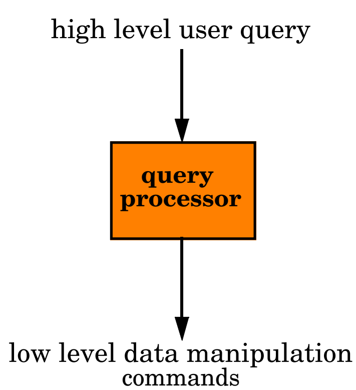
Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Semantic Data Control
- Distributed Query Processing
  - Query Processing Methodology
  - Distributed Query Optimization
- Distributed Transaction Management
- Distributed Database Operating Systems
- Open Systems and Interoperability
- Parallel Database Systems
- Distributed Object Management
- Concluding Remarks

Query Processing



Query Processing Components

- Query language that is used
  - SQL: “intergalactic dataspeak”
- Query execution methodology
  - The steps that one goes through in executing high-level (declarative) user queries.
- Query optimization
  - How do we determine the “best” execution plan?

Selecting Alternatives

```
SELECT   ENAME
FROM     E,G
WHERE    E.ENO = G.ENO
AND      DUR > 37
```

Strategy 1

$$\Pi_{ENAME}(\sigma_{DUR>37 \wedge E.ENO=G.ENO}(E \times G))$$

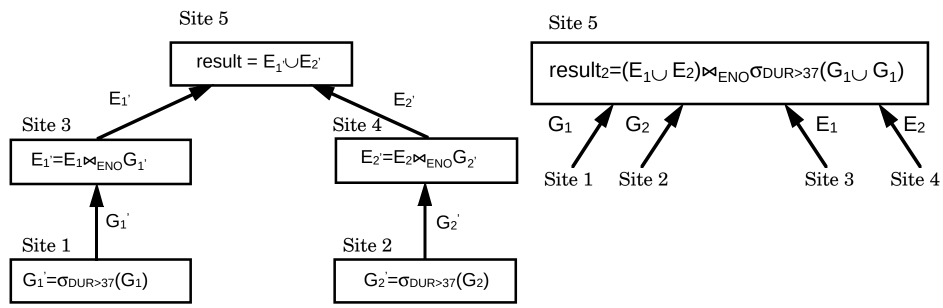
Strategy 2

$$\Pi_{ENAME}(E \bowtie_{ENO} (\sigma_{DUR>37}(G)))$$

Strategy 2 avoids Cartesian product, so is “better”

What is the Problem?

Site 1	Site 2	Site 3	Site 4	Site 5
$G_1=\sigma_{ENO\leq\text{“E3”}}(G)$	$G_2=\sigma_{ENO>\text{“E3”}}(G)$	$E_1=\sigma_{ENO\leq\text{“E3”}}(E)$	$E_2=\sigma_{ENO>\text{“E3”}}(E)$	Result



Cost of Alternatives

- Assume:
  - *size*(E) = 400, *size*(G) = 1000
  - tuple access cost = 1 unit; tuple transfer cost = 10 units
- Strategy 1

❶	produce G': 20*tuple access cost	20
❷	transfer G' to the sites of E: 20*tuple transfer cost	200
❸	produce E': (20*20)*tuple access cost*2	800
❹	transfer E' to result site: 20*tuple transfer cost	200
	Total cost	1,220

- Strategy 2

❶	transfer E to site 5:400*tuple transfer cost	4,000
❷	transfer G to site 5 :1000*tuple transfer cost	10,000
❸	produce G':1000*tuple access cost	1,000
❹	join E and G':20*1000*tuple access cost	20,000
	Total cost	35,000

Query Optimization Objectives

Minimize a cost function

I/O cost + CPU cost + communication cost

These might have different weights in different distributed environments

Wide area networks

- communication cost will dominate
  - low bandwidth
  - low speed
  - high protocol overhead
- most algorithms ignore all other cost components

Local area networks

- communication cost not that dominant
- total cost function should be considered

Can also maximize throughput

Complexity of Relational Operations

- Assume
  - relations of cardinality n
  - sequential scan

Operation	Complexity
Select Project (without duplicate elimination)	$O(n)$
Project (with duplicate elimination) Group	$O(n \log n)$
Join Semi-join Division Set Operators	$O(n \log n)$
Cartesian Product	$O(n^2)$

Query Optimization Issues – Types of Optimizers

- Exhaustive search
  - cost-based
  - optimal
  - combinatorial complexity in the number of relations
- Heuristics
  - not optimal
  - regroup common sub-expressions
  - perform selection, projection first
  - replace a join by a series of semijoins
  - reorder operations to reduce intermediate relation size
  - optimize individual operations

Query Optimization Issues – Optimization Granularity

- Single query at a time
  - cannot use common intermediate results
- Multiple queries at a time
  - efficient if many similar queries
  - decision space is much larger

Query Optimization Issues – Optimization Timing

- Static
  - compilation $\Rightarrow$ optimize prior to the execution
  - difficult to estimate the size of the intermediate results $\Rightarrow$ error propagation
  - can amortize over many executions
  - R*
- Dynamic
  - run time optimization
  - exact information on the intermediate relation sizes
  - have to reoptimize for multiple executions
  - Distributed INGRES
- Hybrid
  - compile using a static algorithm
  - if the error in estimate sizes > threshold, reoptimize at run time
  - MERMAID

Query Optimization Issues – Statistics

- Relation
  - cardinality
  - size of a tuple
  - fraction of tuples participating in a join with another relation
- Attribute
  - cardinality of domain
  - actual number of distinct values
- Common assumptions
  - independence between different attribute values
  - uniform distribution of attribute values within their domain

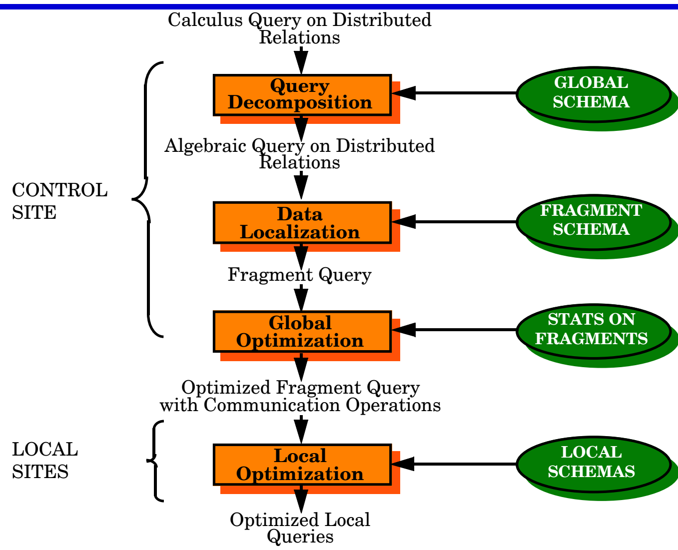
Query Optimization Issues – Decision Sites

- Centralized
  - single site determines the "best" schedule
  - simple
  - need knowledge about the entire distributed database
- Distributed
  - cooperation among sites to determine the schedule
  - need only local information
  - cost of cooperation
- Hybrid
  - one site determines the global schedule
  - each site optimizes the local subqueries

Query Optimization Issues – Network Topology

- Wide area networks (WAN) - point-to-point
  - characteristics
    - low bandwidth
    - low speed
    - high protocol overhead
  - communication cost will dominate; ignore all other cost factors
  - global schedule to minimize communication cost
  - local schedules according to centralized query optimization
- Local area networks (LAN)
  - communication cost not that dominant
  - total cost function should be considered
  - broadcasting can be exploited (joins)
  - special algorithms exist for star networks

Distributed Query Processing Methodology



Step 1 – Query Decomposition

Input : Calculus query on global relations

- Normalization
  - manipulate query quantifiers and qualification
- Analysis
  - detect and reject “incorrect” queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate redundant predicates
- Restructuring
  - calculus query ⇒ algebraic query
  - more than one translation is possible
  - use transformation rules

