Data Management for Large Rule Systems

Arie Segev and J. Leon Zhao

Haas School of Business, Univ. of California at Berkeley, and
Information and Computing Sciences Division, Lawrence Berkeley Laboratory,
Berkeley, California 94720

Abstract

Managing data in large rule systems is a critical issue, and DBMS systems are being extended for the support of rule-based, data-intensive decision making such as in expert system applications. We suggest to selectively materialize the rule-generated data in relations so that rule-based decisions can be made incrementally and automatically when the collected data are updated. An algorithm is developed to select derived relations for materialization so that the overall cost of processing the inference rules is minimized while satisfying requirements on query response time.

1. Introduction

Expert Systems (ESs) are being applied to many business operations, including accounting (Shim & Rice, 1988), finance (Srinivasan & Kim, 1988; Duchessi, Shawky & Seagle, 1988 and Shaw & Gentry, 1988), human resource (Extejt & Lynn, 1988), and production (Rao & Lingarai, 1988). One of the problems ES developers are facing is that access to large amounts of business data is difficult because current ES technology lacks necessary data management functions found in database management systems (DBMSs).

To support decision making while using large amounts of data, DBMS and ES technologies are being integrated into *expert database systems* (EDSs). New DBMS systems are being designed with EDS features, that is, managing both rules and data in a unified system. Relational DBMSs are not designed to handle inference rules and their functions must be extended to address several important issues.

Rules are symbolic data with special internal and external structure. They must be managed in a special way so that they can be defined, stored, and accessed efficiently. This is the *rule storage issue.*

When data are inserted or modified by an transaction, certain rules may be called upon to act on the change of situation. Therefore, relevant rules must be identified quickly. This is the *rule awakening issue.*

When rules are awakened, data must be derived and stored efficiently. This process can be complex if the rule derived data affect other rules. For relevant rules that are conditional on join expressions, appropriate techniques must be used to ensure efficient rule execution. This is the *rule execution issue*, and is addressed in this paper.

Current research and prototype systems deal with the above issues. Among those systems are POSTGRES (Stonebraker, Hason & Potaminos, 1988), Starburst (Haas et al., 1990), Iris (Wilkinson, Lyngbek & Hasan, 1990), LDL (Chimenti et al., 1990), RDL1 (Kieman, Maindreville & Simon, 1989), DIPS (Sellis, Lin & Raschid, 1989), and Ariel (Hanson, 1989). Currently, most rule systems in these prototypes focus on supporting database services such as triggers, integrity constraints, data security and relational views (e.g., Widom & Finkelstein, 1989 and Ceri & Widom, 1990). In this paper, we focus on processing inference rules that derive data based on existing data. Such data derivation rules and their interaction are likely to be complex in a large ES. Among the derived data, some may be materialized in order to reduce processing costs or query response times.

Materialization of rule-derived data in an EDS is related to materialization of user-defined views in a conventional DBMS, which has been studied previously in (Blakeley, Larson & Tompa, 1986; Hanson, 1987 and Segev & Park, 1989). Our research is different from these previous studies in two major aspects. First, in an EDS, inference rules are usually chained to one another so that decisions on whether or not to materialize a derived relation must be made by considering the chaining effect among rules. Previous work on view materialization (and also work on materializing rule data, such as in (Sellis, Lin & Raschid, 1989 and Segev & Zhao, 1991a,b) did not consider this chaining effect. Second, we introduce auxiliary data constructs that give rise to new opportunities in improving system performance.

This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

The rest of the paper is organized as follows. Section 2 illustrates the relationship between inference rules and relations through a hypothetical application. Section 3 analyzes the characteristics of data propagation through inference rules. Section 4 defines the optimization problem of materializing derived data, presents the rule materialization structures, and develops a decomposition-based algorithm. Section 5 gives an example of the algorithm and presents some computational results, and Section 6 concludes the paper with a summary and future research.

2. Inference Rules

Much of the collected data can not be used directly as decision making variables. Therefore, these data must be preprocessed before they can be used in a decision model. This preprocessing of data can be done by means of inference rules. An inference rule is an If-Then statement that applies constraints to some existing data and derives new data or trigger an action when the If-side is satisfied. The If-side of a rule is called *rule antecedent*, or *rule body*, and the Then-side is called *rule consequent*, or *rule head*. In general, we distinguish between *data rules* and *action rules*; the former are rules that directly affect existing data or derive new data and the latter are rules that activate DBMS commands or procedures such as ABORT a transaction or PRINT a report.

A hypothetical EDS application is illustrated next through two sets of rules and four relations by extending an example in (Turban, 1989). Suppose a commercial bank uses an expert database system to support decision making in the evaluation of financial performance and loan approval. The relevant relation schemas are shown in Figure 2.1, where COMPANY contains the financial statements of loan applicants, LOAN contains the information about the types of loans, and PERFORM and APPROVAL contain the data derived from COMPANY and LOAN through the inference rules described next.

COMPANY(Name,CFTD,TDTA,Type)
LOAN(Type,Term,Size,RiskFactor)
PERFORM(Name,BYear,Solvent,Type)
APPROVAL(Name,Type,BYear,Solvent,Appr)

Appr	decision about loan approval
BYear	projected number of years to bankruptcy
CFTD	the ratio of cash flow to total debt
Name	company name
RiskFactor	measure of risk level of a loan type
Size	loan amount
Solvent	predicted financial status
Term	loan period
Type	loan type
TDTA	the ratio of total debt to total assets

Figure 2.1. The relational schemas.

The first set of rules (Bankruptcy rules) predicts financial performance based on the given data in the COMPANY relation, deriving the PERFORM relation. The second set of rules (Approval rules) approves or disapproves the loan application of a company based on its PERFORM data and the type of loan for which it applies. The resulting data of the rule is placed in another derived relation, the APPROVAL relation. Figure 2.2 illustrates the derivation of the data objects with the two sets of rules. The meaning of each rule is as follows.

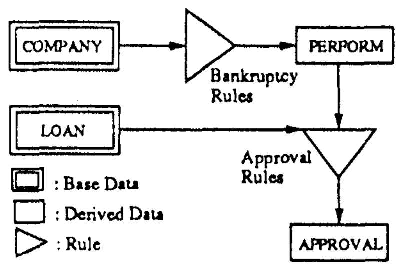


Figure 2.2. Data derivation through rules

Bankruptcy Rules

Rule 1: PERFORM(Name, BYear="5", Solvent="n", Type):-
COMPANY(Name, CFTD>.1309, TDTA>.6975, Type).

Rule 1 says that if the ratio of cash flow to total debt is greater than .1309 and the ratio of total debt to total assets is greater than .6975, then the company could go bankrupt within 5 years.

Rule 2: PERFORM(Name, _, Solvent="y", Type):-
COMPANY(Name, CFTD>.1309, TDTA≤.6975, Type).

Rule 2 says that if the ratio of cash flow to total debt is greater than .1309 and the ratio of total debt to total assets is less than or equal to .6975, then the company is financially solvent.

APPROVAL rules

Rule 3: APPROVAL(Name, Type, BYear, Solvent, Appr="y"):-
LOAN(Type, Term<5, Size<50,000, _),
PERFORM(Name, BYear, Solvent="y", Type).

