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 & Lingaraju, 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 a 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 & Potamnos, 1988), Starburst (Haas et al., 1990), Iris (Wilkinson, Lyngbek & Hasan, 1990), LDL (Chimenti et al., 1990), RDL1 (Kieman, Mairdreville & 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, 1991.a,b) did not consider this chaining effect. Second, we introduce auxiliary data constructs that give rise to new opportunities in improving system performance.
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 triggers 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.

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 2.1. The relational schemas.](image1)

<table>
<thead>
<tr>
<th>COMPANY(Name,CFTD,TDTA,Type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAN(Type,Term,Size,RiskFactor)</td>
</tr>
<tr>
<td>PERFORM(Name,BYear,Solvent,Type)</td>
</tr>
<tr>
<td>APPROVAL(Name,BYear,Solvent,Appr)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appr</td>
<td>decision about loan approval</td>
</tr>
<tr>
<td>BYear</td>
<td>projected number of years to bankruptcy</td>
</tr>
<tr>
<td>CFTD</td>
<td>the ratio of cash flow to total debt</td>
</tr>
<tr>
<td>Name</td>
<td>company name</td>
</tr>
<tr>
<td>RiskFactor</td>
<td>measure of risk level of a loan type</td>
</tr>
<tr>
<td>Size</td>
<td>loan amount</td>
</tr>
<tr>
<td>Solvent</td>
<td>predicted financial status</td>
</tr>
<tr>
<td>Term</td>
<td>loan period</td>
</tr>
<tr>
<td>Type</td>
<td>loan type</td>
</tr>
<tr>
<td>TDTA</td>
<td>the ratio of total debt to total assets</td>
</tr>
</tbody>
</table>

![Figure 2.2. Data derivation through rules Bankruptcy Rules](image2)

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, BYear>Solvent="y", Type):- COMPANY(Name, CFTD>.1309, TDTA≤.8975, 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.

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

Barcelona, September, 1991
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).

\[ G(D, R, T^e, T^i, Q^e, Q^i) \]

\( G \) denotes the graph.
\( D \) denotes the data sets. In Figure 3.1, \( D = \{D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8\} \) where \( D_i, i=1 \) to 4 are base relations, while the others are derived relations.
\( R \) denotes the rule sets in the graph, namely, \( \{R_1, R_2, R_3, R_4\} \). Of those rules, \( R_1 \) and \( R_3 \) 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, \( \{T_1, T_2, T_3, T_4\} \). Each transaction set \( T_i \) 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 \( D_7 \) 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 \( \{Q_1, Q_2, Q_3\} \). An external query set \( Q_i \) 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, \( Q_3 \) 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 \( D_6 \) is the only materialized derived relation in Figure 3.1. In this case, transactions \( T_4 \) will propagate internal transactions to \( D_6 \), but since \( R_3 \) is a join rule, data from \( D_5 \) is needed, thus propagating internal queries back to \( R_1 \).

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 \( D_5 \) in Figure 3.1. The total number of queries at \( D_5 \) include external queries \( Q_1 \) and internal queries generated by queries or by transactions at the relation \( D_6 \). The number of internal queries at \( D_5 \) depends on whether \( D_6 \) is materialized (the number of queries to \( D_6 \), in turn, is dependent on \( D_7 \)). Internal queries at \( D_5 \) will be generated by queries arriving at \( D_6 \) if \( D_6 \) is not materialized; on the other hand, internal queries at \( D_5 \) will be generated by transactions at \( D_6 \) if \( D_6 \) 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 \( D_6 \) may require recomputing \( D_5 \) if \( D_5 \) is not materialized.