Normalization

- Lexical and syntactic analysis
  - check validity (similar to compilers)
  - check for attributes and relations
  - type checking on the qualification
- Put into normal form
  - Conjunctive normal form

    $(p_{11} \lor p_{12} \lor \ldots \lor p_{1n}) \land \ldots \land (p_{m1} \lor p_{m2} \lor \ldots \lor p_{mn})$
  - Disjunctive normal form

    $(p_{11} \land p_{12} \land \ldots \land p_{1n}) \lor \ldots \lor (p_{m1} \land p_{m2} \land \ldots \land p_{mn})$
  - OR's mapped into union
  - AND's mapped into join or selection

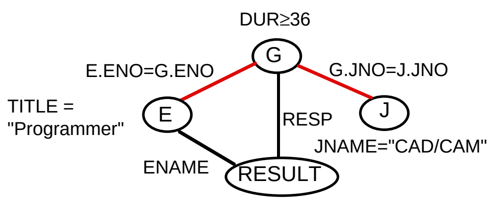
Analysis

- Refute incorrect queries
- Type incorrect
  - If any of its attribute or relation names are not defined in the global schema
  - If operations are applied to attributes of the wrong type
- Semantically incorrect
  - Components do not contribute in any way to the generation of the result
  - Only a subset of relational calculus queries can be tested for correctness
  - Those that do not contain disjunction and negation
  - To detect
    - connection graph (query graph)
    - join graph

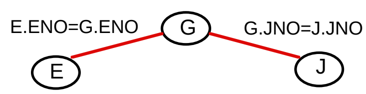
Analysis – Example

```
SELECT    ENAME,RESP
FROM      E, G, J
WHERE     E.ENO = G.ENO
AND       G.JNO = J.JNO
AND       JNAME = "CAD/CAM"
AND       DUR ≥ 36
AND       TITLE = "Programmer"
```

Query graph



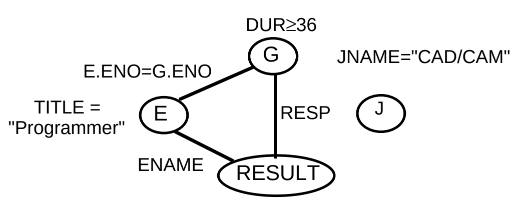
Join graph



Analysis

If the query graph is not connected, the query is wrong.

```
SELECT    ENAME,RESP
FROM      E, G, J
WHERE     E.ENO = G.ENO
AND       JNAME = "CAD/CAM"
AND       DUR ≥ 36
AND       TITLE = "Programmer"
```



Simplification

- Why simplify?
  - Remember the example
- How? Use transformation rules
  - elimination of redundancy
    - idempotency rules

      $p_1 \wedge \neg(p_1) \Leftrightarrow \text{false}$

      $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$

      $p_1 \vee \text{false} \Leftrightarrow p_1$

      …
  - application of transitivity
  - use of integrity rules

Simplification – Example

```
SELECT    TITLE
FROM      E
WHERE     E.ENAME = "J. Doe"
OR        (NOT(E.TITLE = "Programmer")
AND       (E.TITLE = "Programmer"
OR        E.TITLE = "Elect. Eng.")
AND       NOT(E.TITLE = "Elect. Eng."))
```

$\Downarrow$

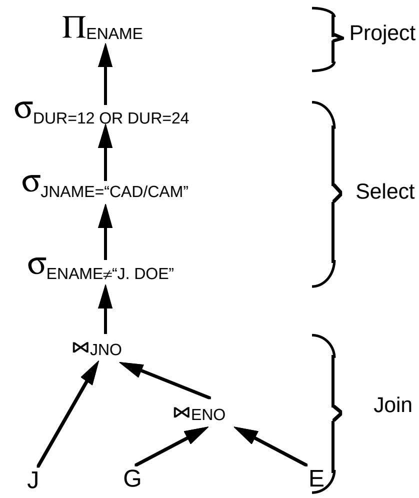
```
SELECT    TITLE
FROM      E
WHERE     E.ENAME = "J. Doe"
```

Restructuring

- Convert relational calculus to relational algebra
- Make use of query trees
- Example

  Find the names of employees other than J. Doe who worked on the CAD/CAM project for either 1 or 2 years.

```
SELECT ENAME
FROM   E, G, J
WHERE  E.ENO = G.ENO
AND    G.JNO = J.JNO
AND    ENAME ≠ "J. Doe"
AND    JNAME = "CAD/CAM"
AND    (DUR = 12 OR DUR = 24)
```



Restructuring – Transformation Rules

- Commutativity of binary operations
  - $R \times S \Leftrightarrow S \times R$
  - $R \bowtie S \Leftrightarrow S \bowtie R$
  - $R \cup S \Leftrightarrow S \cup R$
- Associativity of binary operations
  - $(R \times S) \times T \Leftrightarrow R \times (S \times T)$
  - $(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$
- Idempotence of unary operations
  - $\Pi_{A'}(\Pi_{A'}(R)) \Leftrightarrow \Pi_{A'}(R)$
  - $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) = \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$

  where $R[A]$ and $A' \subseteq A$, $A'' \subseteq A$ and $A' \subseteq A''$
- Commuting selection with projection

Restructuring – Transformation Rules

- Commuting selection with binary operations
  - $\sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)}(R)) \times S$
  - $\sigma_{p(A_i)}(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow (\sigma_{p(A_i)}(R)) \bowtie_{(A_j,B_k)} S$
  - $\sigma_{p(A_i)}(R \cup T) \Leftrightarrow \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(T)$

  where A_i belongs to R and T
- Commuting projection with binary operations
  - $\Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S)$
  - $\Pi_C(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{(A_j,B_k)} \Pi_{B'}(S)$
  - $\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$

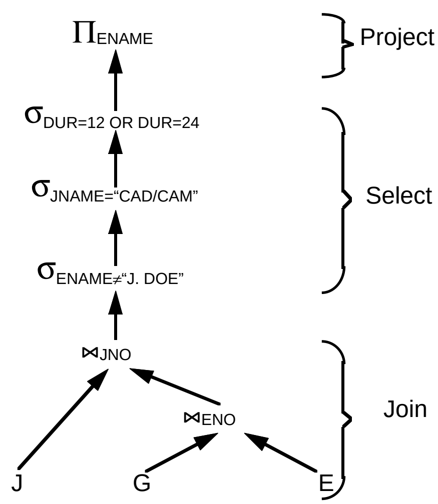
  where $R[A]$ and $S[B]$; $C = A' \cup B'$ where $A' \subseteq A$, $B' \subseteq B$

Example

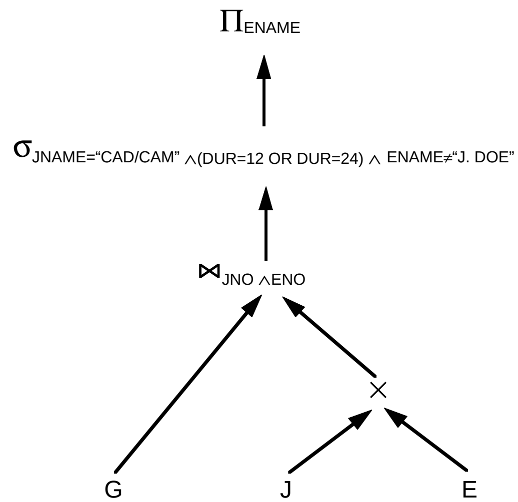
Recall the previous example:

Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years.