If the company is solvent, the loan term is less than five years and the loan size is less than 50,000, then Rule 3 recommends to approve the loan.

Rule 4: APPROVAL(Name, Type, BYear, Solvent, Appr="?"):-
LOAN(Type, Term<BYear, Size<20,000, _),
PERFORM(Name, BYear>3, Solvent="n", Type).

If the company could be bankrupt within more than 3 years, the loan term is less than the bankruptcy horizon, and the loan size is smaller than 20,000, then Rule 4 makes uncertain recommendation.

Rule 5: APPROVAL(Name, Type, BYear, Solvent, Appr="n"):-
LOAN(Type, Term≥BYear, _, _),
PERFORM(Name, BYear, Solvent="n", Type).

If the company could be bankrupt within less than the number of years that the loan spans, then Rule 5 disapproves the loan.

The Bankruptcy and Approval rules derive data objects as consequents, and the derived relations PERFORM and APPROVAL may be kept materialized as a way to support data propagation in the rule system. Notice that materializing rule data, such as PERFORM in Figure 2.2 serves two objectives. First, user queries to the rule data itself will realize better performance, and, second, subsequent rules on the chain (the Approval rules in this example) will be processed more efficiently. The chaining effect of such rules is a major focus of our work, and is elaborated in the next section.

3. Characteristics of Data Propagation

Next, we investigate the characteristics of data propagation in an EDS using the example in Figure 3.1. In the figure, double-lined boxes denote base relations, single-lined boxes illustrate derived relations, and triangles indicate rules that derive data from antecedent relations into consequent relations. Also shown in the figure are the transactions to the base relations and user queries to some of the derived relations. We ignore user queries to base relations since their optimization is independent of the materialization decisions. The figure also illustrates the possibility of storing derived data explicitly with base data in the same relation (This scheme will be described in Section 4.2.1).

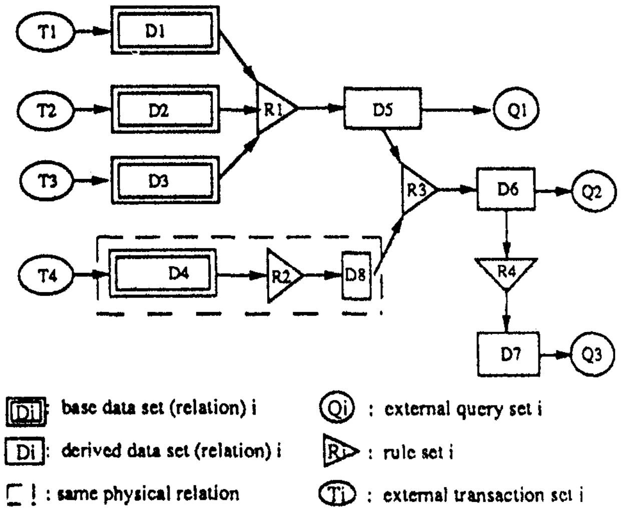

T1
T2
T3
T4
D1
D2
D3
D4
D5
D6
D7
D8
R1
R2
R3
R4
Q1
Q2
Q3
Di : base data set (relation) i
Di : derived data set (relation) i
: same physical relation
Qi : external query set i
Ri : rule set i
Ti : external transaction set i

Figure 3.1. A hypothetical rule data graph

The analysis in this paper is done by constructing a graph $G(D, R, T^e, Q^e)$, where

G denotes the graph.

D denotes the data sets. In Figure 3.1, D = {D1, D2, D3, D4, D5, D6, D7, D8} where Di, i=1 to 4 are base relations, while the others are derived relations.

R denotes the rule sets in the graph, namely, {R1, R2, R3, R4}. Of those rules, R1 and R3 are select-project-join rules, whereas the other rules are select-project rules. The rule sets define the edges that connect the data sets D. The direction of arrows of the rules define the propagation direction of transactions.

T^e denotes the external transactions arriving at the base relations, {T1, T2, T3, T4}. Each transaction set Ti is defined by its arrival rate, its target relation, and the mean number of tuples it updates†. Notice that in Figure 3.1, only external transactions are shown. We use the superscript e to distinguish external transactions from internal ones. Internal transactions result from the propagation of external transactions to update materialized derived data, e.g., if D7 in Figure 3.1 was materialized, internal transactions for its update would be generated for any of the external transactions shown.

Q^e denotes the external queries arriving at the derived relations, namely {Q1, Q2, Q3}. An external query set Qi is defined by its arrival rate, its target relation, and its query selectivity. As for the case of external transactions, external queries may generate internal queries. For example, if none of the derived relations in Figure 3.1 is materialized, Q3 will generate internal queries propagated all the way back to the base relations. Note that internal queries may also be generated by update transactions. To illustrate this, suppose that D6 is the only materialized derived relation in Figure 3.1. In this case, transactions T4 will propagate internal transactions to D6, but since R3 is a join rule, data from D5 is needed, thus propagating internal queries back to R1.

As a result of propagation of transactions and queries, data are derived and rederived within the rule data system, and we call this process the *propagation of data*. In principle, a derived relation should be materialized if the cost of maintaining and querying the materialization is less than the cost of recomputing the relation for all its queries. However, application of this principle to specific problems is complicated by what we call the *chaining effect* of materialization decisions.

We show that the decision on materializing a derived relation is affected by decisions about its consequent relations. Consider the relation D5 in Figure 3.1. The total number of queries at D5 include external queries Q1 and internal queries generated by queries or by transactions at the relation D6. The number of internal queries at D5 depends on whether D6 is materialized (the number of queries to D6, in turn, is dependent on D7). Internal queries at D5 will be generated by queries arriving at D6 if D6 is not materialized; on the other hand, internal queries at D5 will be generated by transactions at D6 if D6 is materialized.

The decision on materializing a derived relation is also affected by decisions on its antecedent relations because the cost of maintaining and querying the given relation depends on whether its antecedent relations are materialized. For example, maintaining and querying D6 may require recomputing D5 if D5 is not material-

† We use the generic term 'updates' to refer to insertions, deletions, and modifications.

ized; this recomputation is unnecessary if D5 is materialized.

Due to this chaining effect, the characteristics of internal transactions and queries to a derived relation are not known before the problem is solved. This dependency of parameters on the solution increases the complexity of the optimization problem significantly.

4. An Optimization Algorithm

In this paper, we assume an immediate update policy. That is, when a derived relation is materialized, the relation will be updated immediately after the completion of transactions at the antecedent relations. For a discussion of other policies, see (Segev & Fang, 1990). When a derived relation is not materialized, the relation will be computed at query times. This section defines the problem and the procedure of selecting derived relations for materialization. Our objective is to minimize the overall processing cost subject to query response time constraints.

