International Journal of Cooperative Information Systems
Vol. 9, No. 3 (2000) 315-355

CLASSIFYING LOCAL QUERIES FOR GLOBAL QUERY OPTIMIZATION IN MULTIDATABASE SYSTEMS*

QIANG ZHU
Department of Computer and Information Science
The University of Michigan, Dearborn, MI 48128, U.S.A.

P.-Å. LARSON
Microsoft Research, One Microsoft Way
Redmond, WA 98052–6399, U.S.A.

A multidatabase system (MDBS) integrates information from multiple pre-existing local databases. A major challenge for global query optimization in an MDBS is that some required local information about local database systems such as local cost models may not be available at the global level due to local autonomy. A feasible method to tackle this challenge is to group local queries on a local database system into classes and then use the costs of sample queries from each query class to derive a cost formula for the class via regression analysis. This paper discusses the issues on how to classify local queries so that a good cost formula can be derived for each query class. Two classification approaches, i.e., bottom-up and top-down, are suggested. The relationship between these two approaches is discussed. Classification rules that can be used in the approaches are identified. Problems regarding composition and redundancy of classification rules are studied. Classification algorithms are given. To test the membership of a query in a class, an efficient algorithm based on ranks is introduced. In addition, a hybrid classification approach that combines the bottom-up and top-down ones is also suggested. Experimental results demonstrate that the suggested query classification techniques can be used to derive good local cost formulas for global query optimization in an MDBS.

Keywords: Multidatabase; query optimization; query classification; classification rules; rank-based membership testing; cost model; query sampling; regression analysis.

1. Introduction

In a large organization, data is typically stored in several (local) databases managed by different database management systems (DBMS), such as ORACLE, DB2, IMS and ObjectStore. To meet users' growing needs to access data in heterogeneous local databases, multidatabase systems (MDBS) have been studied by researchers in recent years. An MDBS is built on the top of existing local database systems (DBS). It integrates data from multiple local databases and provides users with a uniform global view of data. A global user can issue a (global) query on an MDBS

*Research supported in part by IBM Toronto Laboratory, Natural Sciences and Engineering Research Council of Canada, and US National Science Foundation under Grant # IIS-9811980.

to retrieve data from multiple local databases without knowing where the data is stored and how the data is retrieved.

A key feature of an MDBS is that it supports local autonomy. Briefly speaking, local autonomy refers to the situation where each local DBS retains complete control over its data and the global system can only interact with a local DBS at its external user interface. A local DBS continues to serve its existing local users. The MDBS, which serves global users, is just another user of local DBSs.

There are a number of challenges caused by local autonomy for global query optimization in an MDBS.[4,5,10,12,18,19] Among them, a crucial challenge is that some local information required by global query optimization, such as local cost information, may not be available at the global level in an MDBS. Lack of local information increases the difficulty for the global query optimizer to choose a good execution plan for a global query. The global query optimizer needs such local information to decide how to decompose a global query into local queries and where to execute the local queries. This problem does not exist in a traditional distributed database system (DDBS) because all sites run the same piece of software, i.e., the distributed database management system. Its query optimizer can make use of both global and local information to produce an efficient execution plan for a given global query.

How to solve the problem of incomplete local information for global query optimization in an MDBS has been investigated by a number of researchers recently. Du *et al.*[3] proposed a calibration method that uses observed costs of some special queries run on a special synthetic calibrating database at each local site to deduce desired local cost parameters. Gardarin *et al.*[6] extended Du *et al.*'s method so as to calibrate cost models for object-oriented local database systems in an MDBS. Zhu and Larson [20,24] suggested a fuzzy approach that makes use of fuzzy cost information in an MDBS to perform global query optimization. Naacke *et al.*[14] suggested an approach to combine a (default) generic cost model with specific cost information exported by wrappers for local DBSs (if available). Adali *et al.*[1] suggested to maintain a cost vector database to record cost information for every query issued to a local DBS, which is used to estimate costs for similar queries. Roth *et al.*[16] introduced a framework for costing in the *Garlic* federated system. Zhu *et al.*[25] proposed a qualitative approach to developing cost models for a dynamic multidatabase environment.

In [21,22,23], we introduced another method, called the query sampling method, to solve the problem. The key idea is to group local queries on a local DBS into homogeneous classes, draw a sample of queries from each query class, perform sample queries on the underlying DBS, and then use the observed costs of sample queries to derive a local cost formula for each query class by multiple regression. The derived cost formulas are kept in the MDBS catalog. To estimate the cost of a local query, the class to which the query belongs is first identified. The corresponding cost estimation formula is retrieved from the MDBS catalog and used to estimate the cost of the query. Based on the estimated local costs, the global query optimizer chooses a good execution plan for a global query.

Although a number of sampling techniques have been applied to query optimization in the literature,[7,11,13,15,17] all of them perform *data sampling* (i.e., sampling data from databases) instead of *query sampling* (i.e., sampling queries from a query class). Query sampling is our new idea for query optimization.

Apparently, the first step of the query sampling method, i.e., classifying local queries, is crucial for derivation of local cost formulas. A good query classification can yield good cost formulas. In [21,23], only a simple classification approach, i.e., top-down one, was described due to the limitation of the paper length. In this paper, we will discuss the issues for query classification in more details. In addition to the top-down approach, another bottom-up approach is introduced. The bottom-up approach provides a theoretic foundation for query classification. The relationship between two approaches is discussed. A common query classification that can be obtained in either approach is presented. An efficient rank-based method to test the membership of a query in a common query class is also suggested.

One type of query classification was considered by Icaza[8] for adaptive query optimization in a centralized database system. However, the criteria, approach and objective for query classification adopted by Icaza are different from the ones to be discussed in this paper. To our knowledge, no other similar work has been reported in the literature.

In this paper, we assume that the global data model in our MDBS is relational and each local DBMS is associated with an MDBS agent which provides a relational interface if the local DBMS is non-relational. Under this assumption, the global query optimizer in the MDBS may view all participating local DBMSs as relational ones. Since most common queries can be expressed by a sequence of select (σ), project (π) and join ($\bowtie$), only these three types of operations (so-called "work-horse" operations in a relational model) are considered in this paper. The cost of a query composed from these operations can be estimated by composing the costs of the operations. A select that may or may not be followed by a project is called a unary query. A join that may or may not be followed by a project is called a join query. Since the majority of practical joins are equijoins, only equijoins are considered. The qualification of a query is the condition that the tuples from the operand table(s) of the query should satisfy in order to be qualified for the result of the query. Without loss of generality, the qualifications of queries are assumed to be in conjunctive normal form. The basic predicates allowed for unary queries are of the form $R_i.a_n\ \theta\ C$, where $R_i.a_n$ stands for the column a_n in table R_i; C is a constant in the domain of a_n; $\theta \in \{=, \neq, >, <, \geq, \leq, nil\}$. $R_i.a_n\ nil\ C$ represents the '*true*' (empty) predicate. The basic predicates allowed for join queries are of the forms $R_i.a_n = R_j.a_m$ as well as $R_i.a_n\ \theta\ C$. Since we only consider equijoin queries, each join query has at least one conjunct $R_i.a_n = R_j.a_m$. Let $\wedge$ and $\vee$ denote the logical connectives AND and OR, respectively.

The rest of the paper is organized as follows. Section 2 discusses a query classification principle. Section 3 introduces a bottom-up approach for query classification.

Section 4 outlines a top-down approach for query classification and its relationship with the bottom-up approach. Section 5 presents an efficient method to test the membership of a query in a class. Section 6 shows some experimental results. The last section summarizes the conclusions and future research directions.

2. Classification Principle

For a given local database system, let G be the set of all queries that can be performed. The objective of query classification in the query sampling method is to group queries in G into classes so that the costs of queries in each query class can be accurately estimated by the same formula. In addition, characteristics of queries in each query class should be identified so that sample queries can be easily generated for the query class.

In one extreme case, if all queries in G were put into one class, i.e., employing the same formula to estimate all their costs, the cost formula would be very inaccurate. This is because it is difficult to capture the performance behavior of so many different queries in one formula.

In the other extreme case, if each query in G were considered as a separate class, i.e., employing a separate cost estimation formula for each query, we could get a very accurate cost formula for each query class. For example, for the query $\sigma_{R_1.a_1=27}(R_1)$ executed on ORACLE 6.0 running on an IBM RS/6000 model 220, where $R_1(a_1,\ a_2,\ a_3,\ a_4)$ is a table with 500 tuples of randomly-generated data, we can use a constant 0.091038 (sec.) as its cost estimation formula. Fig. 1 shows observed costs of this query executed at different times. For the costs in

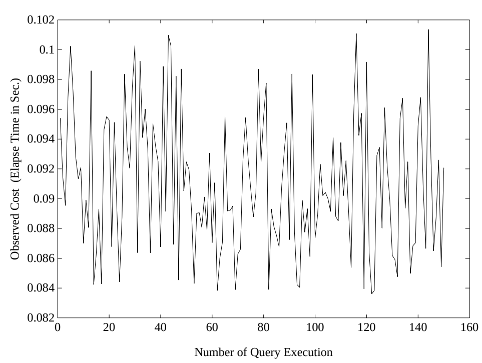

Fig. 1. Costs of Multiple Executions of Query $\sigma_{R_1.a_1=27}(R_1)$

Fig. 1, the estimation formula yields an average relative error of only 4% and a maximum relative error of only 11%. Actually, the estimation formula is the average of the observed costs. Therefore, theoretically speaking, the objective of query classification can always be fulfilled. However, maintaining a separate cost formula for every possible query is impossible since a huge (possibly infinite) number of cost formulas would be required. In [1], Adali *et al.* employed a cost vector database to record cost information for every query issued to a local DBS and used it to estimate the cost of a similar query. Although the number of executed queries is relatively small compared with the number of all possible queries, they realized that the maintenance problem would occur when the number of cost statistics to be recorded became too large. They mitigated the problem by using two compressing techniques, i.e., lossless summarizations and lossy summarizations.

In classifying queries for the query sampling method, we would prefer a classification with a small number of classes because, if so, we need to maintain only a small number of cost formulas — therefore the maintenance overhead is small. On the other hand, we would like to refine the query classes to improve estimation accuracy. As a result, the maintenance overhead increases. Obviously, neither of the two extremes is desirable. We seek an appropriate point (i.e., a classification) in the spectrum so that both accuracy and maintenance overhead are acceptable.

Notice that the costs of queries executed by using the same access method, such as an index join method or a nested-loop join method, can be estimated quite accurately by the same formula because their performance behavior most likely follow the same pattern. Therefore, it is a good principle to classify queries according to their access methods to be employed. However, which access method will be used for a local query may not be known at the global level in an MDBS. It depends on the underlying local DBMS.

Fortunately, there are some common policies for choosing an access method in most DBMSs. Based on available information and these common policies, we can group queries into homogeneous classes. The costs of all queries in a class are estimated by the same formula. If available information is sufficient, each query class corresponds to one access method indeed. The estimated costs are expected to be quite accurate in this case. If the available information is insufficient, it is possible that queries executed by different access methods are put in the same class. Since the practical goal of query optimization in an MDBS is the same as those of many traditional query optimizers, that is, avoiding bad execution plans instead of achieving a truly optimal one, estimation errors can be tolerated to some degree. In general, as long as an estimated cost and the corresponding real cost are of the same order of magnitude, the estimated cost is acceptable.