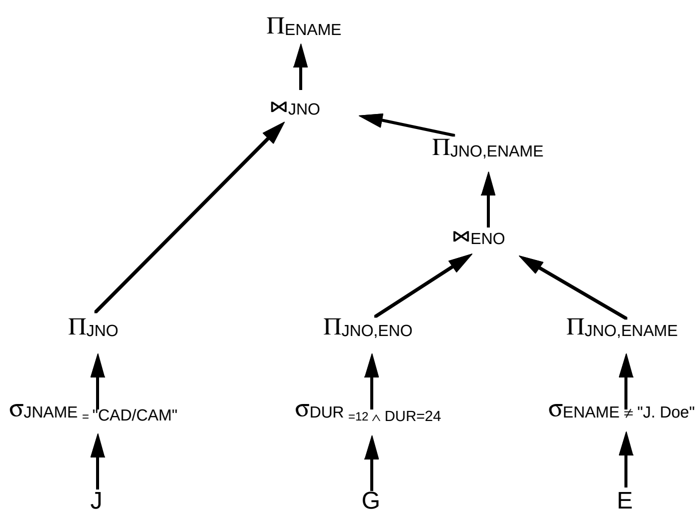
```
SELECT   ENAME
FROM     J, G, E
WHERE    G.ENO=E.ENO
AND      G.JNO=J.JNO
AND      ENAME≠"J. Doe"
AND      J.NAME="CAD/CAM"
AND      (DUR=12 OR DUR=24)
```



Equivalent Query



Restructuring



Step 2 – Data Localization

Input: Algebraic query on distributed relations

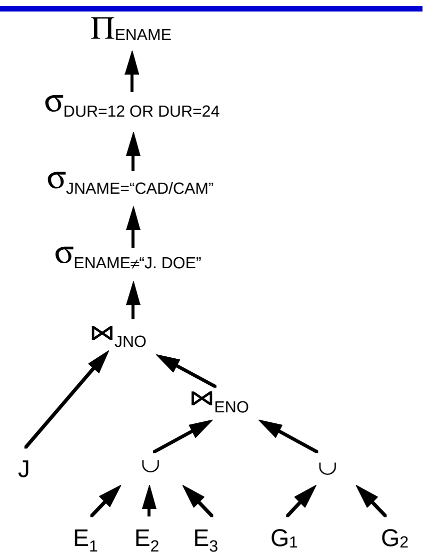
- Determine which fragments are involved
- Localization program
  - substitute for each global query its materialization program
  - optimize

Example

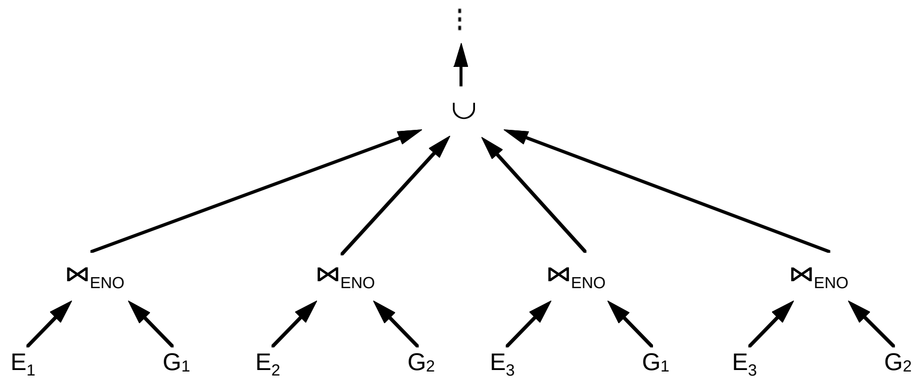
Assume

- E is fragmented into E_1, E_2, E_3 as follows:
  - $E_1 = \sigma_{ENO \leq \text{“E3”}}(E)$
  - $E_2 = \sigma_{\text{“E3”} < ENO \leq \text{“E6”}}(E)$
  - $E_3 = \sigma_{ENO \geq \text{“E6”}}(E)$
- G fragmented into G_1 and G_2 as follows:
  - $G_1 = \sigma_{ENO \leq \text{“E3”}}(G)$
  - $G_2 = \sigma_{ENO > \text{“E3”}}(G)$

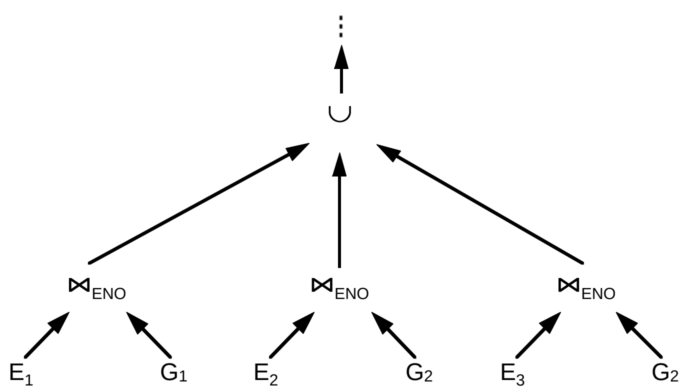
Replace E by $(E_1 \cup E_2 \cup E_3)$ and G by $(G_1 \cup G_2)$ in any query



Provides Parallellism



Eliminates Unnecessary Work

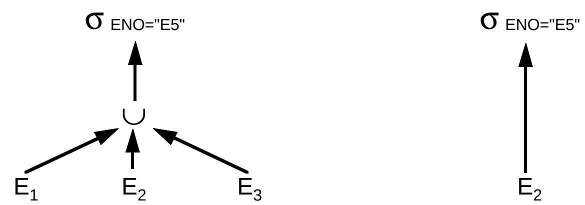


Reduction for PHF

- Reduction with selection
  - Relation R and $F_R=\{R_1, R_2, \ldots, R_w\}$ where $R_j=\sigma_{p_j}(R)$

    $\sigma_{p_i}(R_j)=\phi$ if $\forall x$ in R: $\neg(p_i(x) \wedge p_j(x))$
  - Example

```
SELECT *
FROM   E
WHERE  ENO = "E5"
```



Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over union

    $(R_1 \cup R_2) \bowtie R_3 \Leftrightarrow (R_1 \bowtie R_3) \cup (R_2 \bowtie R_3)$
  - Given $R_i = \sigma_{p_i}(R)$ and $R_j = \sigma_{p_j}(R)$

    $R_i \bowtie R_j = \phi$ if $\forall x$ in Ri, $\forall y$ in R_j: $\neg(p_i(x) \wedge p_j(y))$

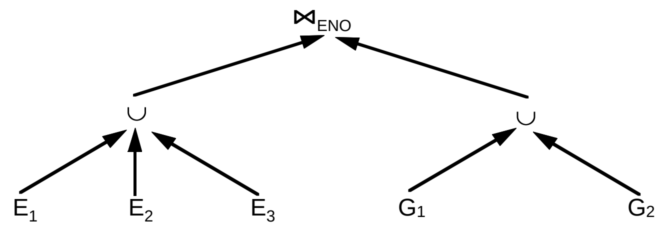
Reduction for PHF

- Reduction with join - Example
  - Assume E is fragmented as before and

    G_1: $\sigma_{ENO \leq \text{"E3"}}(G)$

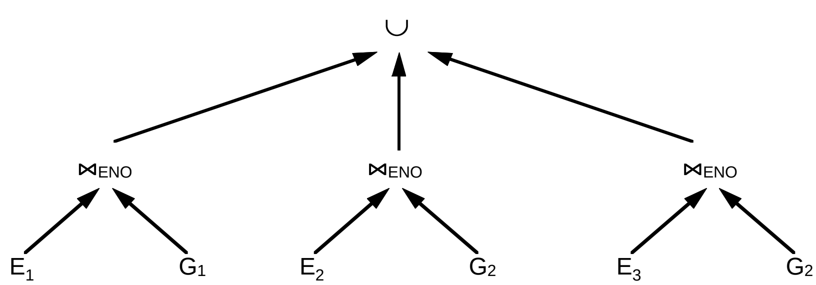
    G_2: $\sigma_{ENO > \text{"E3"}}(G)$
  - Consider the query

```
SELECT  *
FROM  E, G
WHERE E.ENO=G.ENO
```