The optimization algorithm that will be presented in this section is based on the decomposition idea. It first identifies points in the rule data graph, where local decisions are optimal. The result of those decisions is a set of derived relations whose materialization decisions has been made and a set of optimization problems corresponding to subgraphs of the original graph. In this paper, we assume that the decomposed components can be solved optimally by exhaustive search; if this is not the case, heuristic procedures can be applied to them. We separate the presentation of the optimization into two parts. The first part deals with high level cost expressions, and is intended to convey the overall logic of the optimization procedure. The details of particular cost expressions are dependent on numerous factors such as the materialization structure, update methods, data structures, join methods, etc. In the second part of the presentation, we will make specific assumptions about the details which are required for the computational results, but it should be kept in mind that the objective of this paper is not to optimize every lower level decision (existing query optimizers, for example, can be used to evaluate the cost of different join alternatives and choose the best).

The notation summary below is for reference purpose; it is explained (if necessary) where introduced first. Notice that for the cost symbols below, lower-case c is used to indicate unit costs, while upper-case C factors in the frequency of operations.

---

c_{io} = cost per block access

$c_s(x)$ = cost of screening a tuple of relation x

$c_r(x)$ = cost of generating a tuple of relation x

$C_Q(x)$ = total cost due to queries at relation x

$C_T(x)$ = total cost due to transactions at relation x

$C_g^q(x{=}0)$ = total cost of generating relation x for internal and external queries to x, when the antecedent relations are available, and $\Phi(x) = 0$

$C_q^q(x{=}0)$ = total cost of computing the antecedent relations of x for external and internal queries to x, when $\Phi(x) = 0$

$C_q^t(x{=}1)$ = total cost of computing the antecedent relations of x due to transactions at x, when $\Phi(x) = 1$

$C_r^q(x{=}0)$ = total cost of reading relation x (after generating it) to process external and internal queries at x, when $\Phi(x) = 0$

$C_r^q(x{=}1)$ = total cost of reading relation x to process external and internal queries at x, when $\Phi(x) = 1$

$C_u^t(x{=}1)$ = total cost of updating relation x due to transactions at relation x, when $\Phi(x) = 1$

D = the set of base and derived relations in graph G

f_x = ratio of transactions that are non-null during propagation

g_x = rule selectivity factor to tuples of relation x

G = the rule data graph

N_x = number of tuples in relation x

v_x = average number of triggers a tuple invokes

P_x^i = the set containing the ith generation of preceding relations of relation x

q_x = query selectivity factor at relation x

Q^e = the set of external queries in graph G

$Q(x)$ = the set of external and internal queries at relation x

R = the set of rules in graph G

S_x^i = the set containing the ith-level subsequent relations of relation x

$t_g(x)$ = the time it takes to recompute relation x

$t_c(x)$ = the response time constraint on recomputing relation x

T^e = the set of external transactions in graph G

$T(x)$ = the set of transactions at relation x

$Y(\bullet)$ = the Yao function to determine the number of block accesses

$\Phi(x)$ = the decision variable for materialization, $\Phi(x) = 1$ if x is materialized, and 0 otherwise.

λ_x = the arrival rate of transactions at relation x

α_x = the query rate at relation x including all external and internal queries.

α_x^e = the arrival rate of external queries at relation x

$\alpha_t(x,z)$ = the query rate from x to antecedent z initiated by transactions at relation x due to antecedents other than z.

β_x = the mean number of tuples per transaction at relation x

ω_x = the amplification factor, that is, the average number of new JP patterns a tuple of relation x generates.

4.1. A Decomposition-Based Algorithm

Let $\Phi(x)$ be 1 if relation x is materialized and 0 otherwise, $C_T(x)$ be the total cost incurred by processing transactions to relation x, and $C_Q(x)$ be the total cost of processing queries against relation x. The resulting optimization problem is given as follows.

Problem P:

Given a graph $G(D,R,T^e,Q^e)$ and
the requirements on query response time
select $\Phi(x)$ for all $x \in$ D such that
$\sum_x C_T(x) + \sum_x C_Q(x)$ is minimized

we denote the optimal values of $\Phi(x)$ by $\Phi^*(x)$.

The decomposition of the rule data graph is based on a set of lemmas and theorems. We first introduce the concepts of Response Time Decisions, Local Decisions, and Decision Boundary.

Response time decision: Let $t_g(x)$ be the time to generate a non-materialized relation x, and $t_c(x)$ be a constraint on that time. $t_c(x)$ is the query response time constraint less the time to answer the query if x is materialized. If $t_c(x) < 0$, there is no feasible solution to Problem P, so we assume that all $t_c(x) \geq 0$. The optimization algorithm below makes decisions at several steps on whether or not derived relations must be materialized to satisfy response time constraints.

Local decisions: These decisions are made on $\Phi(x)$, assuming that $\Phi(y)$ and $\Phi(z)$ are known, where y and z are subsequent and preceding relations of x respectively. The following notation is used. S_x^i is the set of ith-level subsequent relations of x. In Figure 3.1, if x = D5, then $S_{D5}^1 = \{D6\}$ and $S_{D5}^2 = \{D7\}$. P_x^i is the set of ith-level preceding relations, e.g., $P_{D6}^1 = \{D5, D8\}$ and $P_{D6}^2 = \{D1, D2, D3, D4\}$. Given assumptions on the values of $\Phi(y)$, $y \in \bigcup_i S_x^i$, and of $\Phi(z)$, $z \in \bigcup_i P_x^i$, the value of $\Phi(x)$ can be determined by a cost benefit analysis. The resulting value of the local decision $\Phi(x)$ may give useful insights leading to a decision on the value of $\Phi^*(x)$.

The value of $\Phi(x)$ is determined by comparing the processing cost when $\Phi(x)$ is 1 and the cost when $\Phi(x)$ is 0. The processing cost when $\Phi(x) = 1$ includes the cost of computing the nonmaterialized antecedent relations of x, the cost of updating relation x, and the cost of reading relation x for the queries reaching relation x.

$$C_1(x) = C_q^i(x{=}1) + C_u^i(x{=}1) + C_r^q(x{=}1)$$

and the processing cost when $\Phi(x) = 0$ includes the cost of computing the nonmaterialized antecedent relations of x, the cost of computing relation x, and the cost of reading relation x for the queries reaching relation x.

$$C_0(x) = C_q^q(x{=}0) + C_g^q(x{=}0) + C_r^q(x{=}0)$$

The difference between $C_1(x)$ and $C_0(x)$ is

$$\Delta C(x) = C_q^i(x{=}1) + C_u^i(x{=}1) + C_r^q(x{=}1) - C_q^q(x{=}0) - C_g^q(x{=}0) - C_r^q(x{=}0)$$

Consequently, $\Phi(x) = 1$ if $\Delta C(x) \leq 0$ and $\Phi(x) = 0$ otherwise. (The details of the above cost components are given in Section 4.2).

Decision boundary: Given a rule data graph, those derived relations whose antecedent relations are base relations form the initial boundary of decisions $\Phi(x)$ to be made, and therefore, we call such relations *boundary relations*. x becomes a new boundary relation when $\Phi(z)$ is determined for all $z \in P_x^1$.

Next, we present theorems and corollaries, leading to the decomposition-based algorithm. For lack of space we omit the proofs; they can be found in (Segev & Zhao, 1991c).