Unlike a query optimizer in a traditional DDBS, the global query optimizer in an MDBS has limited available information. To classify queries, the following types of information can be exploited:

- *characteristics of queries*: such as query types (unary or join), the number of conjuncts, types of predicates and so on. This type of information can be obtained by analyzing the syntax of a given query.
- *characteristics of operand tables*: such as the cardinality of a table, number of columns, indications of indexed and clustered-indexed columns and so on. This type of information can usually be obtained from the multidatabase catalog or local catalogs.

- *characteristics of underlying local DBMSs*: such as types of supported access methods and policies for choosing an access method for a query. This information can be obtained from the documents of a local DBMS, such as DBA's (database administer) guidance.

The above types of information are generally available from most DBMSs, hence we assume that they are the minimum available information at the global level in an MDBS. Some other types of information, such as (estimated) sizes of result tables, can also be used for classification. Some local DBMSs may provide additional information, such as the execution plan generated for a query. The additional information can be used to refine a classification, resulting in more accurate cost estimates.

From a mathematical point of view, for a given set G of queries, a classification (partition) $\Re$ of G is a set of subsets $G_1, G_2, \cdots, G_n \subseteq G$ such that $G_1 \cup G_2 \cup \cdots \cup G_n = G$ and $G_i \cap G_j = \emptyset$ (empty) for every $i \neq j$ $(1 \leq i, j \leq n)$. G_i is called a (equivalence) class of $\Re$. From the abstract algebra,[9] it is known that a classification $\Re$ and an equivalence relation $\sim$ can be induced from each other. To classify a set of queries, we thus need to define an equivalence relation among queries.

3. Bottom-up Classification Approach

There are basically two classification approaches: bottom-up and top-down. If we start with the classification where each query comprises a separate class and reduce the number of classes by merging classes into larger superclasses, we are using a bottom-up classification approach. If we start with the classification where all queries are in one class and refine the classification by dividing a class into smaller subclasses, we are using a top-down classification approach. These two classification approaches are to be discussed in this and the following sections.

A common policy or actual rule for choosing an access method for a query can usually result in an equivalence relation. An equivalence relation specifies what queries are in the same class. It therefore introduces a classification rule that characterizes the queries in each query class for a classification. As we will see in this section, a bottom-up classification approach can use one or more such classification rules to produce a satisfactory classification.

3.1. Classification Rules

Let us consider several useful classification rules. Note that not all classification rules discussed here are required for classifying a specific query set. Moreover, as we will see, some classification rules may be implied by one or more other classification rules.

Parameter substitution rule

Consider a query

$$\sigma_{R_1.a_1=13}(R_1). \tag{3.1}$$

Suppose a_1 is an indexed column of table R_1. A local DBMS usually chooses the same access method (i.e., an index scan method in this case) for the query even if we replace constant 13 by another constant 174, i.e., $\sigma_{R_1.a_1=174}(R_1)$. Furthermore, if we replace R_1 by R_2, a_1 in R_1 by a_2 in R_2, and 13 by 67 in (3.1), the DBMS usually still employs the same index scan method for the new query $\sigma_{R_2.a_2=67}(R_2)$ as long as $R_2.a_2$ is also indexed. Hence, these queries can be in the same query class.

We notice that these queries have the general form:

$$F(R,\ a,\ C)\ =\ \sigma_{R.a\ =\ C}(R), \tag{3.2}$$

Query (3.1), for example, is $F(R_1,\ a_1,\ 13)$. It is obtained by substituting a specific parameter value for each parameter in (3.2).

In general, a query formula such as (3.2) is a valid formula formed by relational operations (σ, π, $\bowtie$), comparison operators ($=$, $\neq$, $<$, $\leq$, $>$, $\geq$, nil), logical connectives ($\wedge$, $\vee$), and a number of table, column and constant parameters. For a given query formula, a query can be obtained by instantiating the parameters with specific instances (parameter values). Different instantiations result in different queries. A query formula, therefore, represents a set of queries.

A DBMS may or may not employ the same access method to execute two queries that result from the same formula. We need to find conditions under which a DBMS would most likely employ the same access method for the queries resulting from the same formula.

Assume that each column in a table has one (and only one) of the following physical properties: (1) the column is clustered-indexed; (2) the column is indexed; or (3) the column is sequential without any index. Assume the following statistics are maintained for each table: (a) the cardinality of the table, (b) the tuple length of the table, (c) the maximum and minimum values of each column, and (d) the number of distinct values in each column. Note that the following discussion would be similar if there exist more alternative physical properties and more statistics for a local database.

There are two types of query optimizers. One is based on heuristic rules, the other is based on cost analysis. A real query optimizer may be a mixture of the two types. Different types of optimizers choose access methods based on different types of information.

For a heuristic-based query optimizer, it chooses an access method mainly according to the syntax (formula) of a given query and the physical properties of its referenced columns. Hence, a heuristic-based query optimizer usually chooses the same access method for two queries if their formulas and the physical properties of the corresponding referenced columns are the same.

<u>Definition</u> 1. Let Q_1 and Q_2 be two queries resulting from the same formula. If Q_2 can be obtained by replacing the parameter values (i.e., tables, columns and constants) in Q_1 with another set of parameter values, where each replacing column has the same physical property as the corresponding replaced column, we say that Q_2 is related to Q_1 via a relation $\overset{r_1}{\sim}$, denoted as $Q_2 \overset{r_1}{\sim} Q_1$.

It is not difficult to see that relation $\overset{r_1}{\sim}$ is an equivalence relation. Using equivalence relation $\overset{r_1}{\sim}$, we have the following classification rule:

<u>Classification Rule</u> r_1: *Two queries Q_1 and Q_2 resulting from the same formula are in the same class if $Q_1 \overset{r_1}{\sim} Q_2$.*

Using classification rule r_1, we can first get a classification of the subset of queries with respect to a given query formula first. A classification of all queries can be obtained by combining the classifications for all query formulas. Hence, classification rule r_1 yields a classification for all queries. Such a classification is acceptable if the query optimizer of the underlying local DBMS is heuristic-based. However, it may be unacceptable if the query optimizer is cost-based. A cost-based query optimizer usually chooses an access method according to not only the formula of a query and the physical properties of the referenced columns; but also some statistics about the operand tables and the referenced columns, as well as the constants appearing in the query. If two queries have the same formula, the same physical properties for the relevant referenced columns, the same statistics about their corresponding operand tables and columns, and the same referenced constants, a cost-based query optimizer usually chooses the same access method for the two queries[a]. Thus a cost-based query optimizer requires more homogeneous properties than a heuristic-based query optimizer does to choose the same access method.

<u>Definition</u> 2. Let Q_1 and Q_2 be two queries resulting from the same formula. If Q_2 can be obtained by replacing the parameter values in Q_1 with another set of parameter values where (a) each replacing column has the same physical property as the corresponding replaced column; (b) each replacing parameter value has the same statistics, if any, as the corresponding replaced parameter value; (c) the corresponding constant values referred to in the two queries are the same; we say that Q_2 is related to Q_1 via a relation $\overset{r_2}{\sim}$, denoted as $Q_2 \overset{r_2}{\sim} Q_1$.

Again, it is not difficult to see that $\overset{r_2}{\sim}$ is an equivalence relation. It induces the following classification rule:

<u>Classification Rule</u> r_2: *Two queries Q_1 and Q_2 resulting from the same formula are in the same class if $Q_1 \overset{r_2}{\sim} Q_2$.*

<u>Definition</u> 3. An equivalence relation $\overset{r_i}{\sim}$ is said to be weaker than another relation $\overset{r_j}{\sim}$, if $Q_1 \overset{r_i}{\sim} Q_2$ implies $Q_1 \overset{r_j}{\sim} Q_2$ for any queries Q_1 and Q_2 in a given query

[a]Like most DBMSs, data in operand tables is assumed to follow the same data distribution, e.g., the uniform distribution.

set G. In this case, we also say that the induced classification rule r_i is weaker than the induced classification rule r_j. If both $\overset{r_i}{\sim}$ and $\overset{r_j}{\sim}$ are used to classify G on a local DBMS, then $\overset{r_i}{\sim}$ (i.e., r_i) is redundant.

If $\overset{r_i}{\sim}$ is weaker than $\overset{r_j}{\sim}$, any class of the classification induced by $\overset{r_i}{\sim}$ is completely contained in a class of the classification induced by $\overset{r_j}{\sim}$. If both are used to classify a query set, the weaker relation is useless. The weaker relation is useful only if the stronger relation is not applied. To improve the efficiency of a classification procedure, redundant relations (classification rules) should not be used.

Clearly, relation $\overset{r_2}{\approx}$ is weaker than relation $\overset{r_1}{\sim}$, since if $Q_1 \overset{r_2}{\approx} Q_2$, then $Q_1 \overset{r_1}{\sim} Q_2$. However, the inverse may not be true.

Example 1. Let $R_i(a_1, a_2)$ $(i = 1, 2, 3)$ be three tables in a local database, where a_1 and a_2 are an indexed column and a sequential column of table R_i, respectively. The statistics maintained in the multidatabase catalog are given in Table 1. Consider the following queries resulting from the same formula but with different parameter values:

$$\begin{aligned} Q_1: &\quad \sigma_{R_1.a_1=24}(R_1), \\ Q_2: &\quad \sigma_{R_2.a_1=24}(R_2), \\ Q_3: &\quad \sigma_{R_3.a_1=24}(R_3), \\ Q_4: &\quad \sigma_{R_3.a_2=70}(R_3), \\ Q_5: &\quad \sigma_{R_2.a_2=70}(R_2). \end{aligned}$$

By the definitions, $Q_1 \overset{r_2}{\approx} Q_2$, $Q_1 \overset{r_1}{\sim} Q_3$, $Q_2 \overset{r_1}{\sim} Q_3$, and $Q_4 \overset{r_1}{\sim} Q_5$; but Q_1 *not* $\overset{r_2}{\approx} Q_3$, Q_1 *not* $\overset{r_1}{\sim} Q_4$, Q_2 *not* $\overset{r_2}{\approx} Q_3$, Q_4 *not* $\overset{r_2}{\approx} Q_5$, and so on. □

Table 1. Statistics for Tables in Example 1

$i = 1, 2, 3$	R_1	R_2	R_3
$\|R_i\|$	500	500	600
$Len(R_i)$	80	80	65
$Max(R_i.a_1)$	1000	1000	1000
$Min(R_i.a_1)$	75	75	98
$\|R_i.a_1\|$	450	450	500
$Max(R_i.a_2)$	150	90	90
$Min(R_i.a_2)$	10	5	5
$\|R_i.a_2\|$	75	60	60

$|\cdot|$ — cardinality, $Len(\cdot)$ — tuple length
$Max(\cdot)$ — maximum, $Min(\cdot)$ — minimum

The applicability of Classification Rule r_2 is rather restrictive because it is rare that two practical queries would have the same statistics for corresponding operand tables and columns and use the same corresponding constants. To reduce the requirements of (b) and (c) in Definition 2, we partition the range of a statistic value or constant value into subranges. If two queries resulting from the same formula have their corresponding statistic values and constant values in the same relevant subranges, the two queries have "similar" corresponding statistics and constants. A query optimizer likely chooses the same access method for the two queries.