Reduction for PHF

- Reduction with join - Example
  - Distribute join over unions
  - Apply the reduction rule



Reduction for VF

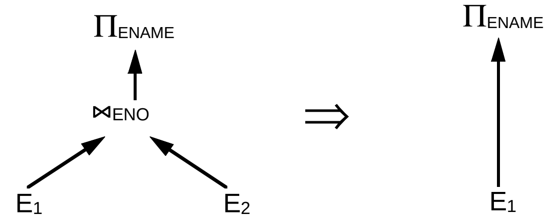
- Find useless (not empty) intermediate relations

  Relation R defined over attributes $A = \{A_1, ..., A_n\}$ vertically fragmented as $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$:

  $\Pi_{D,K}(R_i)$ is useless if the set of projection attributes D is not in A'

  Example: $E_1 = \Pi_{ENO,ENAME}(E)$; $E_2 = \Pi_{ENO,TITLE}(E)$

```
SELECT  ENAME
FROM    E
```



Reduction for DHF

- Rule :
  - Distribute joins over unions
  - Apply the join reduction for horizontal fragmentation
- Example

  G_1: $G \ltimes_{ENO} E_1$

  G_2: $G \ltimes_{ENO} E_2$

  E_1: $\sigma_{TITLE=\text{“Programmer”}}(E)$

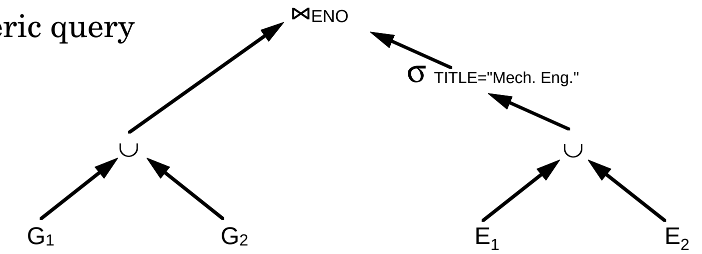
  E_2: $\sigma_{TITLE=\text{“Programmer”}}(E)$

Query

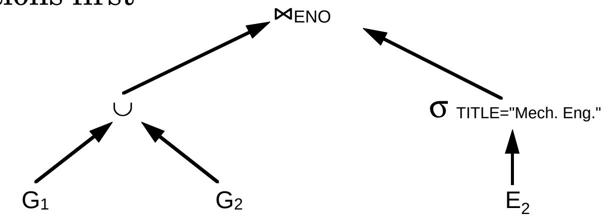
```
SELECT   *
FROM     E, G
WHERE    G.ENO = E.ENO
AND      E.TITLE = "Mech. Eng."
```

Reduction for DHF

Generic query

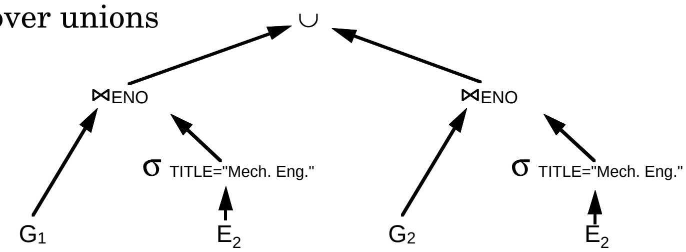


Selections first

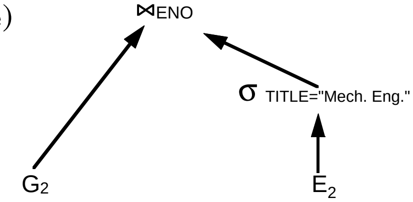


Reduction for DHF

Joins over unions



Elimination of the empy intermediate relations (left sub-tree)



Reduction for HF

- Combine the rules already specified:
  - Remove empty relations generated by contradicting selections on horizontal fragments;
  - Remove useless relations generated by projections on vertical fragments;
  - Distribute joins over unions in order to isolate and remove useless joins.

Reduction for HF

Example

Consider the following hybrid fragmentation:

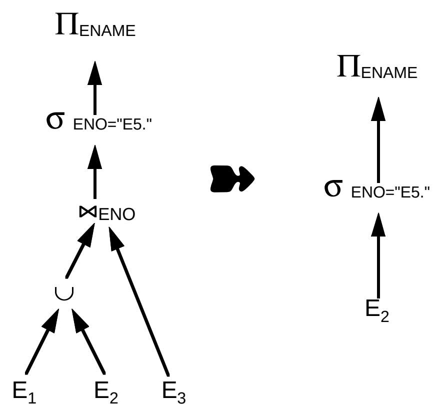
$E_1 = \sigma_{ENO \le "E4"} (\Pi_{ENO,ENAME}(E))$

$E_2 = \sigma_{ENO > "E4"} (\Pi_{ENO,ENAME}(E))$

$E_3 = \Pi_{ENO,TITLE}(E)$

and the query

```
SELECT    ENAME
FROM      E
WHERE     ENO="E5"
```



Step 3 – Global Query Optimization

Input: Fragment query

- Find the *best* (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
    - Bushy vs. linear trees
    - Which relation to ship where?
    - Ship-whole vs ship-as-needed
  - Decide on the use of semijoins
    - Semijoin saves on communication at the expense of more local processing.
  - Join methods
    - nested loop vs ordered joins (merge join or hash join)

Cost-Based Optimization

- Solution space
  - The set of equivalent algebra expressions (query trees).
- Cost function (in terms of time)
  - I/O cost + CPU cost + communication cost
  - These might have different weights in different distributed environments (LAN vs WAN).
  - Can also maximize throughput
- Search algorithm
  - How do we move inside the solution space?
  - Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic,…)

Cost Functions

- Total Time (or Total Cost)
  - Reduce each cost (in terms of time) component individually
  - Do as little of each cost component as possible
  - Optimizes the utilization of the resources

$$\Downarrow$$

Increases system throughput

- Response Time
  - Do as many things as possible in parallel
  - May increase total time because of increased total activity

Total Cost

Summation of all cost factors

Total cost = CPU cost + I/O cost + communication cost

CPU cost = unit instruction cost ∗ no.of instructions

I/O cost = unit disk I/O cost ∗ no. of disk I/Os

communication cost = message initiation + transmission

Total Cost Factors

- Wide area network
  - message initiation and transmission costs high
  - local processing cost is low (fast mainframes or minicomputers)
  - ratio of communication to I/O costs = 20:1
- Local area networks
  - communication and local processing costs are more or less equal
  - ratio = 1:1.6

Response Time

Elapsed time between the initiation and the completion of a query

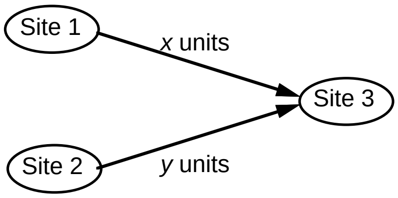
Response time = CPU time + I/O time + communication time

CPU time = unit instruction time ∗ no. of sequential instructions

I/O time = unit I/O time ∗ no. of sequential I/Os

communication time = unit msg initiation time ∗ no. of sequential msg + unit transmission time ∗ no. of sequential bytes

Example



Assume that only the communication cost is considered

Total time = 2 $*$ message initialization time + unit transmission time $*$ $(x+y)$

Response time = max {time to send x from 1 to 3, time to send y from 2 to 3}

time to send x from 1 to 3 = message initialization time + unit transmission time $*$ x

time to send y from 2 to 3 = message initialization time + unit transmission time $*$ y

