]	c[s2]	c[s2]	c[s2]	c[s2∩s4]	c[s2]
u	f	f	f	c[s4]	f

Multiple Joins/Bindings

- Join of 2 relations with multiple bindings for each relation
  - Cross product
    - one result template for each combination
- Ordered sequence of n relations
  - Compute pairwise from left
- Binding pattern depends on join order
  - n! possible join sequences with n relations
  - Compute pattern (pairwise) for each sequence, take union
    - Mediator will have to select appropriate join order depending on query bindings

Example

- R1(X,Y,Z): *fbf* & *bfb*
- R2(Z,U,V): *bfb* & *fbf*
- R3(V,W): *fb*
- Without binding, we have 4 tempates: *fbbfbb, fbfbfb, bfbfbb, bfbbfb*
- For sequence (R1,R2,R3), we have 4 templates: *fbffbb, fbfbfb, bfbfbb & bfbbfb*
- Total: 6 sequences and 24 templates, and end up with 3 templates: *fbfffb, bfbffb, bffbfb*
- Query M(X,y1,Z,U,v1,w1) is feasible if mediator passes bindings

Selections with postprocessing

Selection may or may not provide value for attribute

Base view adornment	Sel. Attribute Adornment	Non-sel attribute adornment
f	f	f
o[s1]	f	f
b	f or b	b
c[s1]	f or c[s1]	c[s1]
u	f	f

Dynamic Mediators

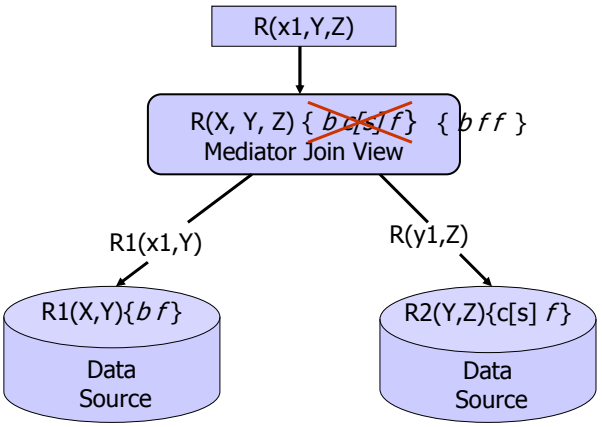
- What we have seen so far
  - A query is answerable if it satisfies a template
- But, some queries are answerable even if no templates can be satisfied
  - M(X,Y,Z,Y): R1(X,Y), R2(Y,Z,U)
  - R1: *bf* R2: *c[2]fb* (s: y1,y2,y3) M: *bc[2]fb*
  - Query M(x1,y1,Z,u1) is answerable
  - Query M(x1,Y,Z,u1) does not satisfy the template but answerable if answer of Y in R1(x1,Y) is subset of s
    - Costly if we need to run R1 since sometimes it may fail
  - Query M(x1,Y,z1,U) also does not satisfy the template but is not answerable irrespective of the set of Y values of R (since U is not specified)
- Can determine if query is answerable without running?
  - Conservative vs liberal templates

Dynamic Mediators

- Liberal templates ("upper bound")
  - Any query that does not satisfy the template cannot be answered
  - Answerability depends on state of data in data source
  - Can't be always answered
- Conservative templates ("lower bound")
  - Can always be answered
  - Computed using techniques discussed in earlier slides

Dynamic Mediators

- Liberal Template Example



Liberal Template for Join Views

Left \ Rt.	**f**	**o[s3]**	**b**	**c[s4]**	**u**
f	f	f	f	f	f
o[s1]	f	f	f	f	f
b	b	b	b	b	b
c[s2]	c[s2]	c[s2]	c[s2]	c[s2]	c[s2]
u	f	f	f	f	f