In the following discussion, we assume that each column is of a numeric data type[b]. Without loss of generality, we also assume that each type of statistic, such as maximum or cardinality, has the same range of values for all tables or columns[c].

Let $L_1 \le |R_i| \le U_1$, $L_2 \le Len(R_i) \le U_2$, $L_3 \le Max(R_i.a_n) \le U_3$, $L_4 \le Min(R_i.a_n) \le U_4$ and $L_5 \le |R_i.a_n| \le U_5$, where R_i is any table, $R_i.a_n$ is any column. We partition the range (interval) $I_j = [L_j,\ U_j]$ $(j = 1,\ ...,\ 5)$ into $N_j (\ge 1)$ equal length subranges (subintervals): $I_{j\ k} = [L_j + (k-1) * h_j,\ L_j + k * h_j)$ for $k = 1,\ ...,\ N_j - 1$ and $I_{j\ N_j} = [L_j + (N_j - 1) * h_j,\ U_j]$, where $h_j = (U_j - L_j)/N_j$. Similarly, we assume each constant referred to in a query has the same range[d] $I_0 = [L_0,\ U_0]$. We divide I_0 into N_0 (≥ 1) subranges: $I_{0\ k} = [L_0 + (k-1)*h_0,\ L_0 + k*h_0)$ for $k = 1,\ ...,\ N_0 - 1$ and $I_{0\ N_0} = [L_0 + (N_0 - 1)*h_0,\ U_0]$, where $h_0 = (U_0 - L_0)/N_0$.

The above partition Π for statistics and constants is called a range partition (see Fig. 2). $I_{j\ k}$ $(j = 0,\ ...,\ 5; k = 1,\ ...,\ N_j)$ is called a subrange in Π. If two relevant

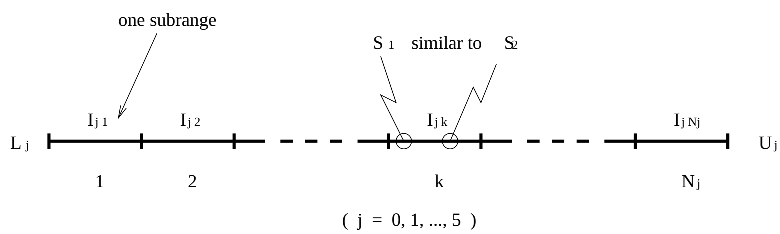

Fig. 2. Range Partition for Statistics and Constants

statistics (or constants) are within the same subrange $I_{j\ k}$, one is called a similar statistic (or constant) of the other with respect to (w.r.t.) Π.

<u>Definition</u> 4. Let Q_1 and Q_2 be two queries resulting from the same formula. For a given range partition Π, if Q_2 can be obtained by replacing the parameter values in Q_1 with another set of parameter values where (a) each replacing column has the same physical property as the corresponding replaced column; (b) each replacing parameter value has the similar statistics, if any, to those of the corresponding replaced parameter value w.r.t Π; (c) each replacing constant is a similar constant of the corresponding replaced constant w.r.t Π, we say that Q_2 is related to Q_1 w.r.t. Π via a relation $\stackrel{r_3}{\sim}_\Pi$, denoted as $Q_2 \stackrel{r_3}{\sim}_\Pi Q_1$.

Since relation $\stackrel{r_3}{\sim}_\Pi$ is reflexive, symmetric and transitive, it is an equivalence relation. It induces the following classification rule:

<u>Classification Rule</u> r_3: *For a given range partition* Π*, two queries* Q_1 *and* Q_2 *resulting from the same formula are in the same class if* $Q_1 \stackrel{r_3}{\sim}_\Pi Q_2$.

[b] A non-numeric data type, such as date and string, usually can be mapped into a numeric data type. The following discussion can be easily extended to such non-numeric data types.

[c] If one type of statistic, such as $|R_i|$ (cardinality), has different ranges for different tables, a unified range can be formed by taking the largest upper bound of the ranges as the upper bound and the smallest lower bound of the ranges as the lower bound.

[d] The range for a constant is determined by the domain of the related column in an operand table of the query.

Example 2. Consider the tables in the Example 1. Assume the ranges I_j ($j =$ 0, 1, $\cdots$, 5) for a constant, $|R_i|$, $Len(R_i)$, $Max(R_i.a_n)$, $Min(R_i.a_n)$ and $|R_i.a_n|$ are $[-1000,\ 5000]$, $[0,\ 1600]$, $[4,\ 512]$, $[100,\ 5000]$, $[-1000,\ 1000]$ and $[1,\ 1501]$, respectively. Let Π be a range partition whose numbers N_j ($j = 0,\ \ldots,\ 5$) of subranges are 60, 4, 4, 49, 20 and 15, respectively. Then, $Q_1 \overset{r_3}{\sim}_{\Pi} Q_2$ because Q_1 and Q_2 have the same corresponding statistics and constant — they are in the same relevant subranges, and $Q_1 \overset{r_3}{\sim}_{\Pi} Q_3$ because $24 \in I_{0\ 11}$; $|R_1|,\ |R_3| \in I_{1\ 2}$; $Len(R_1),\ Len(R_3) \in I_{2\ 1}$; $Max(R_1.a_1),\ Max(R_3.a_1) \in I_{3\ 10}$; $Min(R_1.a_1),\ Min(R_3.a_1) \in I_{4\ 11}$; and $|R_1.a_1|,\ |R_3.a_1| \in I_{5\ 5}$. □

From Definitions 2 – 4, we can see that $\overset{r_2}{\sim}$ is weaker than $\overset{r_3}{\sim}_{\Pi}$ and that $\overset{r_3}{\sim}_{\Pi}$ is weaker than $\overset{r_1}{\sim}$. In fact, relations $\overset{r_2}{\sim}$ and $\overset{r_1}{\sim}$ are two extreme cases for relation $\overset{r_3}{\sim}_{\Pi}$. If all N_j ($j = 0,\ \ldots,\ 5$) for a given range partition Π are 1; i.e., there is only one subrange for the range of each statistic or constant, relation $\overset{r_3}{\sim}_{\Pi}$ boils down to relation $\overset{r_1}{\sim}$. If each value in the range of a statistic or constant comprises a separate subrange for range partition Π, relation $\overset{r_3}{\sim}_{\Pi}$ becomes relation $\overset{r_2}{\sim}$. Although Classification Rule r_2 could give the best accuracy, it would yield too many classes. Hence the overhead for maintaining cost estimation formulas would be high. The range partition Π can be used to reduce the number of query classes — therefore the maintenance overhead.

Reducing the number of query classes does not necessarily mean loss of accuracy. It is quite often that a cost analysis result agrees with a heuristic rule. Therefore, in practice, it is usually sufficient to use Classification Rule r_1 to classify queries for a local database.

Similar predicates merging rule

We have considered how to put queries resulting from the same formula but with different parameter values into a query class. Can we put two queries with different formulas into the same query class? The answer is positive.

For example, most DBMSs choose the same access method for the following two queries: $\sigma_{R_1.a_1 \leq 24}(R_1)$ and $\sigma_{R_1.a_1 < 24}(R_1)$. This is because the predicates $R_1.a_1 \leq 24$ and $R_1.a_1 < 24$ are similar in terms of performance.

In general, the following pairs of predicates are similar to each other:

1. a predicate with $<$ and a predicate with $\leq$, such as $R_i.a_n < C$ and $R_i.a_n \leq C$;

2. a predicate with $>$ and a predicate with $\geq$, such as $R_i.a_n > C$ and $R_i.a_n \geq C$;

3. a predicate with $\neq$ and the '*true*' (empty) predicate, such as $R_i.a_n \neq C$ and '*true*';

4. a predicate and itself[e].

[e]The last condition is required by the reflexive property of an equivalence relation

<u>Definition</u> 5. Let Q_1 and Q_2 be two queries. If Q_2 can be obtained by replacing some predicates in Q_1 with similar predicates, we say that Q_2 is related to Q_1 via a relation $\stackrel{r_4}{\sim}$, denoted as $Q_2 \stackrel{r_4}{\sim} Q_1$.

The equivalence relation $\stackrel{r_4}{\sim}$ induces the following classification rule:

<u>Classification Rule</u> r_4: *Two queries Q_1 and Q_2 are in the same class if $Q_1 \stackrel{r_4}{\sim} Q_2$.*

Example 3. The following queries are related to each other via $\stackrel{r_4}{\sim}$:

$$
\begin{aligned}
Q_1 :&\quad R_1 \underset{R_1.a_1>3 \ \wedge \ R_1.a_2=R_2.a_1}{\bowtie} R_2,\\
Q_2 :&\quad R_1 \underset{R_1.a_1\geq 3 \ \wedge \ R_1.a_2=R_2.a_1}{\bowtie} R_2,\\
Q_3 :&\quad R_1 \underset{R_1.a_1>3 \ \wedge \ R_1.a_2=R_2.a_1 \ \wedge \ R_2.a_2\neq 5}{\bowtie} R_2 \ .
\end{aligned}
$$

Thus Q_1, Q_2 and Q_3 are in the same query class if Classification Rule r_4 is applied. □

Project lists merging rule

In addition to similar predicates, we also notice that DBMSs often choose the same access method for two queries with different target (project) column lists but the same remaining part. For example, a DBMS usually choose the same access method for the following two queries: $\pi_{a_5,a_2,a_3}(\sigma_{a_1=4 \ \wedge \ a_2\neq 5}(R_1))$ and $\pi_{a_2,a_4}(\sigma_{a_1=4 \ \wedge \ a_2\neq 5}(R_1))$.

Based on this fact, we have the following definition:

<u>Definition</u> 6. Let Q_1 and Q_2 be two queries. If Q_2 can be obtained by replacing the project list in Q_1 with another (maybe the same) project list, we say that Q_2 is related to Q_1 via a relation $\stackrel{r_5}{\sim}$, denoted as $Q_2 \stackrel{r_5}{\sim} Q_1$.

It is trivial to see that the relation $\stackrel{r_5}{\sim}$ is an equivalence relation. This relation induces the following classification rule:

<u>Classification Rule</u> r_5: *Two queries Q_1 and Q_2 are in the same class if $Q_1 \stackrel{r_5}{\sim} Q_2$.*

Join order changing rule

Usually, a DBMS chooses an access method for a join query based on the physical properties and statistics of operand tables and their columns, regardless of the order of operand tables specified by a user. For example, a DBMS usually chooses the same access method for the following two queries: $R_1 \underset{R_1.a_1>3 \ \wedge \ R_1.a_2=R_2.a_1}{\bowtie} R_2$ and $R_2 \underset{R_1.a_1>3 \ \wedge \ R_1.a_2=R_2.a_1}{\bowtie} R_1$.

Based on this observation, we have the following definition:

<u>Definition</u> 7. Let Q_1 and Q_2 be two (join) queries. If Q_2 can be obtained by changing the order of join operand tables in Q_1 or $Q_2 = Q_1$, we say that Q_2 is related to Q_1 via a relation $\stackrel{r_6}{\sim}$, denoted as $Q_2 \stackrel{r_6}{\sim} Q_1$.