Optimization Statistics

- Primary cost factor: size of intermediate relations
- Make them precise ⇒ more costly to maintain
  - For each relation $R[A_1, A_2, \ldots, A_n]$ fragmented as $R_1, \ldots, R_r$
    - length of each attribute: $length(Ai)$
    - the number of distinct values for each attribute in each fragment: $card(\prod_{A_i} R_j)$
    - maximum and minimum values in the domain of each attribute: $min(A_i)$, $max(A_i)$
    - the cardinalities of each domain: $card(dom[A_i])$
    - the cardinalities of each fragment: $card(R_j)$
  - Selectivity factor of each operation for relations
    - For joins

$$SF_{\bowtie}(R,S) = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

Intermediate Relation Sizes

Selection

$size(R) = card(R) * length(R)$

$card(\sigma_F(R)) = SF_\sigma(F) * card(R)$

where

$$S\,F_\sigma(A = value) = \frac{1}{card(\prod_A(R))}$$

$$S\,F_\sigma(A > value) = \frac{max(A) - value}{max(A) - min(A)}$$

$$S\,F_\sigma(A < value) = \frac{value - max(A)}{max(A) - min(A)}$$

$$SF_\sigma(p(A_i) \wedge p(A_j)) = SF_\sigma(p(A_i)) * SF_\sigma(p(A_j))$$

$$SF_\sigma(p(A_i) \vee p(A_j)) = SF_\sigma(p(A_i)) + SF_\sigma(p(A_j)) - (SF_\sigma(p(A_i)) * SF_\sigma(p(A_j)))$$

$$SF_\sigma(A \in value) = SF_\sigma(A = value) * card(\{values\})$$

Intermediate Relation Sizes

Projection

$card(\prod_A(R)) = card(R)$

Cartesian Product

$card(R \times S) = card(R) * card(S)$

Union

upper bound: $card(R \cup S) = card(R) + card(S)$

lower bound: $card(R \cup S) = max\{card(R), card(S)\}$

Set Difference

upper bound: $card(R - S) = card(R)$

lower bound: 0

Intermediate Relation Size

Join

- Special case: A is a key of R and B is a foreign key of S; A is a foreign key of R and B is a key of S

$$card(R \bowtie_{A=B} S) = card(R)$$

- More genera:l

$$card(R \bowtie S) = SF_{\bowtie} * card(R) * card(S)$$

Semijoin

$$card(R \ltimes_A S) = SF_{\ltimes}(S.A) * card(R)$$

where

$$SF_{\ltimes}(R \ltimes_A S) = SF_{\ltimes}(S.A) = \frac{card(\prod_A(S))}{card(dom[A])}$$

Centralized Query Optimization

- INGRES
  - dynamic
  - interpretive
- System R
  - static
  - exhaustive search

INGRES Algorithm

❶ Decompose each multi-variable query into a sequence of mono-variable queries with a common variable

❷ Process each by a one variable query processor

- Choose an initial execution plan (heuristics)
- Order the rest by considering intermediate relation sizes

$$\Downarrow$$

No statistical information is maintained

INGRES Algorithm–Decomposition

- Replace an n variable query q by a series of queries

$$q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_n$$

where q_i uses the result of q_{i-1}.

- Detachment
  - Query q decomposed into $q' \rightarrow q''$ where q' and q'' have a common variable which is the result of q'
- Tuple substitution
  - Replace the value of each tuple with actual values and simplify the query

$$q(V_1, V_2, \dots V_n) \rightarrow (q'(t_1, V_2, V_2, \dots, V_n), t_1 \in R)$$

Detachment

q: **SELECT** $V_2.A_2, V_3.A_3, \ldots, V_n.A_n$
FROM $R_1\ V_1, \ldots, R_n\ V_n$
WHERE $P_1(V_1.A_{1'})$ **AND** $P_2(V_1.A_1, V_2.A_2, \ldots, V_n.A_n)$

$$\Downarrow$$

q': **SELECT** $V_1.A_1$ **INTO** R_1'
FROM $R_1\ V_1$
WHERE $P_1(V_1.A_1)$

q'': **SELECT** $V_2.\ A_2, \ldots, V_n.A_n$
FROM $R_1'\ V_1, R_2\ V_2, \ldots, R_n\ V_n$
WHERE $P_2(V_1.A_1, V_2.A_2, \ldots, V_n.A_n)$

Detachment Example

Names of employees working on CAD/CAM project

q_1:
```
SELECT   E.ENAME
FROM     E, G, J
WHERE    E.ENO=G.ENO
AND      G.JNO=J.JNO
AND      J.JNAME="CAD/CAM"
```

$$\Downarrow$$

q_{11}:
```
SELECT   J.JNO INTO JVAR
FROM     J
WHERE    J.JNAME="CAD/CAM"
```

q':
```
SELECT   E.ENAME
FROM     E,G,JVAR
WHERE    E.ENO=G.ENO
AND      G.JNO=JVAR.JNO
```

Detachment Example (cont'd)

q':
```
SELECT  E.ENAME
FROM    E,G,JVAR
WHERE   E.ENO=G.ENO
AND     G.JNO=JVAR.JNO
```

$\Downarrow$

q_{12}:
```
SELECT  G.ENO INTO GVAR
FROM    G,JVAR
WHERE   G.JNO=JVAR.JNO
```

q_{13}:
```
SELECT  E.ENAME
FROM    E,GVAR
WHERE   E.ENO=GVAR.ENO
```


Tuple Substitution

q_{11} is a mono-variable query

q_{12} and q_{13} is subject to tuple substitution

Assume GVAR has two tuples only: <E1> and <E2>

Then q_{13} becomes

q_{131}:
```
SELECT  E.ENAME
FROM    E
WHERE   E.ENO="E1"
```

q_{132}:
```
SELECT  E.ENAME
FROM    E
WHERE   E.ENO="E2"
```

System R Algorithm

❶ Simple (i.e., mono-relation) queries are executed according to the best access path

❷ Execute joins

2.1 Determine the possible ordering of joins

2.2 Determine the cost of each ordering

2.3 Choose the join ordering with minimal cost

System R Algorithm

For joins, two alternative algorithms :

- Nested loops

```
for each tuple of external relation (cardinality n1)
  for each tuple of internal relation (cardinality n2)
    join two tuples if the join predicate is true
  end
end
```

  ➠ Complexity: $n_1 * n_2$

- Merge join

  sort relations

  merge relations

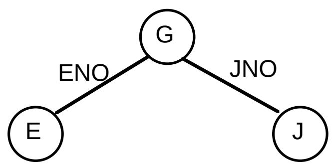
  ➠ Complexity: $n_1 + n_2$ if relations are previously sorted and equijoin

System R Algorithm – Example

Names of employees working on the CAD/CAM project

Assume

- E has an index on ENO,
- G has an index on JNO,
- J has an index on JNO and an index on JNAME



System R Example (cont'd)

❶ Choose the best access paths to each relation

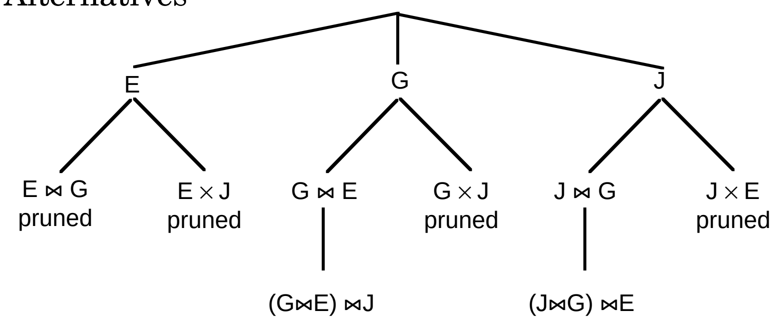
- E: sequential scan (no selection on E)
- G: sequential scan (no selection on G)
- J: index on JNAME (there is a selection on J based on JNAME)

❷ Determine the best join ordering

- E $\bowtie$ G $\bowtie$ J
- G $\bowtie$ J $\bowtie$ E
- J $\bowtie$ G $\bowtie$ E
- G $\bowtie$ E $\bowtie$ J
- E $\times$ J $\bowtie$ G
- J $\times$ E $\bowtie$ G
- Select the best ordering based on the join costs evaluated according to the two methods