Let $\alpha_i(x,z)$, $z \in P_x^1$ be the query rate to z initiated at x by transactions propagated to x from $z' \in P_x^1$, $z' \neq z$. Note that this can occur only if the rule leading to x is a join rule, and thus $\alpha_i(x,z) = 0$ if $|P_x^1| = 1$. Also let α_x be the total query rate at x (both internal and external); Lemmas 2 and 3 in the Appendix specify that $\alpha_x = \min(\alpha_x)$ when $\Phi(y) = 1$, for all $y \in S_x^1$, and $\alpha_x = \max(\alpha_x)$ when $\Phi(y) = 0$ for all $y \in \bigcup_i S_x^i$. Then we have:

Theorem 1. Assume that (a) $\Phi(y) = 1$, for all $y \in S_x^1$, and (b) $\Phi(z) = 1$, for all $z \in P_x^1$ if $\alpha_i(x,z) < \min(\alpha_x)$ then $\Phi(x) = 1$ for all $z \in P_x^1$; else, if $\alpha_i(x,z) \geq \min(\alpha_x)$, then $\Phi(z) = 0$ for all $z \in \bigcup_i P_x^i$. If a local decision under these assumptions gives $\Phi(x) = 1$ $\Phi^*(x)$ must be 1.

Corollary 1. When considering a derived relation at the decision boundary, $\Phi^*(x)$ must be 1 if $\Phi(x) = 1$ when assuming $\Phi(y) = 1$, for all $y \in S_x^1$.

Theorem 2. Assume that (a) $\Phi(y) = 0$, for all $y \in \bigcup_i S_x^i$, and (b) $\Phi(z) = 1$, for all $z \in P_x^1$ if $\alpha_i(x,z) \geq \max(\alpha_x)$ then $\Phi(x) = 1$ for all $z \in P_x^1$; else, if $\alpha_i(x,z) < \max(\alpha_x)$, then $\Phi(z) = 0$ for all $z \in \bigcup_i P_x^i$. If a local decision under these assumptions gives $\Phi(x) = 0$ $\Phi^*(x)$ must be 0.

Corollary 2. When considering a derived relation at the decision boundary, $\Phi^*(x)$ must be 0 if $\Phi(x) = 0$, when assuming $\Phi(y) = 0$, $y \in \bigcup_i S_x^i$.

Theorem 3. Given a graph $G(D,R,T^e,Q^e)$ and $\Phi(x) = 1$ for $x \in X$, X is a subset of D.

If removing some of the relations x, for any $x \in X$ will split the graph G into two graphs, $G1$ and $G2$, then graph G can be separated into $Gi(Di,Ri,Ti^e,Qi^e)$ for i = 1 to 2, where $D1 = (\bigcup_i P_X^i) \cup X$, $D2 = X \cup (\bigcup_i S_X^i)$, $R = R1 \cup R2$, $Q^e = Q1^e \cup Q2^e$, and $T^e = T1^e \cup T2^e - T(X)$. Solving G1 and G2 separately is equivalent to solving G.

A decomposition-based algorithm:

0. Initialize $\Phi(x) = 0$, for all $x \in D$.
1. Determine $T(x)$ by propagating the external transactions T^e at the base relations, and compute $\lambda_x = \sum_{y \in P_x^1} f_y \lambda_y$ and $\beta_x = \sum_{y \in P_x^1} f_y \lambda_y \beta_y g_y v_y / \lambda_x$.
2. Response time analysis (one-step): Do the following for all derived relations $x \in D$.
   2.1.Compute $t_g(x)$ by assuming $\Phi(y) = 1$, $y \in P_x^1$.
   2.2.Set $\Phi^*(x)$ to 1 if $t_g(x) > t_c(x)$ and 0 otherwise.
   2.3.Insert x into S_M, the set of derived relations to be materialized.
3. Partition the rule data graph into smaller segments if $\Phi^*(x) = 1$ for some x (by Theorem 3).
4. For each graph segment, compute $\min(\alpha_x)$ and $\max(\alpha_x)$ for all $x \in D$, that is, propagate the external queries and the internal queries initiated by transac-

tions in Step 1.

5. Cost and benefit analysis
   5.1. Local decisions:
      5.1.1. For each boundary relation $x \in D$, determine $\Phi(x)$ using Corallary 1. If $\Phi(x) = 1$, then set $\Phi^*(x)$ to 1 and insert x into S_M.
      5.1.2. For each nonboundary relation $x \in D$, determine $\Phi(x)$ using Theorem 1. If $\Phi(x) = 1$, then set $\Phi^*(x)$ to 1 and insert x into S_M.
      5.1.3. Repartition the segmented graphs (by Theorem 3) and go to Step 4 for all new segments if any.
      5.1.4. For each boundary relation $x \in D$, determine $\Phi(x)$ using Corollary 2. If $\Phi(x) = 0$, then set $\Phi^*(x)$ to 0 and insert x into S_N, the set of derived relations not to be materialized.
      5.1.5. For each nonboundary relation $x \in D$, determine $\Phi(x)$ using Theorem 2. If $\Phi(x) = 0$, then set $\Phi^*(x)$ to 0 and insert x into S_N.
   5.2. Exhaustive search:
      5.2.1. For each segment of Problem P, define Problem PS. using the results of Steps 2 and 5 as constraints.

      PS: Given a graph $G(DS, RS, TS^*, QS^*)$
      s.t. $\Phi(x) = 1, x \in DS \cap S_M$
      $\Phi(x) = 0, x \in DS \cap S_N$
      and $t_g(x) \le t_c(x)$
      select $\Phi(x)$ for all $x \in$ DS such that $\sum_x C_T(x) + \sum_x C_Q(x)$ is minimized.

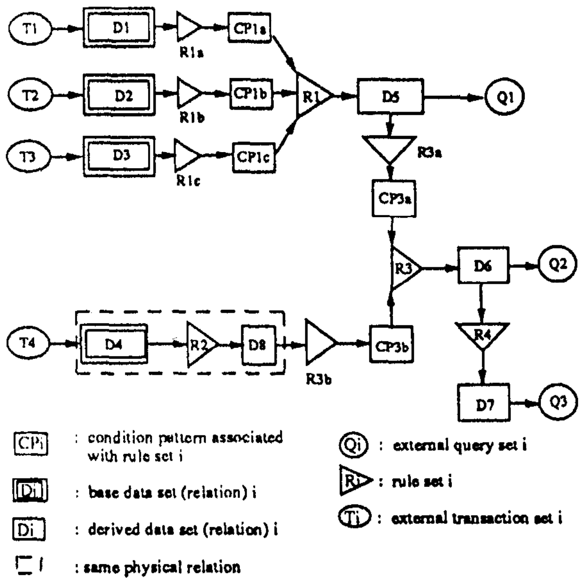
      5.2.2. Test for response time: For each set of values of $\Phi(x)$, $x \in DS$, compute $t_g(x)$ for all x that $\Phi(x) = 0$, and discard the set if $t_g(x) > t_c(x)$ for any $x \in DS$.
      5.2.3. compute the values of $C_T(x)$ and $C_Q(x)$ for the sets passing the test of response time in Step 5.2.2.
      5.2.4. A set of $\Phi(x)$ values that minimize the value of $\left[\sum_x C_T(x) + \sum_x C_Q(x)\right]$ is a solution to Problem PS.
6. The union of the solutions to all segments PS is the solution to Problem P.