It is trivial to see that the relation $\overset{r_6}{\sim}$ is an equivalence relation. This relation induces the following classification rule:

<u>CLASSIFICATION RULE</u> r_6: *Two queries Q_1 and Q_2 are in the same class if $Q_1 \overset{r_6}{\sim} Q_2$.*

Common policy rules

There are some common policies used in many DBMSs to choose an access method for a query. These common policies may yield useful query classification rules. Some of such common policies are listed in Tables 2 and 3.

Table 2. Some Common Policies for Unary Queries

Label	Description of Common Policies
r_7	a clustered-index scan method with a key value is usually chosen if the qualification of a query has at least one conjunct $R_i.a_n = C$ where $R_i.a_n$ is clustered-indexed
r_8	an index scan method with a key value is usually chosen if r_7 is inapplicable and the qualification of a query has at least one conjunct $R_i.a_n = C$ where $R_i.a_n$ is indexed
r_9	a clustered-index scan method with a range is usually chosen if r_7 and r_8 are inapplicable and the qualification of a query has at least one conjunct $R_i.a_n\ \theta\ C$ where $R_i.a_n$ is clustered-indexed and $\theta \in \{<, \leq, >, \geq,\}$
r_{10}	an index scan method with a range is usually chosen if r_7 – r_9 are inapplicable and the qualification of a query has at least one conjunct $R_i.a_n\ \theta\ C$ where $R_i.a_n$ is indexed and $\theta \in \{<, \leq, >, \geq,\}$
r_{11}	a sequential scan method is usually chosen if r_7 – r_{10} are inapplicable

A common policy, such as r_7, is said to be inapplicable if either the underlying DBMS does not support the relevant access method or the qualification of the query does not have a required conjunct. A predicate $R_i.a_n\ \theta\ C$ ($\theta \in \{=, <, >, \leq, \geq\}$) is said to be index-usable if $R_i.a_n$ is (clustered or non-clustered) indexed. A conjunct is said to be index-usable if every predicate in the conjunct is index-usable. For example, the conjunct $[R_1.a_1 < 3 \ \vee \ R_1.a_2 = 9 \ \vee \ R_1.a_3 \leq 2]$ is index-usable if all the involved predicates are index-usable.

Table 3. Some Common Policies for Join Queries

Label	Description of Common Policies
r_{12}	a clustered-index join method is usually chosen if the qualification of a query has at least one conjunct $R_i.a_n = R_j.a_m$ where either $R_i.a_n$ or $R_j.a_m$ (or both) is clustered-indexed
r_{13}	an index join method is usually chosen if r_{12} is inapplicable and the qualification of a query has at least one conjunct $R_i.a_n = R_j.a_m$ where either $R_i.a_n$ or $R_j.a_m$ (or both) is indexed
r_{14}	a nested-loop join method with reduction on operand table(s) via index(es) is usually chosen if r_{12} and r_{13} are inapplicable and the qualification of a query has at least one index-usable conjunct for R_i or R_j (or both)
r_{15}	a sort-merge join method is usually chosen if r_{12} – r_{14} are inapplicable

Each common policy in Tables 2 and 3 specifies what queries are most likely to be executed by the same access method. Therefore, an equivalence relation can be defined based on such a policy.

Definition 8. Let Q_1 and Q_2 be two queries. For a given common policy $r \in \{r_7,\ r_8,\ \cdots,\ r_{15}\}$, if Q_1 and Q_2 use the same access method[f] according to the policy r, we say that Q_1 and Q_2 are related to each other via a relation $\stackrel{r}{\sim}$, denoted as $Q_1 \stackrel{r}{\sim} Q_2$, or $Q_2 \stackrel{r}{\sim} Q_1$.

The induced classification rule is:

Classification Rule r: *For a given $r \in \{r_7,\ r_8,\ \cdots,\ r_{15}\}$, two queries Q_1 and Q_2 are in the same class if $Q_1 \stackrel{r}{\sim} Q_2$.*

Note that we use the same label for both a common policy and its induced classification rule.

Example 4. For common policy r_7, the corresponding classification rule r_7 would give us a classification in which there is a class $G_{1\ 1}$ that contains all the queries whose qualifications have at least one conjunct $R_i.a_n = C$ where $R_i.a_n$ is clustered-indexed, and every other query comprises a separate class on its own. We call $G_{1\ 1}$ a major class under r_7. □

In fact, there is another common policy implied by $r_7 - r_{11}$; that is, the preference of the following access methods for unary queries usually decreases in a DBMS:

1. clustered-index scan method with a key value,
2. index scan method with a key value,
3. clustered-index scan method with a range,
4. index scan method with a range,
5. sequential scan method.

Similarly, there is a common policy implied by $r_{12} - r_{15}$; that is, the preference of the following access methods for join queries usually decreases in a DBMS:

1. clustered-index join method,
2. index join method,
3. nested-loop join method with reduction on operand table(s) via index(es),
4. sort-merge join method.

These two implied common policies do not yield separate classification rules on their own. They are implemented through the common policies in Tables 2 and 3.

[f] Assume that the same access method is chosen for Q_1 and Q_2 by any common policy if they are the same query.

Example 5. Consider the following queries:

$$\begin{aligned}
Q_1 :& \quad R_1 \underset{R_1.a_1 = R_2.a_1}{\bowtie} R_2,\\
Q_2 :& \quad R_1 \underset{R_1.a_2 = R_2.a_2 \,\wedge\, R_1.a_1 = R_2.a_3}{\bowtie} R_2,\\
Q_3 :& \quad R_1 \underset{R_1.a_2 = R_2.a_3}{\bowtie} R_2,
\end{aligned}$$

where $R_1.a_1$ and $R_2.a_1$ are clustered-indexed, $R_1.a_2$, $R_2.a_2$ and $R_2.a_3$ are (non-clustered) indexed. We have $Q_2 \overset{r_{12}}{\sim} Q_1$, but Q_2 not $\overset{r_{13}}{\sim} Q_3$, from Classification Rules r_{12} and r_{13}. In other words, a DBMS usually prefers to choose a clustered-index method for Q_2 although an index-based method is also feasible for it. □

Adjustment of classification rules

The classification rules discussed so far can be used for many DBMSs. However some of them may need to be adjusted for a specific DBMS because a policy for choosing an access method in the DBMS may be inconsistent with one or more common policies discussed above.

For example, a DBMS may choose a sequential scan method if $r_7 - r_9$ are inapplicable and the qualification of a given query has at least one conjunct $R_i.a_n \ \theta \ C$ where $R_i.a_n$ is indexed and $\theta \ \in \ \{<, \ >, \ \leq, \ \geq\}$. This policy is inconsistent with the common policy r_{10} in Table 2. This policy can then be merged with the policy r_{11} to get a new policy r'_{10}. We may, therefore, replace r_{10} and r_{11} by this actual policy r'_{10} when we classify queries for the DBMS.

In fact, if we keep using r_{10} and r_{11} to classify queries for this DBMS, the obtained classification is still good. The reason for this is that we then get two classes and each of them consists of the queries executed by the same access method although the name of access method for one class is not guessed correctly. Hence the name of access method in a common policy is not essential. The important thing is that the relevant queries use the same access method.

Furthermore, even if the queries in a class may use different access methods, as long as these access methods behave similarly in terms of performance the classification is still satisfactory. Therefore, a classification based on the common policies is usually robust.

In general, for a given DBMS, if an actual policy is not consistent with one or more common policies, some adjustments for the relevant common policies can be made to reflect the actual situation.

If the developer of a local DBMS is the same as the developer of the MDBS, all actual policies for choosing access methods on the DBMS are known. Some actual policies may be cost-based; i.e., if the value of a cost function is less than the value of another, a particular access method is chosen; otherwise, another access method is chosen. They can be used in the same way as a heuristic-based policy for classification. A classification rule can be induced by an actual policy, like a common rule.

In general, the more information we have about a local DBMS (policies), the better classification rules we can get, and therefore a better classification would result.

Note that some (local) DBMSs such as ORACLE and DB2 have a facility to describe (explain) the access method chosen for a (local) query. For this type of DBMSs, the common policies (therefore the classification rules) can be verified by running some test queries and checking their access methods. Actual policies on such a DBMS can be guessed via experiments. The common policies used for classification can be adjusted accordingly. However, some MDBSs such as EMPRESS provide no information about the execution of a query. The common policies can be assumed to be valid for these DBMSs. The goodness of the classification depends on the degree of agreement between the common policies and the actual policies. Since the common policies are usually robust, we expect such a classification to be acceptable. A classification rule is said to be valid for a DBMS if it is chosen to be used in classifying queries for the system.

3.2. Composition of Classification Rules

As we have seen, many classification rules may be used to classify queries for a local DBMS. Each of them can yield a classification. How can we compose them so that one composite classification can be obtained?

<u>Definition</u> 9. Let Q' and Q'' be two queries in G, and Γ be a non-empty set of valid classification rules. Q'' is said to be related to Q' w.r.t. Γ via a (composite) relation $\overset{*}{\sim}_\Gamma$ if there exist $x_1, x_2, \cdots, x_n \in \Gamma$ $(n \geq 1)$ and queries $Q_1, Q_2, \cdots, Q_{n-1} \in G$ such that $Q'' \overset{x_1}{\sim} Q_1 \overset{x_2}{\sim} Q_2 \overset{x_3}{\sim} \cdots \overset{x_{n-1}}{\sim} Q_{n-1} \overset{x_n}{\sim} Q'$, denoted as $Q'' \overset{*}{\sim}_\Gamma Q'$.

It is not difficult to see that the relation $\overset{*}{\sim}_\Gamma$ defined in Definition 9 on the query set G is an equivalence relation, which induces the following classification rule for G:

<u>Composite Classification Rule</u> $*_\Gamma$: *For a given set of valid classification rules Γ, two queries Q_1 and Q_2 are in the same class w.r.t. Γ if $Q_1 \overset{*}{\sim}_\Gamma Q_2$.*

The classification obtained by using the composite classification rule $*_\Gamma$ is called a classification w.r.t. Γ. If Γ is empty, we assume that each query in the given query set comprises a separate class of the classification w.r.t. Γ.

3.3. Classification Algorithm

To classify a given query set by using a set of classification rules, the following lemmas are useful.

<u>Lemma</u> 1. Let Γ (may be empty) be a set of valid classification rules, $\Re$ be the classification w.r.t. Γ for a given query set G, $\Gamma' = \Gamma \cup \{w\}$ where w is another valid classification rule that is not in Γ. For any two different classes $\Im_1, \Im_2 \in \Re$, if there exist $s \in \Im_1$ and $t \in \Im_2$ such that $s \overset{w}{\sim} t$, then $\Im_1 \cup \Im_2$ is contained in a class of the new classification $\Re'$ w.r.t. Γ'.