System R Algorithm

Alternatives



Best total join order is one of

((G ⋈ E) ⋈ J)

((J ⋈ G) ⋈ E)

System R Algorithm

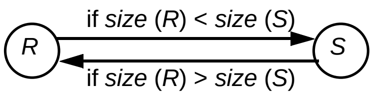
- ((J ⋈ G) ⋈ E) has a useful index on the select attribute and direct access to the join attributes of G and E
- Therefore, chose it with the following access methods:
  - select J using index on JNAME
  - then join with G using index on JNO
  - then join with E using index on ENO

Join Ordering in Fragment Queries

- Ordering joins
  - Distributed INGRES
  - System R*
- Semijoin ordering
  - SDD-1
  - Apers-Hevner-Yao Algorithms

Join Ordering

- Consider two relations only

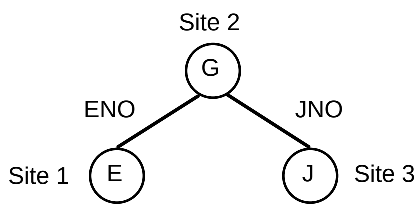


- Multiple relations more difficult because too many alternatives.
  - Compute the cost of all alternatives and select the best one.
    - Necessary to compute the size of intermediate relations which is difficult.
  - Use heuristics

Join Ordering – Example

Consider

$J \bowtie_{JNO} E \bowtie_{ENO} G$



Join Ordering – Example

Execution alternatives:

1. E → Site 2
   Site 2 computes E'=E $\bowtie$ G
   E' → Site 3
   Site 3 computes E' $\bowtie$ J
2. G → Site 1
   Site 1 computes E'=E $\bowtie$ G
   E' → Site 3
   Site 3 computes E' $\bowtie$ J
3. G → Site 3
   Site 3 computes G'=G $\bowtie$ J
   G' → Site 1
   Site 1 computes G' $\bowtie$ E
4. J → Site 2
   Site 2 computes J'=J $\bowtie$ G
   J' → Site 1
   Site 1 computes J' $\bowtie$ E
5. E → Site 2
   J → Site 2
   Site 2 computes E $\bowtie$J$\bowtie$ G

Semijoin Algorithms

- Consider the join of two relations:
  - $R[A]$ (located at site 1)
  - $S[A]$ (located at site 2)
- Alternatives:

1 Do the join $R \bowtie_A S$

2 Perform one of the semijoin equivalents

$$\begin{aligned} R \bowtie_A S &\Leftrightarrow (R \ltimes_A S) \bowtie_A S \\ &\Leftrightarrow R \bowtie_A (S \ltimes_A R) \\ &\Leftrightarrow (R \ltimes_A S) \bowtie_A (S \ltimes_A R) \end{aligned}$$

Semijoin Algorithms

- Perform the join
  - send R to Site 2
  - Site 2 computes $R \bowtie_A S$
- Consider semijoin $(R \ltimes_A S) \bowtie_A S$
  - $S' \leftarrow \prod_A(S)$
  - $S' \rightarrow$ Site 1
  - Site 1 computes $R' = R \ltimes_A S'$
  - $R' \rightarrow$ Site 2
  - Site 2 computes $R' \bowtie_A S$

Semijoin is better if

$$size(\prod_A(S)) + size(R \ltimes_A S)) < size(R)$$

Distributed Query Processing

Algorithms	Optm. Timing	Objective Function	Optm. Factors	Network Topology	Semi Joins	Statistics *	Fragments
Distributed INGRES	Dynamic	Response Time or Total Cost	Msg Size, Processing	General or Broadcast	no	1	Horizontal
R *	Static	Total Cost	#Msg, Msg size, IO, CPU	General or Local	no	1, 2	No
SDD–1	Static	Total Cost	Msg size,	General	yes	1, 3, 4, 5	No
AHY	Static	Response Time or Total Cost	#Msg Msg size,	General	yes	1, 3, 5	No

* 1: relation cardinality, 2: number of unique values per attribute, 3: join selectivity factor, 4: size of projection on each join attribute, 5: attribute size and tuple size

Distributed INGRES Algorithm

Same as the centralized version except

- Movement of relations (and fragments) need to be considered
- Optimization with respect to communication cost or response time possible

R* Algorithm

- Cost function includes local processing as well as transmission
- Considers only joins
- Exhaustive search
- Compilation
- Published papers provide solutions to handling horizontal and vertical fragmentations but the implemented prototype does not

R* Algorithm

Performing joins

- Ship whole
  - larger data transfer
  - smaller number of messages
  - better if relations are small
- Fetch as needed
  - number of messages = O(cardinality of external relation)
  - data transfer per message is minimal
  - better if relations are large and the selectivity is good

R* Algorithm – Vertical Partitioning & Joins

1. Move outer relation tuples to the site of the inner relation

   (a) Retrieve outer tuples

   (b) Send them to the inner relation site

   (c) Join them as they arrive

Total Cost = cost(retrieving qualified outer tuples)

+ no. of outer tuples fetched ∗ cost(retrieving qualified inner tuples)

+ msg. cost ∗ (no. outer tuples fetched ∗ avg. outer tuple size) / msg. size

R* Algorithm – Vertical Partitioning & Joins

2. Move inner relation to the site of outer relation

cannot join as they arrive; they need to be stored

Total Cost = cost(retrieving qualified outer tuples)

+ no. of outer tuples fetched ∗ cost(retrieving matching inner tuples from temporary storage)

+ cost(retrieving qualified inner tuples)

+ cost(storing all qualified inner tuples in temporary storage)

+ msg. cost ∗ (no. of inner tuples fetched ∗ avg. inner tuple size) / msg. size

R* Algorithm – Vertical Partitioning & Joins

3. Move both inner and outer relations to another site

Total cost = cost(retrieving qualified outer tuples)

+ cost(retrieving qualified inner tuples)

+ cost(storing inner tuples in storage)

+ msg. cost ∗ (no. of outer tuples fetched ∗ avg. outer tuple size) / msg. size

+ msg. cost ∗ (no. of inner tuples fetched ∗ avg. inner tuple size) / msg. size

+ no. of outer tuples fetched ∗ cost(retrieving inner tuples from temporary storage)

R* Algorithm – Vertical Partitioning & Joins

4. Fetch inner tuples as needed
   (a) Retrieve qualified tuples at outer relation site
   (b) Send request containing join column value(s) for outer tuples to inner relation site
   (c) Retrieve matching inner tuples at inner relation site
   (d) Send the matching inner tuples to outer relation site
   (e) Join as they arrive

Total Cost = cost(retrieving qualified outer tuples)

+ msg. cost ∗ (no. of outer tuples fetched)

+ no. of outer tuples fetched ∗ (no. of inner tuples fetched ∗ avg. inner tuple size ∗ msg. cost / msg. size)

+ no. of outer tuples fetched ∗ cost(retrieving matching inner tuples for one outer value)

SDD-1 Algorithm

- Based on the Hill Climbing Algorithm
  - No semijoins
  - No replication
  - No fragmentation
  - Cost of transferring the result to the user site from the final result site is not considered
  - Can minimize either total time or response time

Hill Climbing Algorithm

Assume join is between three relations.

Step 1: Do initial processing

Step 2: Select initial feasible solution (ES_0)

- 2.1 Determine the candidate result sites - sites where a relation referenced in the query exist
- 2.2 Compute the cost of transferring all the other referenced relations to each candidate site
- 2.3 ES_0 = candidate site with minimum cost

Step 3: Determine candidate splits of ES_0 into $\{ES_1, ES_2\}$

- 3.1 ES_1 consists of sending one of the relations to the other relation's site
- 3.2 ES_2 consists of sending the join of the relations to the final result site

Hill Climbing Algorithm

Step 4: Replace ES_0 with the split schedule which gives

$$cost(ES_1) + cost(\text{local join}) + cost(ES_2) < cost(ES_0)$$

Step 5: Recursively apply steps 3–4 on ES_1 and ES_2 until no such plans can be found

Step 6: Check for redundant transmissions in the final plan and eliminate them.