4.2. Cost Analysis

In (Sellis, Lin & Raschid, 1989), a data construct, called condition relation, was devised to process join rules. That structure requires propagating matching patterns to multiple relations when the dimension of joins is larger than two. We have developed an alternative method based on auxiliary data constructs: condition pattern relations and join pattern relations (Segev & Zhao, 1991a). Performance evaluation in that study showed that those auxiliary data constructs are very effective in many cases, and they will be considered in this paper to assist processing of join rules.

4.2.1. Materialization Structures

In the following subsections, we develop cost functions for analyses of response time and cost and benefit. We classify rules into select-project rules (SP-rules) and select-project-join rules (SPJ-rules). In the case of SPJ-rules, the condition pattern relations (CP-relations) and the join pattern relation (JP-relation) will be utilized to assist deriving the rule-defined data. In order to do this in a homogeneous way, we logically decompose a SPJ-rule into a set of SP-rules associated with the CP-relations and a join-rule (J-rule) associated with the derived data. As for the JP-relation, we assume that it is a fixture of materializing the derived data. That is, a JP-relation will be used whenever a relation derived by a J-rule is materialized. Figure 3.1 has been expanded as shown in Figure 4.1 to include explicitly the CP-relations, which can be viewed as any derived relation for materialization decisions.



T1
D1
R1a
CP1a
T2
D2
R1b
CP1b
R1
D5
Q1
T3
D3
R1c
CP1c
R3a
CP3a
R3
D6
Q2
T4
D4
R2
D8
R3b
CP3b
R4
D7
Q3
CPi : condition pattern associated with rule set i
Di : base data set (relation) i
Di : derived data set (relation) i
: same physical relation
Qi : external query set i
Ri : rule set i
Ti : external transaction set i

Figure 4.1. The extended rule data graph

In deriving the cost functions in Section 4.2.2, we assume that memory size is large enough to allow computing joins of two relations by retrieving each relation only once. The derived relation will be written to disk at the end of each join computation.

We assume that execution of subsequent transactions are initiated when their immediate preceding transactions are completed. In other words, we do not consider pipelined operation during propagation of transactions, when estimating cost of transactions and queries. We also assume that the characteristics of base relations, rules, external transactions and queries are given.

Data derived by rules may be materialized in several structures according to the rule characteristics. We present three materialization structures as follows.

Case A: Rule consequent data should be materialized in the same relation as the rule antecedent data when the rule body is composed of a single predicate and the selectivity of the rule is large.

Case B: Rule consequent data should be materialized in a separate relation when the rule body is composed of a single predicate and the selectivity of the rule is small.

Case C: When the rule body is composed of more than one predicate, a join operation is usually needed in order to evaluate the rule constraints for a given set of data elements.

We categorize the rule materialization structures into Structure A, B, and C as shown in Figure 4.2 corresponding to Cases A, B, and C respectively.

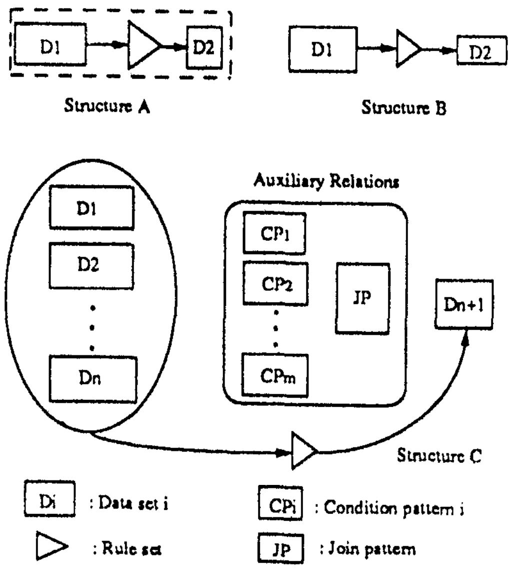


Figure 4.2. Materialization Structures

4.2.2. Derivation of Cost Expressions

Given the materialization and auxiliary structures, we now elaborate on the cost expressions used by the optimization algorithm.

Response Time Analysis

The query response time $t_g(x)$ is defined iteratively next, where $t_g^1(x)$ is the time needed to recompute x when all antecedent relations are readily available for reading, which may be base relations, or materialized derived relations, or nonmaterialized recomputed relations.

In deriving the cost functions next, we assume that base relations and materialized derived relations are indexed on the constrained attributes using indirect hashing and therefore, the cost of reading indexed relations is computed using the approximation of the Yao function (Bernstein, 1981). We also assume that reading a nonmaterialized relation requires a recomputation followed by a complete scan of the computed relation.

The cost of recomputing relation x is the sum of the cost of recomputing relation x from the antecedent relations and the cost of recomputing the antecedent relations if necessary, $t_g(x) = t_g^1(x) + \sum_{y \in P_x^1, \Phi(y)=0} t_g(y)$.

The cost of recomputing relation x from its antecedent relations is separated into the cost of reading and screening the nonmaterialized relations, the cost of reading the materialized relations using indices on the constrained attributes, the cost of generating the tuples of relation x, and the cost of writing the results to disk

$$t_g^1(x) = \sum_{y \in P_x^1, \Phi(y)=0} (c_{io} b_y + c_s(y) N_y)$$

$$+ 2c_{io} \sum_{y \in P_x^1, \Phi(y)=1} Y(b_y, [N_y g_y]) + c_r(x) N_x + c_{io} b_x$$

Where P_x^1 is the antecedent relations of x, b_a and N_a are the number of blocks and tuples of the antecedent relation of x. Notice that the cost of screening tuples and the cost of creating tuples of the derived relation are dependent on the particular relation due to the difference in the number of constrainted attributes, the tuple size and the join dimension.

Cost and Benefit Analysis

In estimating the costs of materializing derived data, we assume an immediate update policy, i.e., the derived data is maintained immediately after a transaction is done at the antecedent data. We assume also that join attributes on all materialized relations are indexed with an indirect hashing method so that a read requires two accesses and a write requires three.

Local decisions

Shown below are the cost functions for the local decisions as defined in Section 4.1. We assume the cost of recomputing a nonmaterialized relation for arriving queries is linear with the time of computing it and the frequency of the relevant queries and transactions.

The cost of computing the antecedent relations of x for materializing x at the transaction times is equal to the sum of the cost of computing the nonmaterialized antecedent relations and the cost of propagating the transactions to the antecedent relations.

$$C_q^l(x=1) = \sum_{y \in P_x^1, \Phi(y)=0} \left[(\lambda_x - f_y \lambda_y) t_g(y) + \lambda_y C_p(y) \right]$$

Where, $\lambda_x = \sum_{y \in P_x^1} f_y \lambda_y$ and f_y is the fraction of non-null transactions† out of all transactions arriving at relation y, after screening against the rule conditions. The cost

† Since transactions are propagated and their tuples are screened, it is possible that the transaction becomes null at a node because none of its tuples passes the screening.

of propagating transactions to an antecedent relation y, $C_p(y)$, is defined iteratively as the cost of propagating transactions from the antecedent relations of y to y plus the cost of propagating transactions to the antecedent relations of y, $C_p(y) = C_p^1(y) + \sum_{z \in P_y^1, \Phi(z)=0} \lambda_z C_p(z)$.