PROOF. It is trivial for $\Gamma = \emptyset$. Assume $\Gamma \neq \emptyset$. Since $\Gamma \subseteq \Gamma'$, if $x \overset{*}{\sim}_{\Gamma} y$ then $x \overset{*}{\sim}_{\Gamma'} y$ for any $x, y \in \Im_1$ or $x, y \in \Im_2$. For any $x \in \Im_1$ and $y \in \Im_2$, since $x \overset{*}{\sim}_{\Gamma'} s$, $s \overset{w}{\sim} t$ and $t \overset{*}{\sim}_{\Gamma'} y$, we have $x \overset{*}{\sim}_{\Gamma'} y$. Therefore, $\Im_1 \cup \Im_2$ is contained in the same class of the new classification $\Re'$. □

Lemma 1 states that two old classes can be merged into one class that is contained in a class of the new classification if there exist two queries in the two old classes such that they are related via a new relation corresponding to the new classification rule (See Fig. 3). Furthermore, we have

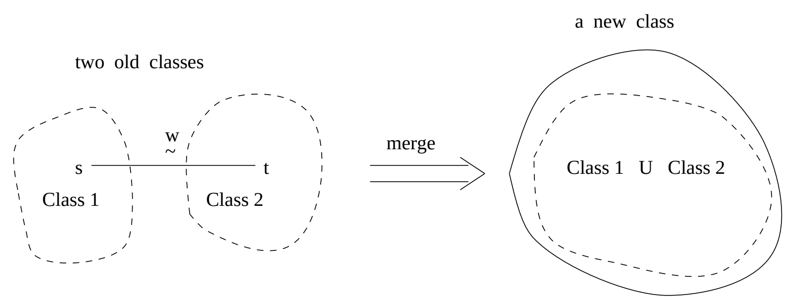

Fig. 3. Merging Two Old Classes into a New Class

<u>LEMMA</u> 2. Let $\Re$ and $\Re'$ be the classifications in Lemma 1. Let $\Im_1$ ($\Im_2$) be a class of $\Re$ or a union of classes of $\Re$ that is contained in a class of $\Re'$. $\Im_1 \cup \Im_2$ is contained in a class of $\Re'$ if there exist $s \in \Im_1$ and $t \in \Im_2$ such that $s \overset{w}{\sim} t$.

PROOF. It is trivial for $\Gamma = \emptyset$. Assume $\Gamma \neq \emptyset$. Notice that $x \overset{*}{\sim}_{\Gamma'} y$ for any $x, y \in \Im_1$ or $x, y \in \Im_2$ under the assumptions about $\Im_1$ and $\Im_2$ in the lemma. The rest of proof is similar to that for Lemma 1. □

Lemma 2 says that the merging procedure in Lemma 1 can be recursively applied to the classes that are resulted from a previous application of the merging procedure.

The following lemma describes the relationship between a class in an old classification and a class in a new upgraded classification.

<u>LEMMA</u> 3. Let Γ, Γ' be two sets of valid classification rules, $\Gamma' \supseteq \Gamma$, $\Re$ be the classification w.r.t. Γ, and $\Re'$ be the classification w.r.t. Γ'. For any classes $\Im \in \Re$ and $\Im' \in \Re'$, if $\Im \cap \Im'$ is not empty, then $\Im \subseteq \Im'$.

PROOF. If $\Gamma = \emptyset$, $\Im = \Im \cap \Im' \subseteq \Im'$. Assume $\Gamma \neq \emptyset$. Since $\Gamma \subseteq \Gamma'$, for any $x, y \in \Im$, $x \overset{*}{\sim}_{\Gamma'} y$. Since $\Im \cap \Im'$ is not empty, there exists $z \in \Im \cap \Im'$. Hence for any $x \in \Im$, $x \overset{*}{\sim}_{\Gamma'} z \in \Im'$. Therefore, $\Im \subseteq \Im'$. □

<u>COROLLARY</u> 1. Any class of $\Re$ is contained in a class of $\Re'$.

PROOF. Clearly, an empty class (set) is contained in any other class (set). For a non-empty class of $\Re$, it contains a query that should belong to a class of $\Re'$. Hence the claim is true by Lemma 3. □

Based on Lemmas 1 – 3, the following bottom-up classification algorithm to classify a query set is given.

<u>ALGORITHM</u> 1. : Classifying a query set using the bottom-up approach
Input: A query set G and a set Γ of valid classification rules
Output: A classification $\Re$ for G w.r.t. Γ
Method:

1. **begin**
2. initially, take each query in G as a separate class of $\Re$;
3. **while** Γ is not empty **do**
4. take $w \in \Gamma$;
5. change_flag := true;
6. **while** change_flag = true **do**
7. **if** there exist two classes $\Im_1, \Im_2$ in $\Re$ such that
8. there exist $s \in \Im_1, t \in \Im_2$ and $s \overset{w}{\sim} t$ **then**
9. $\Re := (\Re - \{\Im_1, \Im_2\}) \cup \{\Im_1 \cup \Im_2\}$;
10. **else** change_flag := false;
11. **end**;
12. $\Gamma := \Gamma - \{w\}$;
13. **end**;
14. **return** $\Re$;
15. **end.**

Let us show that Algorithm 1 generate the correct classification for the query set.

<u>THEOREM</u> 1. **(Correctness)** For a set G of queries and a set Γ of valid classification rules, Algorithm 1 returns the classification $\Re$ for G w.r.t. Γ.

PROOF.

Let m be the number of passes that have been finished for the loop 3 – 13. Let $\Gamma = \{ w_1, w_2, \cdots, w_{|\Gamma|} \}$ where w_m be the classification rule considered in the m-th loop. We claim that after m loops the current classification $\Re$ is the classification $\Re_m$ w.r.t. $\Gamma_m = \{ w_1, w_2, \cdots, w_m \}$. We prove this claim by mathematical induction on m.

Case 1. $m = 1$. Initially, each query in G is taken as a separate class in the current $\Re$. $\Re$ is actually the classification w.r.t. an empty set of classification rules. After the loop 3 – 13 is executed once, every (non-empty) class X of the current $\Re$ is contained in a class Y of $\Re_1$ by Lemmas 1 and 2, i.e., $X \subseteq Y$. On the other hand, for any $x, y \in Y$, $x \overset{w_1}{\sim} y$. The steps 6 – 11 in Algorithm 1 put x, y into a class X' of $\Re$. Hence $X \subseteq Y \subseteq X'$. Note that the intersection of any two different classes of a classification must be empty by definition. Thus $X = X'$ since $X \cap X' \neq \emptyset$. Therefore, $X = Y$. In other words, every class of $\Re$ is a class of $\Re_1$. Similarly, it can be shown that every class of $\Re_1$ is a class of $\Re$. Therefore, $\Re$ is the same as $\Re_1$. In other words, the claim is true for $m = 1$.

Case 2. Assume the claim is true for $m = k - 1$. Consider $m = k$.

(I) After the k-th loop 3 – 13 is executed, every class X of the current $\Re$ is contained in a class Y of $\Re_k$ by Lemmas 1 and 2, i.e., $X \subseteq Y$.

Now let us show that $X \supseteq Y$. If there exists $y \in Y - X$, $y \overset{*}{\sim}_{\Gamma_k} x$ for $x \in X$ since $X \subseteq Y$. Then there exist $y' \in Y - X$ and $x' \in X$ such that $y' \overset{w}{\sim} x'$ for some $w \in \Gamma_k$. If $w = w_k$, the class U of $\Re_{k-1}$ containing y' and the class V of $\Re_{k-1}$ containing x' are merged into one class of $\Re$ by the steps 6 – 11 in Algorithm 1 after the k-th loop. Note that U, V are also classes of $\Re$ after the $(k-1)$-th loop by the induction assumption. Since $x' \in X$, y' is also in X. This contradicts $y' \in Y - X$. If $w \neq w_k$, x', y' are both in a class U of $\Re_{k-1}$ by definition. U is also a class of $\Re$ after the $(k-1)$-th loop by the induction assumption. Since Algorithm 1 never splits a class and $x' \in X$, $U \subseteq X$. Thus $y' \in X$. This contradicts $y' \in Y - X$. Hence $Y - X = \emptyset$. In other words, $Y \subseteq X$.

Since $X \subseteq Y$ and $Y \subseteq X$, $X = Y$; that is, every class of $\Re$ is a class of $\Re_k$.

(II) Let us now show that every class Y of $\Re_k$ is a class of $\Re$ after the k-th loop. By Lemma 3, Y is the union of one or more classes $X_1, X_2, \cdots, X_n$ $(n \geq 1)$ of $\Re_{k-1}$. For any class Z of $\Re_{k-1}$ satisfying $Z \cap Y = \emptyset$, there is no $z \in Z$ such that $z \overset{w_k}{\sim} x$ for $x \in X_i$ $(1 \leq i \leq n)$. Otherwise, $Z \subseteq Y$ by Lemma 3. If $n = 1$, clearly, $Y = X_1$ is also a class of $\Re$ after the k-th loop. If $n \geq 2$, for any two X_i, X_j $(1 \leq i, j \leq n)$, either there exist $x_i \in X_i$ and $x_j \in X_j$ such that $x_i \overset{w_k}{\sim} x_j$, or there exist $x_i \in X_i$, $x_s \in X_s$, $\cdots$, $x_t \in X_t$ and $x_j \in X_j$ $(1 \leq s, t \leq n)$ such that $x_i \overset{w_k}{\sim} x_s, \cdots, x_t \overset{w_k}{\sim} x_j$. Otherwise, X_i, X_j cannot be both contained in Y. By the induction assumption, all X_i $(1 \leq i \leq n)$ are also classes of $\Re$ after the $(k-1)$-th loop. Clearly, the steps 6 – 11 will merge all X_i $(1 \leq i \leq n)$ into a class of $\Re$ after the k-th loop and no other class of $\Re_{k-1}$ will be merged into the class. In other words, Y is a class of $\Re$ after the k-th loop. Hence every class of $\Re_k$ is a class of $\Re$ after the k-th loop.

From (I) and (II), it is shown that $\Re_k$ is the same as the current $\Re$ after the k-th loop.

From Cases 1 and 2, it is concluded, by the mathematical induction, that Algorithm 1 returns the correct classification. □

Example 6. A set G of queries contains the following queries:

$$
\begin{aligned}
Q_1 :&\quad \pi_{a_1,a_2}\ (\ \sigma_{a_1>3}(R_1)\ ),\\
Q_2 :&\quad \pi_{a_1,a_2}\ (\ \sigma_{a_2>6}(R_1)\ ),\\
Q_3 :&\quad \pi_{a_1,a_2,a_3}\ (\ \sigma_{a_2\geq 6}(R_1)\ ),\\
Q_4 :&\quad \pi_{a_1,a_2}\ (\ \sigma_{a_2\geq 6}(R_1)\ ),\\
Q_5 :&\quad \sigma_{a_3=3\ \wedge\ a_1=3}(R_1),\\
Q_6 :&\quad \sigma_{a_2=4\ \wedge\ (a_1<4\ \vee\ a_3\geq 3)}(R_1),\\
Q_7 :&\quad \pi_{a_1,a_2,a_3}\ (R_1),
\end{aligned}
$$

where a_1, a_2 are clustered-indexed columns of R_1, a_3 is a sequential column of R_1. Let $\Gamma = \{r_1, r_5, r_4, r_8, r_7\}$ be the set of valid classification rules. Use Algorithm 1 to get the classification $\Re$ for G with respect to Γ.