\( \dagger \) 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 have 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_w = \text{cost per block access} \\
c_s(x) = \text{cost of screening a tuple of relation } x \\
c_r(x) = \text{cost of generating a tuple of relation } x \\
C_Q(x) = \text{total cost due to queries at relation } x \\
C_T(x) = \text{total cost due to transactions at relation } x \\
C_F(x=0) = \text{total cost of generating relation } x \text{ for internal and external queries to } x, \text{ when the antecedent relations are available, and } \Phi(x) = 0 \\
C_F(x=0) = \text{total cost of computing the antecedent relations of } x \text{ for external and internal queries at } x, \text{ when } \Phi(x) = 0 \\
C_F(x=1) = \text{total cost of computing the antecedent relations of } x \text{ due to transactions at } x, \text{ when } \Phi(x) = 1 \\
C_F(x=0) = \text{total cost of reading relation } x \text{ (after generating it) to process external and internal queries at } x, \text{ when } \Phi(x) = 0 \\
C_F(x=1) = \text{total cost of reading relation } x \text{ to process external and internal queries at } x, \text{ when } \Phi(x) = 1 \\
C_F(x=1) = \text{total cost of updating relation } x \text{ due to transactions at relation } x, \text{ when } \Phi(x) = 1 \\
D = \text{the set of base and derived relations in graph } G \\
f_x = \text{ratio of transactions that are non-null during propagation} \\
g_r = \text{rule selectivity factor to tuples of relation } x \\
G = \text{the rule data graph} \\
N_p = \text{number of tuples in relation } x \\
v_t = \text{average number of triggers a tuple invokes} \\
P_i = \text{the set containing the } i\text{th generation of preceding relations of relation } x \\
\sigma_x = \text{query selectivity factor at relation } x \\
Q^* = \text{the set of external queries in graph } G \\
Q(x) = \text{the set of external and internal queries at relation } x \\
R = \text{the set of rules in graph } G \\
S_i = \text{the set containing the } i\text{th-level subsequent relations of relation } x \\
t_r = \text{the time it takes to recompute relation } x \\
t_r = \text{the response time constraint on recomputing relation } x \\
T = \text{the set of external transactions in graph } G \\
T(x) = \text{the set of transactions at relation } x \\
Y(x) = \text{the Yao function to determine the number of block accesses} \\
\Phi(x) = \text{the decision variable for materialization, } \Phi(x) = 1 \text{ if } x \text{ is materialized, and } 0 \text{ otherwise.} \\
\lambda_x = \text{the arrival rate of transactions at relation } x \\
\alpha_x = \text{the query rate at relation } x \text{ including all external and internal queries.} \\
\alpha_x = \text{the arrival rate of external queries at relation } x \\
\alpha_x(x,z) = \text{the query rate from } x \text{ to antecedent } z \text{ initiated by transactions at relation } x \text{ due to antecedents other than } z. \\
\bar{\beta}_x = \text{the mean number of tuples per transaction at relation } x \\
\omega_x = \text{the amplification factor, that is, the average number of new JP patterns a tuple of relation } x \text{ 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 \( G(D,R,T,Q^*) \) and the requirements on query response time select \( \Phi(x) \) for all \( x \in D \) such that \( \sum_{x \in D} C_T(x) + \sum_{x \in R} C_Q(x) \) 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 \( t_e(x) \) be the time to generate a non-materialized relation \( x \), and \( t_o(x) \) be a constraint on that time. \( t_e(x) \) is the query response time constraint less the time to answer the query if \( x \) is materialized. If \( t_e(x) < 0 \), there is no feasible solution to Problem P, so we assume that all \( t_e(x) \geq 0 \). 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. \( S_i \) is the set of \( i \)-th level subsequent relations of \( x \). In Figure 3.1, if \( x = D_5 \), then \( S_3 = \{D_6\} \) and \( S_4 = \{D_7\} \). \( P_k \) is the set of \( i \)-th level preceding relations, e.g., \( P_4 = \{D_5, D_8\} \) and \( P_3 = \{D_1, D_2, D_3, D_4\} \). Given assumptions on the values of \( \Phi(y) \), \( y \in \bigcup S_i \), and of \( \Phi(z) \), \( z \in \bigcup P_k \), 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 \( \Phi(x) = 1 \) and the cost when \( \Phi(x) = 0 \). The processing cost when \( \Phi(x) = 1 \) includes the cost of computing the nonmaterialized antecedent relations of \( x \), the cost of reading relation \( x \), and the cost of reading relation \( x \) for the queries reaching relation \( x \).

\[
C_i(x) = C_i^r(x=1) + C_i^u(x=1) + C_i^f(x=1)
\]

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 \( x \).

\[
C_0(x) = C_0^r(x=0) + C_0^u(x=0) + C_0^f(x=0)
\]

The difference between \( C_i(x) \) and \( C_0(x) \) is

\[
\Delta C(x) = C_i^r(x=1) + C_i^u(x=1) + C_i^f(x=1) - C_0^f(x=0) - C_0^f(x=0)
\]

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 \( \Phi(x) \) is determined for all \( x \in P_k \).