The cost of propagating transactions from the antecedent relations of y to y consists of the cost of reading the nonmaterialized partner relations of the join and screening tuples read, the cost of reading materialized partner relations, and the cost of generating the tuples for the transactions.

$$C_p^1(y) = \sum_{z \in P_y^1, \Phi(z)=0} (c_{io}(\lambda_y - f_z \lambda_z) b_z + c_s(z) f_z \lambda_z \beta_z)$$
$$+ 2c_{io}\left[(\lambda_y - f_z \lambda_z) \sum_{y \in P_x^1, \Phi(y)=1} Y(b_z, [\beta_z g_z])\right] + c_r(y) \lambda_y \beta_y$$

The cost of computing the antecedent relations of x for external and internal queries at relation x is equal to the sum of the cost of computing the nonmaterialized antecedent relations, $C_q^q(x=0) = \alpha_x \sum_{y \in P_x^1, \Phi(y)=0} t_g(y)$.

The cost of computing relation x for external and internal queries at relation x is the product of the query rate and the cost of computing relation x given the antecedent relations, $C_g^q(x=0) = \alpha_x t_g^1(x)$.

The cost of updating relation x at the transaction times is equal to the sum of the cost of reading the nonmaterialized antecedent relations and the cost of reading the materialized antecedent relations, the cost of maintaining the join pattern relation and relation x for J-rules, the cost of maintaining relation x for SP-rules, and the cost of generating the tuples of relation x.

$$C_u^t(x=1) = \sum_{y \in P_x^1, \Phi(y)=0} (c_{io}(\lambda_x - f_y \lambda_y) b_y + c_s(y) \lambda_y \beta_y)$$
$$+2c_{io} \sum_{y \in P_x^1, \Phi(y)=1} (\lambda_x - f_y \lambda_y) Y(b_y, [N_y g_y])$$
$$+3c_{io} I(|P_x^1|>1) \sum_{y \in P_x^1} \lambda_y \left[Y(b_{JP}, [\beta_y g_y \omega_y]) + Y(b_x, [\beta_y g_y \omega_y p_y])\right]$$
$$+3c_{io} I(|P_x^1|=1) \lambda_a Y(b_x, [\beta_a g_a]) + c_r(x) \lambda_x \beta_x$$

Where $I(\bullet)$ is an indicator function, and it is equal to 1 if true and 0 otherwise.

The cost of reading relation x after x is generated when $\Phi(x) = 0$ is $C_r^q(x=0) = c_{io} \alpha_x\, b_x$.

The cost of reading relation x when $\Phi(x) = 1$ is We assume that materialized relations are indexed on their constrained attributes and that the query optimizer is smart enough to decide whether or not to used the index, $C_r^q(x=1) = c_{io} \alpha_x \min\left[2Y(b_x, [N_x q_x]), b_x\right]$.

Exhaustive Search Decisions

For each derived relation x, the cost of processing the transactions and queries can be computed using the formulas shown next. Notice that the query cost in the formulas involve only the cost of processing the external queries because the costs of internal queries are included in the costs of processing transactions and external queries.

Case $\Phi(x) = 1$:

In the case where a derived relation is materialized, the transaction cost is the sum of the cost of computing the antecedent relations of x and the cost of updating relation x, which have been defined above.

$$C_T(x) = C_q^t(x=1) + C_u^t(x=1)$$

The query cost is simply the cost of reading the materialized relation x.

$$C_Q(x) = C_r^q(x=1) = c_{io} \alpha_x^e \min\left[2Y(b_x, [N_x, q_x]), b_x\right]$$

Case $\Phi(x) = 0$:

In the case where a derived relation is not materialized, the transaction cost is zero because relation x will not be updated in this situation; $C_T(x) = 0$.

The query cost is the sum of the cost of generating the antecedent relations of x, the cost of generating x given the antecedent relations, and the cost of reading x. We assume that nonmaterialized relations are not indexed and that querying a recomputed relation requires a complete scan of the relation.

$$C_Q(x) = C_q^q(x=0) + C_g^q(x=0) + C_r^q(x=0)$$
$$= \alpha_x^e \left[\sum_{y \in P_x^1, \Phi(y)=0} t_g(y) + t_g^1(x) + c_{io} b_x\right]$$

5. Example and Computational Results

In this section, we use the rule data graph in Figure 4.1 as an example to demonstrate the decomposition based algorithm developed in Section 4. We present also some preliminary computational results to illustrate our findings in this paper.

The following parameter values will be used in the example and the computational results presented next unless stated otherwise. These parameters have been selected to cover a wide spectrum of cases; for example, the relational sizes have been selected to range from large to small.

5.1. An Example

Next, we demonstrate the decomposition based algorithm by working through the problem represented by Figure 4.1 using the parameter values in Table 5.1 except letting α_x^e equal 10 (1/min). We go through the algorithm in the following steps.

0. Initialize $\Phi(x) = 0$, for all $x \in$ {D5, D6, D7, D8, CP1a, CP1b, CP1c, CP3a, CP3b}, and $\Phi(x) = 1$, for all $x \in$ {D1, D2, D3, D4}.

x	N_x	B_x	b_x	λ_x	β_x	α_x^s	f_x	g_x	q_x	v_x	w_x
D1	200000	30	1500	100	100	0	.85	.5	1	1	1
D2	10000	100	250	500	50	0	.8	.25	1	1	1
D3	2000	200	100	100	10	0	.9	.05	1	1	1
D4	10000	90	225	50	20	0	1	1	1	1	1
D5	55100	60	830	500	20	3	.75	.5	.9	1	1
D6	23780	70	420	420	7	5	.8	.5	.1	1	1
D7	11900	20	60	340	3	2	1	–	1	1	1
D8	10000	35	90	50	20	0	.9	.5	1	1	1
CP1a	100000	50	1250	10	50	0	1	1	1	.5	.5
CP1b	2500	40	25	400	12	0	1	1	1	2	10
CP1c	100	25	1	90	1	0	1	1	1	1	7
CP3a	27550	100	690	380	10	0	1	1	1	.5	.5
CP3b	5000	20	25	45	10	0	1	1	1	2	1.5

The units for the parameters are: B_x, bytes; b_x, blocks; λ_x, 1/min; and α_x^s, 1/min. The following formulas are used to compute the parameters for derived relations:
$b_x = N_x B_x / S$, where S = 4000 bytes per block,
$\lambda_x = \sum_{y \in P_x^1} f_y \lambda_y$, $\beta_x = \sum_{y \in P_x^1} \lambda_y f_y \beta_y g_y v_x / \lambda_x$, $N_x = \sum_{y \in P_x^1} N_y g_y v_y$.

Other parameters not shown in the table are: $b_{/P_1}$ = 380 blocks and $b_{/P_3}$ = 55 blocks.