Initially,

$$\Re \;=\; \{\ \{Q_1\},\ \{Q_2\},\ \{Q_3\},\ \{Q_4\},\ \{Q_5\},\ \{Q_6\},\ \{Q_7\}\ \}.$$

From $r_1 \in \Gamma$,

$$\Re \;=\; \{\ \{Q_1,\ Q_2\},\ \{Q_3\},\ \{Q_4\},\ \{Q_5\},\ \{Q_6\},\ \{Q_7\}\ \}.$$

From $r_5 \in \Gamma$,

$$\Re \;=\; \{\ \{Q_1,\ Q_2\},\ \{Q_3,\ Q_4\},\ \{Q_5\},\ \{Q_6\},\ \{Q_7\}\ \}.$$

From $r_4 \in \Gamma$,

$$\Re \;=\; \{\ \{Q_1,\ Q_2,\ Q_3,\ Q_4\},\ \{Q_5\},\ \{Q_6\},\ \{Q_7\}\ \}.$$

From $r_8 \in \Gamma$,

$$\Re \;=\; \{\ \{Q_1,\ Q_2,\ Q_3,\ Q_4\},\ \{Q_5\},\ \{Q_6\},\ \{Q_7\}\ \}.$$

From $r_7 \in \Gamma$,

$$\Re \;=\; \{\ \{Q_1,\ Q_2,\ Q_3,\ Q_4\},\ \{Q_5,\ Q_6\},\ \{Q_7\}\ \}.$$

The final $\Re$ is the expected classification. A local DBMS most likely chooses a clustered-index scan method with a range, a clustered-index scan method with a key value and a sequential method for classes $\{Q_1,\ Q_2,\ Q_3,\ Q_4\}$, $\{Q_5,\ Q_6\}$ and $\{Q_7\}$, respectively. From the above procedure we can see that r_8 is redundant for classifying the query set G. In fact, r_8 is weaker than any other classification rule in Γ for G.

If we put r_9 into Γ, we would still get the same final classification. However, r_1, r_4 and r_5 would become redundant for classifying the query set G; i.e., they could be removed from Γ. $\square$

3.4. Redundant Classification Rules

Even if a classification rule is not weaker than any other classification rule in a given set of valid classification rules, it may still be useless for classifying a query set. The reason for this is that it may be weaker than a composite rule composed from several other classification rules in the set of classification rules.

<u>Definition</u> 10. Given a set Γ of classification rules for a set G of queries, a classification rule $r \in \Gamma$ is redundant if there exists a non-empty subset $\Gamma' \subset \Gamma$ excluding r such that, if $Q_1 \overset{r}{\sim} Q_2$, then $Q_1 \overset{*}{\sim}_{\Gamma'} Q_2$, for any Q_1, Q_2 in G; i.e., $\overset{r}{\sim}$ is weaker than the composite relation $\overset{*}{\sim}_{\Gamma'}$. In this case, we also say that the relation $\overset{r}{\sim}$ is redundant.

Clearly, Definition 3 is a special case of Definition 10 when $\Gamma = \{r_i,\ r_j\}$.

Consider the set of all classification rules $\Gamma = \{r_1,\ r_4,\ r_5,\ r_6,\ r_7,\ \cdots,\ r_{15}\}$ that we have discussed. The following theorem holds:

<u>THEOREM</u> 2. For the set G of all unary and join queries, classification rules r_1, r_4, r_5 and r_6 in Γ are redundant.

PROOF. Let G_1 be the subset of all unary queries in G, and G_2 be the subset of all join queries in G. Obviously, $G = G_1 \cup G_2$, and $G_1 \cap G_2 = \emptyset$.

Let us follow the loop 3 – 13 in Algorithm 1. Consider the classification rules $r_7, r_8, \cdots, r_{11}$ first.

After r_7 is applied in the loop, $\Re$ contains a class of queries:

$$\begin{aligned} G_{1\ 1} \quad = \quad & \{\ \textit{unary queries whose qualifications have at least one conjunct} \\ & R_i.a_n = C \textit{ where } R_i.a_n \textit{ is clustered} - \textit{indexed}\ \}. \end{aligned} \tag{3.3}$$

Any query that is not in $G_{1\ 1}$ comprises a separate class of $\Re$. As mentioned, $G_{1\ 1}$ is called a major class under r_7. Similarly, r_8 also generates a major class under it:

$$\begin{aligned} G_{1\ 2} \quad = \quad & \{\ \textit{unary queries whose qualifications have at least one conjunct} \\ & R_i.a_n = C \textit{ where } R_i.a_n \textit{ is indexed}\ \} \ - \ G_{1\ 1}. \end{aligned} \tag{3.4}$$

Clearly, $G_{1\ 1}$ and $G_{1\ 2}$ are disjoint. Similarly, $r_9, \cdots, r_{11}$ generate the following major classes under them, respectively:

$$\begin{aligned} G_{1\ 3} \quad = \quad & \{\ \textit{unary queries whose qualifications have at least one conjunct} \\ & R_i.a_n\ \theta\ C \textit{ where } R_i.a_n \textit{ is clustered} - \textit{indexed}\ \} \ - \ G_{1\ 1} \\ & - \ G_{1\ 2}, \end{aligned} \tag{3.5}$$

$$\begin{aligned} G_{1\ 4} \quad = \quad & \{\ \textit{unary queries whose qualifications have at least one conjunct} \\ & R_i.a_n\ \theta\ C \textit{ where } R_i.a_n \textit{ is indexed}\ \} \ - \ G_{1\ 1} \ - \ G_{1\ 2} \\ & - \ G_{1\ 3}, \end{aligned} \tag{3.6}$$

$$G_{1\ 5} \quad = \quad G_1 \ - \ G_{1\ 1} \ - \ G_{1\ 2} \ - \ G_{1\ 3} \ - \ G_{1\ 4}, \tag{3.7}$$

where $\theta \ \in \ \{<, \ >, \ \leq, \ \geq\}$. Clearly, $G_{1\ 1}, \ \cdots, \ G_{1\ 5}$ are disjoint with each other, and

$$G_{1\ 1} \ \cup \ G_{1\ 2} \ \cup \ \cdots \ \cup \ G_{1\ 5} \ = \ G_1. \tag{3.8}$$

Similarly, $r_{12}, \ \cdots, \ r_{15}$ generate the following major classes under them, respectively:

$$\begin{aligned} G_{2\ 1} \quad = \quad & \{\ \textit{join queries whose qualifications have at least one conjunct} \\ & R_i.a_n = R_j.a_m \textit{ where either } R_i.a_n \textit{ or } R_j.a_m \textit{ (or both) is} \\ & \textit{clustered} - \textit{indexed}\ \}, \end{aligned} \tag{3.9}$$

$$\begin{aligned} G_{2\ 2} \quad = \quad & \{\ \textit{join queries whose qualifications have at least one conjunct} \\ & R_i.a_n = R_j.a_m \textit{ where either } R_i.a_n \textit{ or } R_j.a_m \textit{ (or both) is indexed}\ \} \\ & - \ G_{2\ 1}, \end{aligned} \tag{3.10}$$

$$\begin{aligned} G_{2\ 3} &= \{\, join\ queries\ whose\ qualifications\ have\ at\ least\ one\ index-usable \\ &\qquad conjunct\ for\ at\ least\ one\ operand\ table\,\} - G_{2\ 1} - G_{2\ 2}, \qquad (3.11) \\ G_{2\ 4} &= G_2 - G_{2\ 1} - G_{2\ 2} - G_{2\ 3}. \qquad (3.12) \end{aligned}$$

Clearly, $G_{2\ 1} - G_{2\ 4}$ are disjoint with each other, and

$$G_{2_1} \cup G_{2\ 2} \cup \cdots \cup G_{2\ 4} = G_2. \qquad (3.13)$$

Therefore, classes $G_{1\ 1} - G_{2\ 4}$ form a classification of the query set G, which is obtained by applying the classification rules $r_7,\ r_8,\ \cdots,\ r_{15}$ in Algorithm 1.

For any class $G_{i\ j}$ ($1 \le j \le 5$ for $i = 1$; $1 \le j \le 4$ for $i = 2$) and any query $q \in G_{i\ j}$, if there exists another query $q' \in G$ such that $q' \overset{r_1}{\sim} q$, q' is also in the $G_{i\ j}$, because changing constants and/or tables and/or columns with the same physical property in q does not change its membership in $G_{i\ j}$. In other words, for any $q',\ q'' \in G$, if $q' \overset{r_1}{\sim} q''$, both q' and q'' belong to the same $G_{i\ j}$; that is, $q' \overset{r_{5*i+j+1}}{\sim} q''$. Therefore, r_1 is redundant.

Similarly, changing the projection list or using other similar predicates or changing the order of join tables in a query does not change its membership in a class. Therefore, r_4, r_5 and r_6 are redundant. □

Theorem 2 indicates that the elementary classification rules r_1, r_4, r_5 and r_6 are implied by the common classification rules r_7, r_8, $\cdots$, r_{15}.

The proof of Theorem 2 also leads to the following corollary:

<u>Corollary</u> 2. Let $\Gamma_c = \{r_7,\ r_8,\ \cdots,\ r_{15}\}$. The classification induced by the composite relation $\sim^*_{\Gamma_c}$ is $\Re_c = \{G_{1\ 1},\ \cdots,\ G_{1\ 5},\ G_{2\ 1},\ \cdots,\ G_{2\ 4}\}$, where $G_{1\ 1},\ \cdots,\ G_{1\ 5}, G_{2\ 1},\ \cdots,\ G_{2\ 4}$ are specified in (3.3) – (3.12), respectively.

Classification $\Re_c$ is called a common classification, because it is applicable for many local DBMSs. A query class in $\Re_c$ is called a common query class.

4. Top-down Classification Approach

The bottom-up classification approach discussed in Section 3 provides a theoretical insight into the query classification problem. However, since the number of queries in a given query set is usually huge, it may not be practical to start with each query as a separate class. In contrast, the top-down classification approach is more feasible in practice because it is easy to start and the refinement can stop at any stage, depending on the expectation of estimation accuracy.

The idea of the top-down classification approach is as follows. The initial classification consists of only one class that is the whole set of given queries. Based on available information and policies for choosing access methods for queries in the underlying local DBMS, the classification can be refined by dividing the class into subclasses. The subclasses can be further divided into smaller subclasses if more information is available. This procedure can be repeated until a satisfactory classification is achieved. Usually, the smaller the classes are, the better the cost estimation formulas can be derived. However, refining a classification would require more information, and the classification with too many classes should be avoided in order to reduce the maintenance overhead for the cost formulas.

4.1. Classification Procedure

Let us describe the top-down classification procedure in more details. The following policy is obviously valid on any local DBMS:

r_{16}: *a unary query and a join query are executed by using different access methods.*

Hence, the sole class G in the initial classification can be divided into two smaller classes:

$$G \;=\; G_1 \;\cup\; G_2, \tag{4.14}$$

where $G_1 = \{unary\ queries\}$, $G_2 = \{join\ queries\}$. This classification is based on policy r_{16} and the syntax information about the queries.

If we know more policies for choosing access methods for queries in a local DBMS, we can refine the classification (4.14). For example, if we know the common policy r_7 in Table 2 is valid, we can divide G_1 into two smaller classes:

$$G_1 \;=\; G_{1\ 1} \;\cup\; G_1', \tag{4.15}$$

where $G_{1\ 1}$ is specified in (3.3), G_1' contains the queries in G_1 that are not in $G_{1\ 1}$. If we know that the common policies $r_8 - r_{11}$ in Table 2 are valid as well, we can further divide G_1' into smaller classes:

$$G_1' \;=\; G_{1\ 2} \;\cup\; G_{1\ 3} \;\cup\; \cdots \;\cup\; G_{1\ 5}, \tag{4.16}$$

where $G_{1\ 2},\ \cdots,\ G_{1\ 5}$ are specified in (3.4) – (3.7), respectively.