Dynamic Mediators

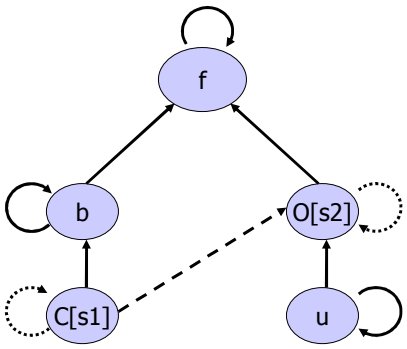
- Simple Mediators
- Mediators employing advanced techniques
  - Post Processing
  - Passing bindings
- Dynamic mediators
  - Liberal Templates

Concise Capability Description

- Mediator view space is exponential
- k templates per view, n sources
  - View space will be k^n
- Large number of templates are redundant
  - M1(*b f b*) and M2(*f f f*) :
    M2 is redundant

Concise Capability Description

- Adornment Graph



Concise Capability Description

- Subsumption test
  - Template T subsumes T1 iff
    Every attribute x in T1 is atleast as restrictive as T
- Subsumption test partitions the set of templates
- Select the template which subsumes all other templates in a partition

Concise Capability Description

- Suppose a view has the following templates:
  {bff, fbf, ffb, c[s1]fb, ubo[s2]}
- Based on Subsumption Test defined above, c[s1]fb is subsumed by bff and ubo[s2] is subsumed by fbf
- Resultant templates:
  {bff, fbf, ffb}

"Optimizing Large Join Queries In Mediation Systems"
(Paper II)

Problems

Heterogeneity in sources

Diverse and limited capabilities of sources in answering queries

Not all translations are feasible

An example

Mediator View:
`Movie(studio,title,year,stars) :- R(studio,title), S(title,year), T(title,stars)`

Source	Contents	Must Bind
S_1	`R(studio, title)`	either `studio` or `title`
S_2	`S(title, year)`	`title`
S_3	`T(title, stars)`	`title`

User's query `ans(title) :- Movie('Paramount', title, '1955', 'Gregory Peck')`

Mediator's logical plan `ans(title) :- R('Paramount', title), S(title, '1955'), T(title, 'Gregory Peck')`

subgoal; bound variable; variables

Many physical plans

Plan P_1: Send query `R('Paramount', title)` to S_1; send query `S(title, '1955')` to S_2; and send query `T(title, 'Gregory Peck')` to S_3. Join the results of the three source queries on the `title` attribute and return the `title` values to the user. **x**

Plan P_2: Get the titles of movies produced by Paramount from source S_1. For each returned title t, send a query to S_2 to get its `year` and check if it is '1955.' If so, send a query to S_3 to get the `stars` of movie t. If the set of `stars` contains 'Gregory Peck,' return t to the user. ✓

Challenges

- Given a logical plan and the description of the source capabilities, find feasible physical plans for the logical plan
- Determine the evaluation order for logical plan subgoals, so that attributes are appropriately bound
- Among all feasible physical plans, pick the most efficient one (based on some cost model)
- Consider conjunctive queries that are the most expensive in mediation systems → large join queries

Binding Relations and Source Queries

- n subgoals $C_1, C_2, \ldots, C_n$ $\xrightarrow{\text{define}}$ $n+1$ binding relations $I_0, I_1, \ldots, I_n$
- I_0 has as its schema the set of variables bound in the logical plan and it has a single tuple denoting the bindings specified in the logical plan
- The schema of I_1 is the union of the schema of I_0 and the schema of the source relation of C_1. Its instance is the join of I_0 and the source relation of C_1.
- The answer to the conjunctive query is defined by a projection operation on I_n

Example

- I0 = (Studio, Year, Star)
- I1 = (Studio, Year, Star, Title)
- I2 = (Studio, Year, Star, Title)
- T3 = (Studio, Year, Star, Title)
- Ans = Π_{Title}(I3)

- Subgoal R is a block query
- Subgoals S and T are parameterized queries

Binding Relations and Source Queries

In order to compute a binding relation I_j, I_{j-1} has to be joined with C_j

2 ways to perform this operation:

- Use I_0 to send a query to the source of C_j (by binding a subset of its attribute). Perform the join of the result of this source query with I_{j-1} at the mediator to obtain I_j

block query

- For j⩾2, use I_{j-1} to send a set of queries to the source relation of C_j (by binding a subset of its attribute). Union the results of these source queries. Perform the join of this union relation with I_{j-1} to obtain I_j.

parameterized query

The access templates for the corresponding source relations require bindings of variables that are not available in the logical plan → It may not be possible to answer some of the subgoals in the logical plan through block queries → In order to answer these subgoals, **parameterized queries** execute other subgoals and collect bindings for the required parameters of *Cj*

The Formal Cost Model

- The cost of a subgoal in the feasible plan is the number of source queries needed to answer this subgoal
- The cost of a feasible plan is the sum of the costs of all the subgoals in the plan

Even in this simple cost model, the problem of finding the optimal feasible plan with the minimum number of source queries is NP-hard.

But, it turns out that it is safe to restrict the space of plans to those based on left-deep-tree executions of the set of subgoals. So, **do not miss the optimal plan by not considering the executions of the logical plan based on bushy trees of subgoals**.

The CHAIN Algorithm

- Is based on a greedy strategy of building a single sequence of subgoals that is feasible and efficient
  - CHAIN starts by finding all subgoals that are answerable with the initial bindings in the logical plan.
  - It then picks the answerable subgoal with the least cost and computes the additional variables that are now bound due to the chosen subgoal.
  - It repeats the process of finding answerable subgoals, picking the cheapest among them and updating the set of bound variables, until no more subgoals are left or some subgoals are left but none of them is answerable.
  - If there are subgoals left over, CHAIN declares that there is no feasible plan; otherwise it outputs the plan it has constructed.

The CHAIN Algorithm (Example)

$R^{bff}(A,B,D)$	$S^{bf}(B,E)$	$T^{bf}(D,F)$
(1, 1, 1)	(1, 1)	(4, 1)
(1, 2, 2)	(2, 1)	(5, 1)
(1, 3, 3)	(3, 1)	(6, 1)
(1, 1, 4)	(4, 1)	(7, 1)

Consider one logical Plan:

$H \leftarrow R(1, B, D), S(B, E), T(D, F)$

The CHAIN Algorithm (Example)

S1: Only R(1, B, C) is answerable. Cost is 1.

S2: Now, we have R' which has 4 tuples. Now both B and D are bound. So, both S and T are answerable.

S3: Cost for S = 3. Cost for T = 4. So, subgoal S is evaluated.

Steps S2 and S3 are repeated. So, next round evaluates subgoal T.

We have a feasible plan: $R \rightarrow S \rightarrow T$. Cost = 1+3+4 = 8

CHAIN may miss the optimal plan

A better plan exists:

$R \rightarrow T \rightarrow S$

Cost = 1+4+1 = 6

The CHAIN Algorithm

- ✓ CHAIN runs in $O(n^2)$ time, where n is the number of subgoals
- ✓ CHAIN is guaranteed to find feasible plans when they exist
- ✓ CHAIN is guaranteed to find the optimal plan if the result of the user query is nonempty, and the number of subgoals in the logical plan is less than 3
- ✓ CHAIN is *n-competitive* → the plan generated by CHAIN can be at most n times as expensive as the optimal plan (n is the number of subgoals)

X CHAIN can miss the optimal plan if the logical plan has more than 2 subgoals

→ But, at least, there is a linear bound on its worst case performance

The PARTITION Algorithm

- Is based on a partitioning scheme

First phase

• Organizes the subgoals into clusters based on the capabilities of the sources.
• All the subgoals in the first cluster are answerable by block queries.
• All the subgoals in each subsequent cluster are answerable by parameterized queries that use attribute bindings from the subgoals of the earlier clusters.
• If in a round of the first phase there are subgoals yet to be picked and none of them is answerable, PARTITION declares that there is no feasible plan for the user query.

Second phase

PARTITION finds the best subplan for each cluster of subgoals and combines the subplans to arrive at the best overall plan for the user query. The subplan for each cluster is found by enumerating all the sequences of subgoals in the cluster and choosing the one with the least cost.