1. Propagate the external transactions T^e at the base relations, and the resulting values of λ_x and β_x are shown in the parameter table above.
2. Compute the one-step response time $t_G(x)$ in seconds, assuming all antecedent relations of x are materialized.

x	5	6	7	8	1a	1b	1c	3a	3b
$t_G^1(x)$ (sec)	254	74	39	26	207	18	4	98	11

3. In order to concentrate on the cost benefit analysis, we assume $t_c(x) > 300$, and therefore, no partition is possible.
4. For each graph segment, compute min(α_x) and max(α_x) for all non-base relations $x \in D$, propagating the external and internal queries initiated by transactions.

x	5	6	7	8	1a	1b	1c	3a	3b
min(α_x)	3	5	2	0	490	100	410	370	45
max(α_x)	380	7	2	52	870	480	790	377	52

5. The boundary relations include CP1a, CP1b, CP1c, and D8. By Corallary 1, we assume the consequent relations of these boundary relations are materialized. Analyze CP1a, CP1b, and CP1c using the cost functions in Section 4.2.3, and we get $\Delta C(x) < 0$ for $x \in$ {CP1a, CP1b, CP1c}. Therefore, we have $\Phi^*(x) = 1$ for $x \in$ {CP1a, CP1b, CP1c}. Now, we have a new boundary relation D5. D5 and D8 are analyzed using Corollary 1 and we get $\Delta C(5) > 0$ and $\Delta C(8) > 0$; these results do not lead to the decisions on $\Phi^*(x)$ for x = 5 and 8. The set of relations to be materialized so far is S_M = {CP1a, CP1b, CP1c}.

   Using Theorem 1, the nonboundary relations CP3a and CP3b are analyzed. Since min(α_{3a}) is 370 which is equal to $\alpha_t(3a,5)$, and min(α_{3b}) is 45 that is less than $\alpha_t(3b,8)$, we assume $\Phi(5) = 1$ and $\Phi(8) = 1$ in the local analysis of CP3a and CP3b respectively. The results turn out to be $\Phi(CP3a) = 1$ and $\Phi(CP3b) = 1$. By Theorem 1, we know that $\Phi^*(CP3a) = 1$ and $\Phi^*(CP3b) = 1$. Now S_M includes {CP1a, CP1b, CP1c, CP3a, CP3b}.

   D6 becomes a boundary relation after CP3a and CP3b are materialized, we use Corollary 2 to confirm the hypothesis that D6 is not to be materialized. Using max(α_6) = 7, we obtain $\Delta C(6) > 0$, and therefore, $\Phi(6) = 0$ for the decomposed decision. Consequently, $\Phi^*(6)$ is set to 0. Similar analysis of D8 is done using $\alpha_8 = 0$ since CP3b is materialized, resulting in $\Phi^*(8) = 0$. The set of relations not to be materialized is S_N = {D6, D8}.

   The graph G of Figure 4.1 can now be partitioned into six smaller segments, G1, G2, G3, G4, G5, and G6 that contain relations {D1, CP1a}, {D2, CP1b}, {D3, CP1c}, {CP1a, CP1b, CP1c, D5, CP3a}, {D4, D8, CP3b}, and {CP3a, CP3b, D6, D7} respectively.

   Applying Corollary 2 to relation D5 and Theorem 2 to relation D7 does not give useful insights. Therefore, we do an exhaustive search on graphs G4 and G6 using the cost functions derived in Section 4.3. The resulting decisions are $\Phi^*(5) = 0$ and $\Phi^*(7) = 0$.
6. The final results for this particular example are to materialize relations CP1a, CP1b, CP1c, CP3a, CP3b, and to leave others nonmaterialized.

5.2. Computational Results

In this section we present preliminary computational results using the cost model defined in Section 4 and the parameter values in Table 5.1. The basis for evaluating our algorithm are two straightforward strategies. The first strategy (S1) is no materialization at all. The second strategy (S2) is to materialize all derived data. S3 denotes the strategy of using the decomposition-based algorithm. The computational results demonstrate that by materializing derived data selectively, the cost reduction can be significant.

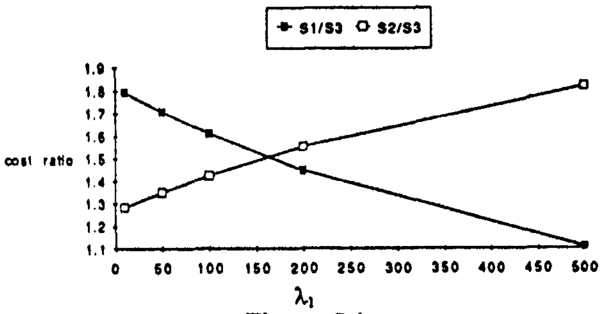


Figure 5.1

Figure 5.1 illustrates how the cost ratios of S1 to S3 and of S2 to S3 vary with the transaction arrival rate to relation D1. By selecting some relations to materialize that minimizes the total processing cost, the system can achieve cost reduction compared to either no materialization (S1) or total materialization (S2). In the range of λ_1 shown in the figure, Strategy S1 is more expensive than S2 when the value of λ_1 is lower while S1 is less costly than S2 when λ_1 becomes higher, that is, materialization is more expensive. In this particular case, the materialized relations in Strategy S3 include CP1a and CP3b, achieving the lowest cost among all three strategies.

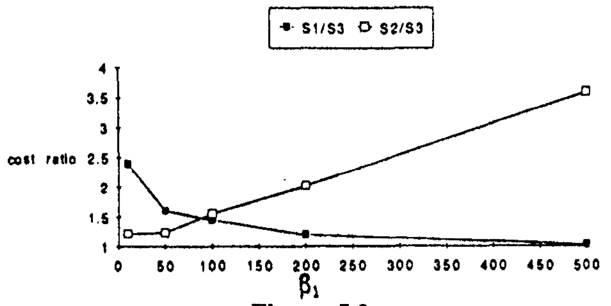


Figure 5.2

Figure 5.2 shows that when the mean number of tuples a transaction accesses at relation D1, β_1, varies, the cost ratios of S1/S3 and S2/S3 approach 1 towards the higher and lower values of β_1 respectively. In this computation, the materialized relations are D5, CP1a, CP1b, CP1c, CP3a, and CP3b When β_1 is at the lower end; these relations reduce to CP1a and CP3b around the medium value of β_1, and they further reduce to CP3b when β_1 reaches the higher end.

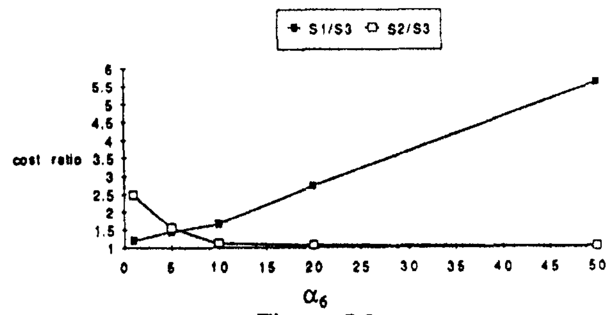


Figure 5.3

The effect of α_6 to the selection of materialized relations is the opposite of the effects of λ_1 and β_1 as shown in Figure 5.3. The derived relations selected for materialization vary from {CP1a, CP3b}, to {CP1a, CP1b, CP1c, CP3a, CP3b}, and to {D5, D6, CP1a, CP1b, CP1c, CP3a, CP3b} as α_6 changes from low to high.