Similarly, if we know that the common policies $r_{12},\ r_{13},\ \cdots,\ r_{15}$ in Table 3 are valid for the local DBMS, we can divide G_2 into smaller classes:

$$G_2 \;=\; G_{2\ 1} \;\cup\; G_{2\ 2} \;\cup\; \cdots \;\cup\; G_{2\ 4}, \tag{4.17}$$

where $G_{2\ 1},\ G_{2\ 2},\ \cdots,\ G_{2\ 4}$ are specified in (3.9) – (3.12), respectively.

In principle, any of classes $G_{1\ i}, G_{2\ j}$ $(1 \le i \le 5; 1 \le j \le 4)$ can be further divided into smaller classes if more information is available (see Fig. 4).

For example, in some DBMSs, such as ORACLE 7.0, an access method via concatenating indexes may be supported to make use of an index-usable conjunct, such as $R_1.a_1 < 10\ \vee\ R_1.a_2 = 15$ where $R_1.a_1$ and $R_1.a_2$ are indexed for a query in $G_{1\ 5}$. Then the class $G_{1\ 5}$ can be further divided into two smaller classes — one $G_{1\ 5}'$ contains queries having one or more index-usable conjuncts and the other $G_{1\ 5}''$ contains queries without such conjuncts.

As another example, any class can be divided into smaller classes according to the (estimated) sizes of query result tables (or operand tables) because a DBMS may

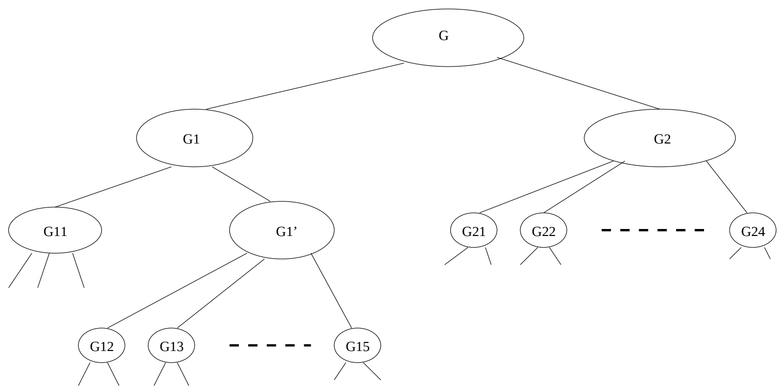

Fig. 4. Classifying Queries in Top-down Approach

adopt different processing and buffering strategies to handle queries with different result sizes.

Also, the query classes can be refined according to the data type(s) of referenced column(s), because the same access method may behave differently for different data types in term of performance.

In addition, a join query class can be refined according to whether there is a joining column pair which satisfies a referential constraint.

The classification of queries may vary from one DBMS or application to another. In general, a refined classification is expected to yield better cost formulas because its query classes are usually more homogeneous in terms of performance. However, as mentioned before, the overhead of maintaining the cost parameters (in cost formulas) grows as the number of query classes increases. A trade-off between estimation accuracy and overhead is required.

After a classification is given, a cost estimation formula can be derived for each class based on the observed costs of sample queries by using multiple regression.[21,23] In practice, the classification procedure and the cost formula derivation may need to be iterated several times before satisfactory cost formulas can be obtained.

Some local DBMSs in an MDBS may retain strong local autonomy in the sense that no much information is available at the global level for query classification. In this case, the assumed common policies and some estimated information can be used to classify queries. As we pointed out before, the goodness of derived cost formulas depends on the degree of agreement between the assumed/estimated information and the actual information in the local DBMS. Some warnings may be issued to the global query optimizer to alert this case. The global query optimizer should use such derived cost formulas with caution during query optimization.

4.2. Relationship between Bottom-up and Top-down Approaches

When a common policy is used to classify a query set in the bottom-up classification approach in Section 3, one assumption is made; that is, each query that is not qualified for the common policy comprises a separate class on its own. For example, the common policy r_7 states that all queries in $G_{1\ 1}$ (major class) specified in (3.3) employ the clustered-index scan method with a key value. It is assumed that any query that is not in $G_{1\ 1}$ comprises a separate class on its own when the relevant relation $\overset{r_7}{\sim}$ and classification rule r_7 are defined. However, the top-down classification approach makes a different assumption; that is, all queries that are not in the major class described by a common policy are assumed to be all in another class. Therefore, the relevant equivalence relation induced by the common policy r_7 in the top-down classification approach is defined as follows:

<u>Definition</u> 11. Let Q_1 and Q_2 be two queries in the query set G. If they both are in the set $G_{1\ 1}$ in (3.3), or neither is in $G_{1\ 1}$, Q_1 and Q_2 are said to be related to each other via a relation $\overset{r_7^+}{\sim}$, denoted as $Q_1 \overset{r_7^+}{\sim} Q_2$, or $Q_2 \overset{r_7^+}{\sim} Q_1$.

The induced classification rule is:

<u>Classification Rule</u> r_7^+: *Two queries Q_1 and Q_2 are in the same class if $Q_1 \overset{r_7^+}{\sim} Q_2$.*

Using this classification rule to classify G, one can get two classes: $G_{1\ 1}$ and $G - G_{1\ 1}$. For the other common policies, we can define similar equivalence relations and classification rules.

When multiple classification rules are applied to classify queries in the bottom-up approach, the OR logic is used to compose them; that is, two queries are in the same class if they are directly or indirectly related to each other via any given classification rule(s). However, when multiple classification rules are applied to classify queries in the top-down approach, the AND logic is used to compose the rules. For example, assume the following classification rule corresponding to the common policy r_{16} is given:

<u>Classification Rule</u> r_{16}^+: *Two queries Q_1 and Q_2 are in the same class if they both are unary queries or they both are join queries.*

If one wants to apply the classification rules r_7^+ and r_{16}^+ to classify the given query set G, two queries are put in the same class if and only if they are in the same class under both r_7^+ and r_{16}^+. Hence the obtained classification consists of classes $G_{1\ 1}$, G'_1 in (4.15) and G_2 in (4.14).

In general,

<u>Definition</u> 12. Let Q' and Q'' be two queries in G, and Γ be a non-empty set of valid classification rules. Q'' is said to be related to Q' w.r.t. Γ via a relation $\overset{+}{\sim}_\Gamma$ if for every $x \in \Gamma$ it is true that $Q'' \overset{x}{\sim} Q'$, denoted as $Q'' \overset{+}{\sim}_\Gamma Q'$.

It is easy to show that $\overset{+}{\sim}_\Gamma$ is an equivalence relation. Its induced classification rule is:

<u>Composite Classification Rule</u> $+_\Gamma$: *For a given set of valid classification rules* Γ*, two queries* Q_1 *and* Q_2 *are in the same class w.r.t.* Γ *if* $Q_1 \overset{+}{\sim}_\Gamma Q_2$.

A top-down classification algorithm similar to Algorithm 1 can also be given (omitted).

4.3. Hybrid Classification Approach

Note that although the bottom-up approach helps to understand the theoretical foundations of query classification, it may not be practical since there may be too many classes to start with. The top-down approach, on the other hand, is more practical. However, it may not be most efficient since the obtained first couple of initial classifications with a few classes are usually not satisfactory. A more efficient approach is the hybrid one that combines the top-down and bottom-up approaches. The idea is to start with a guessed promising initial classification, then refining some of its classes (top-down) as well as merging some others (bottom-up) to improve the classification until a satisfactory classification is obtained.

5. Membership Testing Method Based on Ranks

An issue related to query classification is how to test the membership of a given query. In order to use the cost formula for a query class to estimate the cost of a query, we need to test if the query belongs to the query class. Theoretically speaking, once a query class is generated, the membership of a query in the class can be tested by comparing the query with the queries in the class. However, this approach may not be efficient in practice because the cardinality of the class can be very large. In practice, the membership of a query in a class can be tested by checking whether the query has the characteristics of the queries in the class if such characteristics have been identified. For example, whether a query is in the common query class $G_{1\ 1}$ can be tested by checking if the query has at least one conjunct $R_i.a_n = C$ in its qualification where $R_i.a_n$ is clustered-indexed.

In this section, we introduce another efficient membership testing method that can be easily implemented in practice. The idea is to assign ranks to the predicates in queries and calculate the ranks of the qualifications in such a way that only the queries in the same query class have the same rank for their qualifications. Hence the rank of the qualification of a query indicates the class to which it belongs. Let us illustrate the method by considering the common query classes in Section 3.4.

Let $F = T_1 \wedge T_2 \wedge \cdots \wedge T_n$ $(n \geq 1)$ be the qualification of a query. Each conjunct T_i $(1 \leq i \leq n)$ can be a basic predicate or several basic predicates connected by $\vee$'s. If F is '*true*', a dummy basic predicate $R_i.a_n$ *nil* C is used.

5.1. Ranks for Unary Query Predicates and Qualifications

Let us consider unary queries first. Ranks are assigned to different basic unary query predicates as shown in Table 4. If the qualification F contains some logic

Table 4. Ranks for Unary Query Predicates

rank \ physical property of R.a / predicate	clustered-indexed	indexed	sequential
R.a = C	1	2	5
R.a < C	3	4	5
R.a <= C	3	4	5
R.a > C	3	4	5
R.a >= C	3	4	5
R.a != C	5	5	5
R.a nil C ('true')	5	5	5

R -- table a -- column C -- constant

connectives $\wedge$'s and $\vee$'s, the following formulas are used to calculate its rank:

$$rank(P_1 \ \vee \ P_2 \ \ \vee \ \cdots \ \vee \ P_m) \ = \ 5, \qquad (m \geq 2) \tag{5.18}$$

and

$$\begin{aligned} & rank(T_1 \ \wedge \ T_2 \ \wedge \ \cdots \ \wedge \ T_n) \\ & \quad = \ min\{rank(T_1), \ rank(T_2), \ \cdots, \ rank(T_n)\}, \end{aligned} \tag{5.19}$$

where P_i $(1 \leq i \leq m)$ is a basic predicate and $rank(X)$ denotes the rank of X.

In fact, each rank corresponds to one type of unary access method as follows:

rank 1: clustered-index scan method with a key value,
rank 2: index scan method with a key value,
rank 3: clustered-index scan method with a range,
rank 4: index scan method with a range,
rank 5: sequential scan method.

Usually, the lower the rank is, the more efficient the corresponding access method. Therefore, if two access methods can be used to execute a query, a DBMS usually chooses the one with a lower rank. This fact is reflected in formula (5.19). That is, if different access methods can be applied to different conjuncts of the qualification of a query, the best access method (with the smallest rank) is chosen for the whole query.

Although it is possible to use a better access method for $P_1 \ \vee \ P_2 \ \vee \ \cdots \ \vee \ P_n$, such as an index-based method, than the sequential scan method, improvement of efficiency is usually small. Hence many DBMSs simply applies the sequential scan method to it. This fact is reflected in the formula (5.18).