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 \( \alpha_i(x,z) \), \( z \in P_k \), be the query rate to \( z \) initiated at \( x \) by transactions propagated to \( x \) from \( z \in P_k \), \( z \in X \). Note that this can occur only if the rule leading to \( x \) is a join rule, and thus \( \alpha_i(x,z) = 0 \) if \( P_k = 1 \). Also let \( \alpha_o \) be the total query rate at \( x \) (both internal and external); Lemmas 2 and 3 in the Appendix specify that \( \alpha_o = \min(\alpha_i(z)) \) when \( \Phi(y) = 1 \), for all \( y \in S_i \), and \( \alpha_o = \max(\alpha_i(z)) \) when \( \Phi(y) = 0 \) for all \( y \in S_i \). Then we have:

Theorem 1. Assume that (a) \( \Phi(y) = 1 \), for all \( y \in S_i \), and (b) \( \Phi(z) = 1 \), for all \( z \in P_k \) if \( \alpha_i(x,z) < \min(\alpha_i(z)) \) then \( \Phi(x) = 1 \) for all \( x \in P_k \); else, if \( \alpha_i(x,z) \geq \min(\alpha_i(z)) \) then \( \Phi(x) = 0 \) for all \( x \in P_k \). 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 \( y \in S_i \).

Theorem 2. Assume that (a) \( \Phi(y) = 0 \), for all \( y \in \bigcup S_i \), and (b) \( \Phi(z) = 1 \), for all \( z \in P_k \) if \( \alpha_i(x,z) \geq \max(\alpha_i(z)) \) then \( \Phi(x) = 1 \) for all \( x \in P_k \); else, if \( \alpha_i(x,z) < \max(\alpha_i(z)) \), then \( \Phi(x) = 0 \) for all \( x \in \bigcup P_k \). 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 \( \Phi(y) = 0 \), \( y \in \bigcup S_i \).

Theorem 3. Given a graph \( G(D,R,T',Q') \) 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, \( G_1 \) and \( G_2 \), then graph \( G \) can be separated into \( Gi(D_i,R_i,T'_i,Q'_i) \) for \( i = 1 \) to 2, where \( D_1 = (\bigcup P_k) \cup X \), \( D_2 = X \cup (\bigcup S_i) \), \( R = R_1 \cup R_2 \), \( Q' = Q_1 \cup Q_2 \), and \( T' = T_1 \cup T_2 \cdot T(X) \).

Solving \( G_1 \) and \( G_2 \) separately is equivalent to solving \( G \).

A decomposition-based algorithm:

1. Initialize \( \Phi(x) = 0 \), for all \( x \in D \).
2. Response time analysis (one-step): Do the following for all derived relations \( x \in D \):
   1. Determine \( T(x) \) by propagating the external transactions \( T' \) at the base relations, and compute \( \lambda_x = \sum f_y \lambda_y \) and \( \beta_x = \sum f_y \beta_y \nu_y / \lambda_x \).
   2. Set \( \Phi(x) \) to 1 if \( t_e(x) > t_o(x) \) and 0 otherwise.
   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 \( \min(\alpha_i) \) and \( \max(\alpha_i) \) for all \( x \in D \), that is, propagate the external queries and the internal queries initiated by transac-
5. Cost and benefit analysis

5.1. Local decisions:

5.1.1. For each boundary relation \( x \in D \), determine \( \Phi(x) \) using Corollary 1. If \( \Phi(x) = 1 \), then set \( \Phi'(x) = 1 \) 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 \( \Phi'(x) = 1 \) 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 \( \Phi'(x) = 0 \) 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 \( \Phi'(x) = 0 \) 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.

\[ PS: \text{Given a graph } G(DS, RS, TS, QS) \]
\[ \text{s.t. } \Phi(x) = 1, \ x \in DS \cap S_M \]
\[ \Phi(x) = 0, \ x \in DS \cap S_N \]
\[ \text{and } t_p(x) \leq t_s(x) \]

select \( \Phi(x) \) for all \( x \in DS \) such that \( \sum C_T(x) + \sum C_Q(x) \) is minimized.