Selectivity of rules is another important factor when selecting derived relations for materialization. As shown in Figure 5.4, the rule selectivity (g_1) affects the cost ratios greatly. When g_1 goes up, the cost of Strategy 2 rises while the cost of Strategy 1 decreases. Our computational result indicates that the relations selected

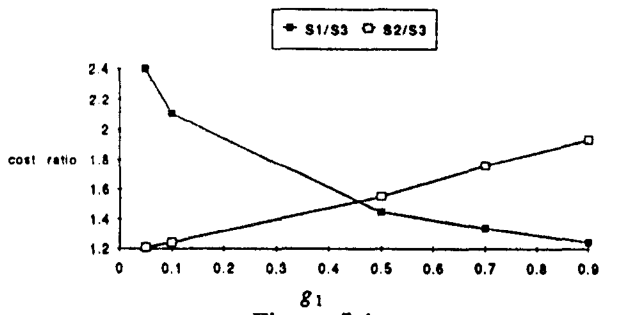


Figure 5.4

for materialization are {CP1a, CP3b}.

In summary, Strategy 3 is always the best among the three strategies (Strategies 1 and 2 are, in some sense, extreme cases of Strategy 3). By selecting appropriate derived data for materialization, the overall processing cost can be reduced to a minimum.

6. Conclusion

To support decision making while using large amounts of data, DBMS and ES technologies are being integrated into *expert database systems* (EDSs) that manage both rules and data in a unified system. When rules are processed, data must be derived and stored efficiently. This process can be complex if the rule derived data affect other rules. In this paper, we have studied the optimization problem of deriving data through inference rules. Our main contributions include the followings.

- We studied the chaining effect when selecting derived data for materialization. The decision on whether or not to materialize a derived relation depends upon cost benefit analysis. Generally speaking, materialization decision about a derived relation which is linked by inference rules to some other relations must be made in conjunction with decisions on all other related relations because of the interactive nature of the chained relations.
- The optimization algorithm presented in this paper is based on the decomposition idea. It first identifies points in the rule data graph, where local decisions are optimal. The result of those decisions is a set of derived relations whose materialization decisions has been made and a set of optimization problems corresponding to subgraphs of the original graph.
- We presented some preliminary computational results based on the decomposition-based algorithm. Results showed that an optimization algorithm is necessary when implementing inference rules in order to reduce the cost of data management for large rule systems. Moreover, selective materialization can reduce the processing costs significantly compared to "all-materialized" or "non-materialized" strategies.

We are interested in exploring the following issues in future research.

- To construct a more elaborate model that can be used to analyze issues such as the effect of buffer management and access methods on the materialization decisions.
- To investigate the effectiveness of the decomposition-based algorithm in various graph topologies, arrival rates of transactions and queries, and other important paramenters.
- To study the sensitivity of materialization decisions to various parameter values. This is an important issue especially in more dynamic environments.
- To explore issues related to the implementation of the selective materialization strategy. For example, the proposal by Hanson (1989) to adopt the Rete network to a relational EDS can easily be applied to the case where all derived relations are materialized, but with selective materialization it needs to be augmented by information about materialization status of specific nodes.

7. Reference

P. A. Bernstein, et al. *Query processing in a system for distributed databases (SDD-1).* ACM Transactions on Database Systems, 6, 4, 1981.

J. A. Blakeley, P. Larson, and F. W. Tompa. *Efficiently updating materialized views.* Proceedings of the ACM-SIGMOD conference on Management of Data, May 1986.

S. Ceri and J. Widom. *Deriving production rules for constraint maintenance.* Research Report, IBM Almaden Research Center, RJ 7348 (68829) 3/1/90.

D. Chimenti et al. *The LDL system prototype.* IEEE Trans. on Knowledge and Data Engineering, Vol 2, No 1, March 1990.

P. Duchessi, H. Shawky, and J. P. Seagle. *A knowledge-engineered system for commercial loan decisions.* Financial Management, Autumn 1988, pp 57-65.

M. M. Extejt and M. P. Lynn. *Expert systems as human resource management decision tools.* Journal of Systems Management, December 1988, pp 10-15.

L. Haas et al. *Starburst mid-flight: as the dust clears.* IEEE Trans. on Knowledge and Data Engineering, Vol 2, No 1, March 1990.

E. R. Hanson. *A performance analysis of view materialization strategies.* Proceedings of the ACM-SIGMOD international conference on Management of Data, May 1987.

E. R. Hanson. *An initial report on the design of Ariel.* SIGMOD RECORD, Vol 18, No. 3, Sep. 1989.

J. Kieman, C. de Maindreville, and E. Simon. *The design and implementation of an extendible deductive database system.* ACM SIGMOD RECORD, Vol. 18, No. 3, September 1989.

H. R. Rao and B. P. Lingarai. *Expert systems in production and operations management: classification and prospects.* Interfaces 18: 6 November-December 1988 pp 80-91.

Timos Sellis, Chih-Chen Lin, and Louiqa Raschid. *Data intensive production systems: The DIPS approach.* SIGMOD RECORD, Vol 18, No. 3, Sep. 1989.

A. Segev and W. Fang. *Currency-Based Updates to Distributed Materialized Views.* IEEE 6th International Conference on Data Engineering, pp. 512-520, 1990.

A. Segev and J. Park. *Updating Distributed Materialized Views.* IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 2, June 1989.

A. Segev and J. L. Zhao. *Rule Management in Expert Database Systems.* Tech Report LBL-30044, Lawrence Berkeley Laboratory, 1991a.

A. Segev and J. L. Zhao. *Evaluation of rule processing strategies in expert databases.* IEEE 7th International Conference on Data Engineering, Kobe, Japan, 1991b.

A. Segev and J. L. Zhao. *Data Management for Large Rule Systems.* Tech Report LBL-30362, Lawrence Berkeley Laboratory, 1991c.

M. J. Shaw and J. A. Gentry. *Using an expert system with inductive learning to evaluate business loans.* Financial Management, Autumn 1988, pp 45-56.

J. K. Shim and J. S. Rice. *Expert systems applications to managerial accounting.* Journal of Systems Management, June 1988, pp 6-13.

V. Srinivasan and Y. H. Kim. *Designing expert financial systems: a case study of corporate credit management.* Financial Management, Autumn 1988, pp 32-44.

Michael Stonebraker, Eric N. Hanson and Spyros Potamianos. *The Postgres Rule Manager.* IEEE Trans. on Software Engineering, Vol. 14, No. 7, July 1988.

E. Turban. *Decision Support and Expert Systems.* Macmillan Publishing Company, New York, 1989.

P. Valduriez. *Join indices.* ACM Transactions on Database Systems, Vol. 12, No. 2, June 1987.

Jennifer Widom and Sheldon J. Finkelstein. *Syntax and semantics for set-oriented production rules in relational database systems.* SIGMOD RECORD, Vol 18, No. 3, Sep. 1989.

K. Wilkinson, P. Lyngbek, and W. Hasan. *The Iris architecture and implementation.* IEEE Trans. on Knowledge and Data Engineering, Vol 2, No 1, March 1990.