Hill Climbing Algorithm – Example

What are the salaries of engineers who work on the CAD/CAM project?

$$\Pi_{SAL}(S \bowtie_{TITLE}(E \bowtie_{ENO}(G \bowtie_{JNO}(\sigma_{JNAME="CAD/CAM"}(J)))))$$

Relation	Size	Site
E	8	1
S	4	2
J	4	3
G	10	4

Assume:

- Size of relations is defined as their cardinality
- Minimize total cost
- Transmission cost between two sites is 1
- Ignore local processing cost

Hill Climbing Algorithm – Example

Step 1:

Selection on J; result has cardinality 1

Relation	Size	Site
E	8	1
S	4	2
J	1	3
G	10	4

Hill Climbing Algorithm – Example

Step 2: Initial feasible solution

Alternative 1: Resulting site is Site 1

Total cost = $cost(\mathsf{S} \rightarrow \text{Site 1}) + cost(\mathsf{G} \rightarrow \text{Site 1}) + cost(\mathsf{J} \rightarrow \text{Site 1})$

= 4 + 10 + 1 = 15

Alternative 2: Resulting site is Site 2

Total cost = 8 + 10 + 1 = 19

Alternative 3: Resulting site is Site 3

Total cost = 8 + 4 + 10 = 22

Alternative 4: Resulting site is Site 4

Total cost = 8 + 4 + 1 = 13

Therefore $ES_0 = \{\mathsf{E} \rightarrow \text{Site 4}; \mathsf{S} \rightarrow \text{Site 4}; \mathsf{J} \rightarrow \text{Site 4}\}$

Hill Climbing Algorithm – Example

Step 3: Determine candidate splits

Alternative 1: $\{ES_1, ES_2, ES_3\}$ where

ES_1: E $\rightarrow$ Site 2

ES_2: (E $\bowtie$ S) $\rightarrow$ Site 4

ES_3: J $\rightarrow$ Site 4

Alternative 2: $\{ES_1, ES_2, ES_3\}$ where

ES_1: S $\rightarrow$ Site 1

ES_2: (S $\bowtie$ E) $\rightarrow$ Site 4

ES_3: J $\rightarrow$ Site 4

Hill Climbing Algorithm – Example

Step 4: Determine costs of each split alternative

$$\begin{aligned} cost(\text{Alternative 1}) &= cost(\text{E} \rightarrow \text{Site 2}) + cost((\text{E} \bowtie \text{S}) \rightarrow \text{Site 4}) + \\ &\quad cost(\text{J} \rightarrow \text{Site 4}) \\ &= 8 + 8 + 1 = 17 \\ cost(\text{Alternative 2}) &= cost(\text{S} \rightarrow \text{Site 1}) + cost((\text{S} \bowtie \text{E}) \rightarrow \text{Site 4}) + \\ &\quad cost(\text{J} \rightarrow \text{Site 4}) \\ &= 4 + 8 + 1 = 13 \end{aligned}$$

Decision : DO NOT SPLIT

Step 5: ES_0 is the "best".

Step 6: No redundant transmissions.

Hill Climbing Algorithm

Problems :

❶ Greedy algorithm → determines an initial feasible solution and iteratively tries to improve it

❷ If there are local minimas, it may not find global minima

❸ If the optimal schedule has a high initial cost, it won't find it since it won't choose it as the initial feasible solution

Example : A better schedule is

J → Site 4

G' = (J ⋉ G) → Site 1

(G' ⋉ E) → Site 2

Total cost = 1 + 2 + 2 = 5

SDD-1 Algorithm

Initialization

Step 1: In the execution strategy (call it *ES*), include all the local processing

Step 2: Reflect the effects of local processing on the database profile

Step 3: Construct a set of beneficial semijoin operations (*BS*) as follows :

$BS = \emptyset$

For each semijoin SJ_i

$BS \leftarrow BS \cup SJ_i$ if $cost(SJ_i) < benefit(SJ_i)$

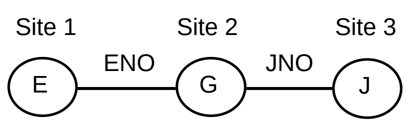
© 1994 M. Tamer Özsu & Patrick Valduriez

SDD-1 Algorithm – Example

Consider the following query

```
SELECT   *
FROM     E, G, J
WHERE    E.ENO = G.ENO
AND      G.JNO = J.JNO
```

which has the following query graph and statistics:



relation	card	tuple size	relation size
E	30	50	1500
G	100	30	3000
J	50	40	2000

attribute	$SF_{\ltimes}$	$size(\Pi_{attribute})$
E.ENO	.3	120
G.ENO	.8	400
G.JNO	1	400
J.JNO	.4	200

SDD-1 Algorithm – Example

- Beneficial semijoins:
  - $SJ_1 = G \ltimes E$, whose benefit is 2100 = (1 – 0.3)∗3000 and cost is 120
  - $SJ_2 = G \ltimes J$, whose benefit is 1800 = (1 – 0.4) ∗3000 and cost is 200
- Nonbeneficial semijoins:
  - $SJ_3 = E \ltimes G$, whose benefit is 300 = (1 – 0.8) ∗1500 and cost is 400
  - $SJ_4 = J \ltimes G$, whose benefit is 0 and cost is 400

SDD-1 Algorithm

Iterative Process

Step 4: Remove the most beneficial SJ_i from BS and append it to ES

Step 5: Modify the database profile accordingly

Step 6: Modify BS appropriately

- compute new benefit/cost values
- check if any new semijoin need to be included in BS

Step 7: If $BS \neq \emptyset$, go back to Step 4.

SDD-1 Algorithm – Example

- Iteration 1:
  - Remove SJ_1 from BS and add it to ES.
  - Update statistics of G

    $size$(G) = 900 (= 3000∗0.3)

    $SF_{\ltimes}$(G.ENO) = ~0.8∗0.3 = 0.24
- Iteration 2:
  - Two beneficial semijoins:

    SJ_2 = G' ⋉J, whose benefit is 540 = (1–0.4) ∗900 and cost is 200

    SJ_3 = E ⋉G', whose benefit is 1400 = (1–0.24) ∗1500 and cost is 400
  - Add SJ_3 to ES
  - Update statistics of E

    $size$(E) = 360 (= 1500∗0.24)

    $SF_{\ltimes}$(E.ENO) = ~0.3∗0.24 = 0.072

SDD-1 Algorithm – Example

- Iteration 3:
  - No new beneficial semijoins.
  - Remove remaining beneficial semijoin SJ_2 from *BS* and add it to *ES*.
  - Update statistics of G

    size(G) = 360 (= 900*0.4)

    Note: selectivity of G may also change, but not important in this example.

SDD-1 Algorithm

Assembly Site Selection

Step 8: Find the site where the largest amount of data resides and select it as the assembly site

Example:

Amount of data stored at sites:

Site 1: 360

Site 2: 360

Site 3: 2000

Therefore, Site 3 will be chosen as the assembly site.

SDD-1 Algorithm

Postprocessing

Step 9: For each R_i at the assembly site, find the semijoins of the type

$$R_i \ltimes R_j$$

where the total cost of ES without this semijoin is smaller than the cost with it and remove the semijoin from ES.

Note : There might be indirect benefits.

- Example: No semijoins are removed.

Step 10: Permute the order of semijoins if doing so would improve the total cost of ES.