The PARTITION Algorithm

- Consider a logical plan with 4 subgoals
  $H \leftarrow R(A, B), S(A, D), T(B,E), U(D, B)$
- Let there be 4 source relations
  $R^{ff}(A, B)$ $S^{bf}(A, D)$ $T^{bf}(B, E)$ $U^{bf}(D, B)$
- Assume no attribute in the query has been bound.
- First Phase
  - Initially, only R is answerable; after R is answered, S and T will become feasible, so they are in the second cluster; finally, variables D and E are bound, so U is in cluster 3.
- Second Phase
  - Only cluster 2 has a choice: either $S \rightarrow T$ or $T \rightarrow S$
- PARTITION considers 2 plans

How bad can PARTITION be?

- Plan A: $R \rightarrow S \rightarrow T \rightarrow U$
  - Cost = 1+1 + 10000 + 1
- Plan B: $R \rightarrow T \rightarrow S \rightarrow U$
  - Cost = 10003
- One of these 2 plans are picked – same cost

$R^{ff}(A,B)$	$S^{bf}(A,D)$	$T^{bf}(B,E)$	$U^{bf}(D,B)$
(1,1)	(1,1)	(1,1)	(1,2)
(1,2)			
(1,3)			
…..			
(1,10000)			

- Note that once S has been answered, U becomes feasible.
- By answering U before T, all tuples whose B value is not 2 will be filtered.
- So, we have a better plan: $R \rightarrow S \rightarrow U \rightarrow T$. Cost = 1+1+1+1= 4!!!
- PARTITION misses this plan.
- Ratio of plan cost = 10003/4. Clearly, this ratio could have been *arbitrarily* large.

The PARTITION Algorithm

- ✓ PARTITION – like CHAIN – always finds feasible plans when they exist
- ✓ PARTITION is guaranteed to find optimal plans in more scenarios than CHAIN
- ✓ If there are fewer than 3 clusters generated, and the result of the query is nonempty, then PARTITION is guaranteed to find the optimal plan
- ✓ If the number of subgoals in the logical plan does not exceed 3, and the result of the query is not empty, then PARTITION will always find the optimal plan.

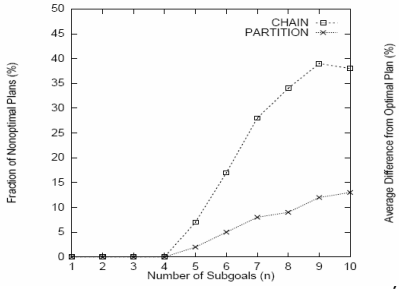
- X PARTITION is much less efficient than CHAIN and can take time exponential in the number of subgoals of the logic plan

- X When PARTITION misses the optimal plans, it can miss them by an unboundary margin

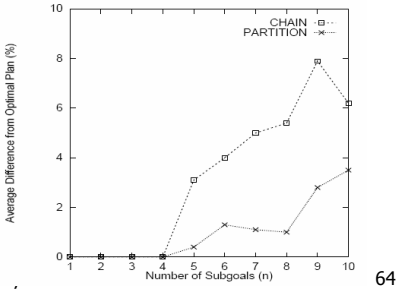
Performance Analysis

- Test set: 15 sources, 1000 queries with 1-10 subgoals
- PARTITION generates the optimal plan in more than 95% of the cases
- CHAIN generates the optimal plan in more than 75% of the cases

Surprising! Both algorithms have very good practical performance, even though they give very few theoretical guarantees!



Percentage of non-optimal plans



Average case miss margin

Summary

- The join order problem is that of ordering subgoals to find the best feasible sequence can
- Some solutions perform a rather exhaustive enumeration of plans, and hence do not scale well
- Other heuristics may generate efficient plans, but they do not have any quality guarantees
- CHAIN & PARTITION: A combination of efficient, scalable algorithms that generate provably good plans

Conclusion

- Not all sources are created equal!
- Need to
  - describe what sources can do
  - efficiently process queries with limited sources
  - describe what mediators can do
  - exploit content information
  - deal with unavailable sources