5.2.2. Test for response time: For each set of values of \( \Phi(x) \), \( x \in DS \), compute \( t_p(x) \) for all \( x \) that \( \Phi(x) = 0 \), and discard the set if \( t_p(x) > t_s(x) \) 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

\[ \sum C_T(x) + \sum C_Q(x) \]

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 this 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.

![Figure 4.1. The extended rule data graph](image-url)

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.

![Diagram of Materialization Structures](image)

**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 $t_q(x)$ is defined iteratively next, where $t_q(x)$ is the time needed to compute $x$ 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, $t_q(x) = t_q(x) + \sum_{y \in P_x^*:N_y^*>0} t_q(y)$.

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

$$t_q(x) = \sum_{y \in P_x^*:N_y^*>1} (c_{by} + c_{by}(y)N_y + c_{by}b_x) + 2c_{by} \sum_{y \in P_x^*:N_y^*>1} Y(b_y(N_y^*) + c_{by}(x)N_x + c_{by}b_x)$$

Where $P_x^*$ is the antecedent relations of $x$, $b_x$ and $N_x$ 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 constrained 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.

$$C(x=1) = \sum_{y \in P_x^*:N_y^*>1} \left[ (\lambda_x\bar{r}_x\lambda_y) \bar{r}_y(y) + \lambda_y C_y(y) \right]$$

Where, $\lambda_x = \sum_{y \in P_x^*:N_y^*>1} \bar{r}_x\lambda_y$ and $\bar{r}_y$ is the fraction of non-null transactions out of all transactions arriving at relation $y$, after screening against the rule conditions. The cost

\[\text{Since transactions are propagated and their tuples are screened, it is possible that the transaction becomes null at a node because none of its tuples passes the screening.}\]
of propagating transactions to an antecedent relation \(y\), \(C_y(y)\), 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 \(y\), \(C_y(y) = C_y(y) + \sum_{\sigma \in P_y : \Phi(\sigma) > 0} \lambda_y C_x(x)\).

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.

\[
C_y(y) = \sum_{\sigma \in P_y : \Phi(\sigma) = 0} (c_{\sigma}(\lambda_y - f_y, \lambda_y) b_y + c_x(x) f_x, \lambda_x) + c_y(y) \lambda_y b_y + 2c_\sigma \left(\lambda_y - f_y, \lambda_y \right) \sum_{\sigma \in P_y : \Phi(\sigma) = 1} Y(b_x, [\beta_x, g_x]) + c_y(y) \lambda_y b_y
\]

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, \(C_f(x=0) = \alpha_x \sum_{\sigma \in P_x : \Phi(\sigma) = 0} t_f(\sigma)\).

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 \(S\)-rules, and the cost of generating the tuples of relation \(x\).

\[
C_t(x=1) = C_f(x=1) + C_t(x=1) + C_t(x=1) + C_t(x=1) + C_t(x=1)
\]

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

\[
C_t(x=1) = \alpha_x \sum_{\sigma \in P_x : \Phi(\sigma) = 0} t_f(\sigma) + t_f(x) + c_x b_x
\]

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 \(a_x\) 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 \(x \in \{D1, D2, D3, D4\}\).
The units for the parameters are: $B_x$, bytes; $b_x$, blocks; $\lambda_x$, 1/min; and $\alpha_x^*$, 1/min. The following formulas are used to compute the parameters for derived relations: 

$$b_x = N_x S / b_x$$

where $S = 4000$ bytes per block,

$$\lambda_x = \sum_{y \in P_x} \lambda_y f_x \beta_y g_x \gamma_x$$

Other parameters not shown in the table are: $b_{p1} = 380$ blocks and $b_{p2} = 55$ blocks.