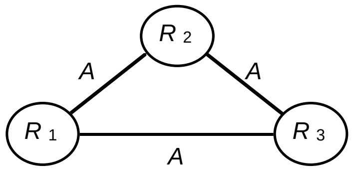
- Example: Final strategy:
  Send (G ⋉ E) ⋉ J to Site 3
  Send E ⋉ G' to Site 3

Apers-Hevner-Yao Algorithms

- User specified result site
- Makes use of semijoins
- Can be used for total time minimization or response time minimization
- Considers transmission cost only
- Simple queries
  - Those queries where, after initial local processing, each relation in the query contains only the common join attribute, which is also the only output of the query.
- General queries
  - Any good old query.

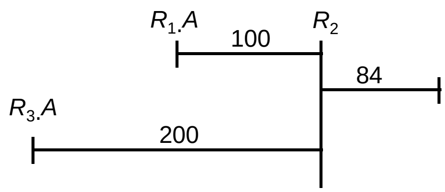
AHY Algorithms – Representation

Consider the following simple query and its statistics



attribute	$SF_{\ltimes}$	$size(\Pi_{attribute})$
$R_1.A$	.3	100
$R_2.A$	1	400
$R_3.A$	.7	200

It can be represented by the following schedule



Simple Query Optimization

Total Time Minimization

Move smaller relations to the larger ones

Algorithm SERIAL

Step 1: Order relations such that

$$size(R_1) \leq size(R_2) \leq \ldots \leq size(R_n)$$

Step 2: Assume R_r is the relation at the result site

Compare the cost of

$$R_1 \to R_2 \to R_3 \to \ldots \to R_r \to \ldots \to R_n \to R_r$$

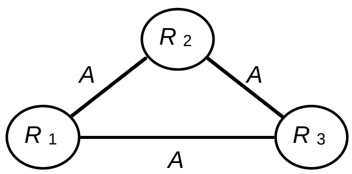
versus that of

$$R_1 \to R_2 \to \ldots \to R_{r-1} \to R_{r+1} \to \ldots \to R_n \to R_r$$

Step 3: Select the one with the smaller cost.

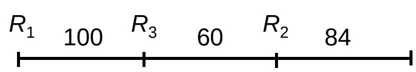
Simple Query Optimization – Example

Consider the same example query and assume that message initiation cost is 0, and unit message transmission cost is 1.



attribute	$SF_{\ltimes}$	$size(\Pi_{attribute})$
$R_1.A$	.3	100
$R_2.A$	1	400
$R_3.A$	.7	200

The optimal execution schedule is:



Simple Query Optimization

Response Time Minimization

Start with an initial feasible solution (IFS) an improve

Algorithm PARALLEL

Step 1: Order relations such that

$$size(R_1) \leq size(R_2) \leq \ldots \leq size(R_n)$$

Step 2: IFS: Transmit all relations *in parallel* to the result site. Response time is

$$max\{\text{transmission } cost(R_i), 1 \leq i \leq n\}$$

Step 3: Set $i = 1$

Simple Query Optimization

Step 4: Pick R_i. Compute the cost of transmitting R_j, $\forall j < i$, and all other R_k $(k < j)$ to R_i in parallel. Let this cost be denoted by $cost(R_{ji})$.

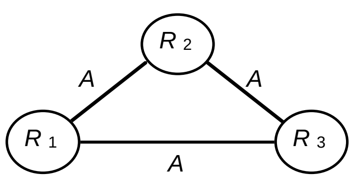
Step 5: Repeat Step 4 for all $i \leq n$.

Step 6: Select the schedule with minimum $cost(R_{ji})$ as the new feasible solution

Step 7: Eliminate redundant schedules.

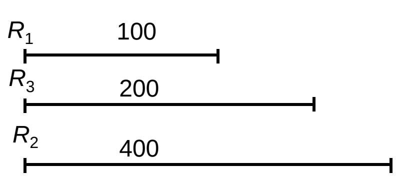
Simple Query Optimization – Example

Again consider the same example query:



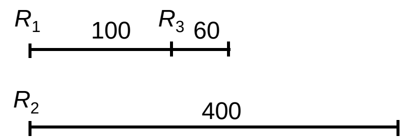
attribute	$SF_{\ltimes}$	$size(\Pi_{attribute})$
$R_1.A$	.3	100
$R_2.A$	1	400
$R_3.A$	.7	200

The initial feasible solution is:

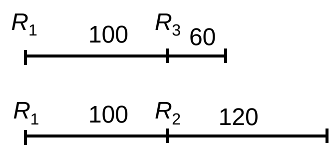


Simple Query Optimization – Example

Schedule for R_1 cannot be improved (it is the smallest). After improving the schedule of R_3:



After improving the schedule of $R2$:



General Query Optimization

- Assume n join attributes are present in query
- Treat each join attribute in isolation
  - decompose into n simple queries
  - apply SERIAL or PARALLEL to each simple query
- Integrate the results

AHY – General Query Optimization

Step 1: Local processing at each site

Step 2: For each join attribute generate the set of candidate strategies

- 2.1 Isolate the simple query
- 2.2 Apply SERIAL or PARALLEL

Step 3: Order candidate strategies for each relation on the join attribute in incresing order of cost

AHY – General Query Optimization

Step 4: Response time

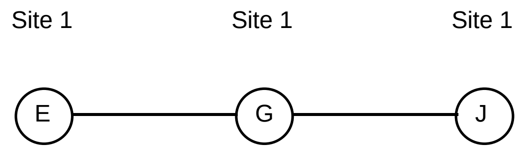
- For each candidate ES_{ij} (for relation R_i)
  - perform ES_{ij}
  - move all ES_{ik} ($k < j$) to R_i
  - reduce R_i
  - move R_i to the result site
- Pick alternative with minimum cost

Total time

- Similar

Step 5: Eliminate redundancies.

AHY – General Query Example

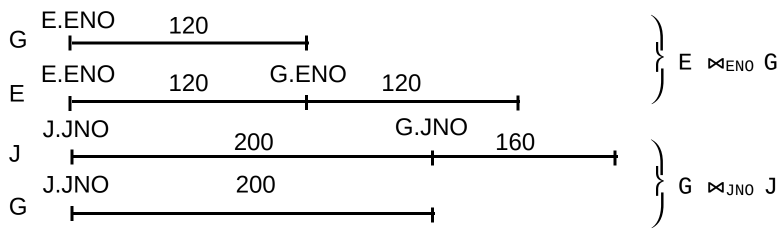


Relation	Card	Tuple size	Rel. size
E	30	50	1500
G	100	30	3000
J	50	40	2000

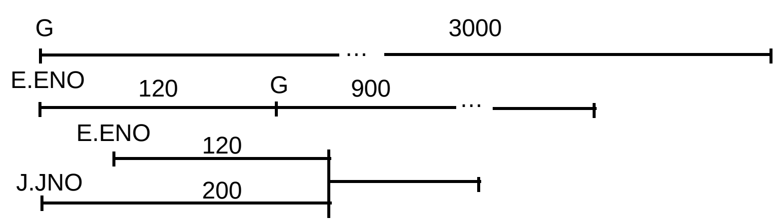
Attribute	$SF_{\ltimes}$	$size(\Pi_{attribute})$
E.ENO	.3	120
G.ENO	.8	400
G.JNO	.1	400
J.JNO	.4	200

AHY – General Query Example

Best strategies for response time minimization



Common relation is G to reduce. Alternatives:



Step 4 – Local Optimization

Input: Best global execution schedule

- Select the best access path
- Use the centralized optimization techniques

Distributed Query Optimization Problems

- Cost model
  - multiple query optimization
  - heuristics to cut down on alternatives
- Larger set of queries
  - optimization only on select-project-join queries
  - also need to handle complex queries (e.g., unions, disjunctions, aggregations and sorting)
- Optimization cost vs execution cost tradeoff
  - heuristics to cut down on alternatives
  - controllable search strategies
- Optimization/reoptimization interval
  - extent of changes in database profile before reoptimization is necessary