1. Propagate the external transactions $T'$ at the base relations, and the resulting values of $\lambda_x$ and $\beta_x$ are shown in the parameter table above.

2. Compute the one-step response time $t(x)$ in seconds, assuming all antecedent relations of $x$ are materialized.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$N_x$</th>
<th>$B_x$</th>
<th>$b_x$</th>
<th>$\lambda_x$</th>
<th>$\alpha_x^*$</th>
<th>$f_x$</th>
<th>$g_x$</th>
<th>$\beta_x$</th>
<th>$\gamma_x$</th>
<th>$w_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>200000</td>
<td>30</td>
<td>1500</td>
<td>100</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>100000</td>
<td>100</td>
<td>250</td>
<td>500</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D3</td>
<td>200000</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D4</td>
<td>100000</td>
<td>90</td>
<td>225</td>
<td>50</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>55100</td>
<td>60</td>
<td>830</td>
<td>500</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D6</td>
<td>23780</td>
<td>70</td>
<td>420</td>
<td>420</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D7</td>
<td>11900</td>
<td>20</td>
<td>60</td>
<td>340</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D8</td>
<td>10000</td>
<td>35</td>
<td>90</td>
<td>50</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CP1a</td>
<td>100000</td>
<td>50</td>
<td>1250</td>
<td>50</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CP1b</td>
<td>2500</td>
<td>40</td>
<td>25</td>
<td>400</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CP1c</td>
<td>100</td>
<td>25</td>
<td>1</td>
<td>90</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CP2a</td>
<td>27550</td>
<td>100</td>
<td>690</td>
<td>380</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CP3b</td>
<td>5000</td>
<td>20</td>
<td>25</td>
<td>45</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3. In order to concentrate on the cost benefit analysis, we assume $t(x) > 300$, and therefore, no partition is possible.

4. For each graph segment, compute min($\alpha_x$) and max($\alpha_x$) for all non-base relations $x \in D$, propagating the external and internal queries initiated by transactions.

<table>
<thead>
<tr>
<th>$x$</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>1a</th>
<th>1b</th>
<th>1c</th>
<th>3a</th>
<th>3b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{\text{avg}} (x)$ (s)</td>
<td>254</td>
<td>74</td>
<td>39</td>
<td>26</td>
<td>207</td>
<td>18</td>
<td>4</td>
<td>98</td>
<td>11</td>
</tr>
</tbody>
</table>

5. The boundary relations include CP1a, CP1b, CP1c, and CP3b. By Corollary 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($\alpha_{CP3a}$) is 370 which is equal to $\alpha_{3a,5}$. and min($\alpha_{CP3b}$) is 45 that is less than $\alpha_{3b,8}$, we assume $\Phi(5) = 1$ and $\Phi(8) = 1$ in the local analysis of CP3a and CP3b respectively. The results turn out to be $\Phi(3a,5) = 1$ and $\Phi(3b,8) = 1$. 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 $\max(\alpha_x) = 7$, 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, D5, 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 \( \lambda \) shown in the figure, Strategy S1 is more expensive than S2 when the value of \( \lambda \) is lower while S1 is less costly than S2 when \( \lambda \) 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 5.1](image1)

**Figure 5.1**

Figure 5.2 shows that when the mean number of tuples a transaction accesses at relation D1, \( \beta_1 \), varies, the cost ratios of S1/S3 and S2/S3 approach 1 towards the higher and lower values of \( \beta_1 \), respectively. In this computation, the materialized relations are D5, CP1a, CP1b, CP1c, CP3a, and CP3b. When \( \beta_1 \) is at the lower end; these relations reduce to CP1a and CP3b around the medium value of \( \beta_1 \), and they further reduce to CP3b when \( \beta_1 \) reaches the higher end.

![Figure 5.2](image2)

**Figure 5.2**

The effect of \( \alpha_6 \) to the selection of materialized relations is the opposite of the effects of \( \lambda_1 \) and \( \beta_1 \) 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 \( \alpha_6 \) changes from low to high.

![Figure 5.3](image3)

**Figure 5.3**

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 have 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 parameters.

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


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


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


Proceedings of the 17th International Conference on Very Large Data Bases