Example 7. Consider the following query qualifications:

$$F_1^{(1)}: \quad R_1.a_1 = 3 \ \wedge \ R_1.a_2 > 5 \ \wedge \ R_1.a_3 \leq 7,$$

$$\begin{aligned} F_2^{(1)}: &\quad R_1.a_1 \geq 8 \ \wedge \ (R_1.a_2 = 6 \ \vee \ R_1.a_3 < 9), \\ F_3^{(1)}: &\quad R_1.a_1 = 3 \ \wedge \ R_1.a_2 = 5 \ \wedge \ R_1.a_3 < 6, \\ F_4^{(1)}: &\quad R_1.a_3 < 7 \ \wedge \ (R_1.a_1 < 5 \ \vee R_1.a_2 > 9), \end{aligned}$$

where $R_1.a_1$, $R_1.a_2$ and $R_1.a_3$ are indexed, clustered-indexed and sequential, respectively. The ranks and key conjuncts for the above qualifications are as follows:

$$\begin{aligned} rank(F_1^{(1)}) &= min\{rank(R_1.a_1 = 3),\ rank(R_1.a_2 > 5),\ rank(R_1.a_3 \leq 7)\} \\ &= min\{2,\ 3,\ 5\} = 2, \\ rank(F_2^{(1)}) &= min\{4,\ 5\} = 4, \\ rank(F_3^{(1)}) &= min\{2,\ 1,\ 5\} = 1, \\ rank(F_4^{(1)}) &= min\{5,\ 5\} = 5, \end{aligned}$$

In other words, the access methods used for the queries with the qualifications $F_1^{(1)}$ to $F_4^{(1)}$ by a local DBMS are most likely the index scan with a key value, the index scan with a range, the clustered-index scan with a key value, and the sequential scan, respectively. □

The ranks can be adjusted if the preferences of access methods are different and/or more access methods are supported in the underlying local DBMS.

5.2. Ranks for Join Query Predicates and Qualifications

Similarly, ranks can be assigned to basic predicates in the qualification of a join query, as shown in Tables 5 and 6. For the qualification F of a join query involving

Table 5. Ranks for Joining Predicates in Join Queries

rank of conjunct Ri.a= Rj.b / physical property of Ri.a / physical property of Rj.b	clustered-indexed	indexed	sequential
clustered-indexed	1	1	1
indexed	1	2	2
sequential	1	2	4
rank (non-conjunct Ri.a = Rj.b) = 4			
Ri, Rj -- tables, a, b -- columns			

Table 6. Ranks for Unary Predicates in Join Queries

rank / predicate / physical property of R.a	clustered-indexed	indexed	sequential
R.a = C	3	3	4
R.a < C	3	3	4
R.a <= C	3	3	4
R.a > C	3	3	4
R.a >= C	3	3	4
R.a != C	4	4	4
R.a nil C (‘true’)	4	4	4
R -- table a -- column C -- constant			

logic connectives, formula (5.19) still holds, but formula (5.18) becomes:

$$\begin{aligned} &rank(P_1 \ \vee \ P_2 \ \vee \ \cdots \ \vee \ P_m) \\ &\qquad = max\{rank(P_1),\ rank(P_2),\ \cdots,\ rank(P_m)\}, \end{aligned} \tag{5.20}$$

The join access methods and their associated ranks are as follows:

rank 1: clustered-index join method,
rank 2: index join method,
rank 3: nested-loop join method with reduction on operand table(s) via index(es),
rank 4: sort-merge join method.

The lower the rank is, the higher preference the corresponding access method has. As before, if a local DBMS supports more types of access methods, such as hash join, appropriate additional ranks can be added. The ranks may also be adjusted for a particular local DBMS if the preferences of its join access methods are different.

Example 8. Consider the following join query qualifications:

$$\begin{aligned}
F_1^{(1\ 2)}: &\quad R_1.a_1 = 6 \ \wedge\ R_2.a_2 > 4 \ \wedge\ R_1.a_1 = R_2.a_1, \\
F_2^{(1\ 2)}: &\quad R_1.a_2 = R_2.a_1 \ \wedge\ (R_2.a_1 = 9 \ \vee\ R_1.a_2 < 7), \\
F_3^{(1\ 2)}: &\quad R_1.a_2 = R_2.a_2 \ \wedge\ R_2.a_2 > 12, \\
F_4^{(1\ 2)}: &\quad R_1.a_2 = R_2.a_2 \ \wedge\ R_1.a_1 > 12 \ \wedge\ (R_2.a_1 = 5 \ \vee\ R_2.a_1 = 15), \\
F_5^{(1\ 2)}: &\quad R_1.a_2 = R_2.a_2 \ \wedge\ R_1.a_1 = R_2.a_2 \ \wedge\ R_2.a_1 \leq 2 \\
&\qquad \wedge\ (R_2.a_1 > 13 \ \vee\ R_2.a_1 = R_1.a_2),
\end{aligned}$$

where $R_1.a_1$ and $R_2.a_1$ are clustered-indexed and indexed, respectively, and $R_1.a_2$ and $R_2.a_2$ are sequential. The ranks for the qualifications are as follows:

$$\begin{aligned}
rank(F_1^{(1\ 2)}) &= min\{3,\ 4,\ 1\} = 1, \\
rank(F_2^{(1\ 2)}) &= min\{2,\ max\{3,\ 4\}\ \} = \{2,\ 4\} = 2, \\
rank(F_3^{(1\ 2)}) &= min\{4,\ 4\} = 4, \\
rank(F_4^{(1\ 2)}) &= min\{4,\ 3,\ max\{3,\ 3\}\ \} = min\{4,\ 3,\ 3\} = 3, \\
rank(F_5^{(1\ 2)}) &= min\{4,\ 1,\ 3,\ max\{3,\ 4\}\ \} = min\{4,\ 1,\ 3,\ 4\} = 1,
\end{aligned}$$

□

5.3. Membership Testing Algorithm

Since the rank of the qualification of a query tells us which access method is most likely to be used, two queries can be in the same query class if they have the same rank for their qualifications.

Definition 13. Let Q_1 and Q_2 be two unary queries or two join queries. If the rank of the qualification of Q_2 is the same as that of the qualification of Q_1, we say that Q_2 is related to Q_1 via a relation $\stackrel{q}{\sim}$, denoted as $Q_2 \stackrel{q}{\sim} Q_1$.

Clearly, $\stackrel{q}{\sim}$ is an equivalence relation. The induced classification rule is as follows:

<u>CLASSIFICATION RULE</u> q: *Two queries Q_1 and Q_2 are in the same class if $Q_1 \overset{q}{\sim} Q_2$.*

In fact, the resulting classification contains the following classes:

$$G_{1\ s} \quad = \quad \{\ \pi_{\alpha^{(i)}}(\sigma_{_{F^{(i)}}}(R_i)) \mid rank(F^{(i)}) = s\ \} \qquad (s = 1, 2, 3, 4, 5), \qquad (5.21)$$

$$G_{2\ t} \quad = \quad \{\ \pi_{\alpha^{(i\ j)}}(R_i \underset{F^{(i\ j)}}{\bowtie} R_j) \mid rank(F^{(i\ j)}) = t\ \} \quad (t = 1, 2, 3, 4). \quad (5.22)$$

We can show that

<u>THEOREM</u> 3. The query classes $G_{1\ s}$ $(s = 1, 2, 3, 4, 5)$ and $G_{2\ t}$ $(t = 1, 2, 3, 4)$ in (5.21) and (5.22) are the common query classes specified in (3.3) – (3.12).

PROOF. Consider the query class $G_{1\ 1}$ in (5.21). Since the rank of the qualification of any query in $G_{1\ 1}$ is 1, there must be at least one conjunct $R_i.a_n = C$ in the query, where $R_i.a_n$ is clustered-indexed, from observing Table 4 and the formulas (5.18) and (5.19). Conversely, the rank of the qualification of a query in $G_{1\ 1}$ specified in (3.3) is 1. Hence $G_{1\ 1}$ is the same as the one specified in (3.3).

For $G_{1\ 2}$, since the rank of the qualification of any query in the class is 2, there should not be any conjunct in the query whose rank is 1, and there must be at least one conjunct $R_i.a_n = C$ in each query where $R_i.a_n$ is indexed, from observing Table 4 and the formulas (5.18) and (5.19). Conversely, the rank of the qualification of a query in $G_{1\ 2}$ specified in (3.4) is 2. Hence $G_{1\ 2}$ is the same as the one specified in (3.4).

Similarly, it can be shown that $G_{1\ 3}$ – $G_{1\ 5}$ are the same as the ones specified in (3.5) – (3.7), respectively.

For $G_{2\ 1}$, since the rank of the qualification of any query in the class is 1, there must be at least one conjunct $R_i.a_n = R_j.a_m$ in each query, where either $R_i.a_n$ or $R_j.a_m$ or both are clustered-indexed, from observing Tables 5 and 6 and formulas (5.19) and (5.20). Conversely, the rank of the qualification of a query in $G_{2\ 1}$ specified in (3.9) is 1. Hence $G_{2\ 1}$ is the same as the one specified in (3.9).

Similarly, $G_{2\ 2}$ – $G_{2\ 4}$ are the same as the ones specified in (3.10) – (3.12). □

We therefore have the following simple algorithm to test which class a given query belongs to.

<u>ALGORITHM</u> 2. : Testing query membership
Input: A query Q
Output: The common query class to which Q belongs
Method:

1. **begin**
2. **if** Q is a unary query **then**
3. calculate the rank k of the qualification of Q
4. by using Table 4 and formulas (5.18) and (5.19);
5. **return** $G_{1\ k}$;
6. **else**

```
        calculate the rank k of the qualification of Q
            by using Tables 5 and 6 and formulas (5.19) and (5.20);
        return G_{2 k};
    end;
end.
```

6. Experiments

Experiments were conducted on an MDBS prototype[2] with local DBSs supported by three commercial DBMSs: ORACLE 7.0, DB2/6000 1.1.0 and EMPRESS 4.6. A local database that contains 12 tables with randomly-generated data was created for each local DBS. The sizes of tables range from 300 to 25,000 tuples. Indexes were created on some columns for the tables in each local database. Clustered indexes were also created for some tables in the local database managed by ORACLE 7.0.

Using either the bottom-up classification approach with classification rules $r_7, r_8, \cdots, r_{15}, *_\Gamma$, or the top-down classification approach with classification rules $r_7^+, r_8^+, \cdots, r_{15}^+, r_{16}^+, +_\Gamma$, we can obtain common query classes $G_{1\ 1}$, $G_{1\ 2}$, $G_{1\ 3}$, $G_{1\ 4}$, $G_{1\ 5}$, $G_{2\ 1}$, $G_{2\ 2}$, $G_{2\ 3}$ and $G_{2\ 4}$ for the local DBS using ORACLE and common query classes $G_{1\ 2}$, $G_{1\ 4}$, $G_{1\ 5}$, $G_{2\ 2}$, $G_{2\ 3}$ and $G_{2\ 4}$ for the local DBSs using DB2/6000 and EMPRESS. These classifications were used in our experiments.

Based on the query sampling method, a sample of queries were drawn from each query class and a cost formula was derived for each query class by performing regression analysis on the observed costs of the sample queries[g]. For example, sample queries drawn from query class $G_{1\ 4}$ are of the following form:

$$\pi_{\alpha^{(i)}}(\sigma_{R_i.a_n\ \theta\ C_1^{(i.a_n)}\ \wedge\ R_i.a_m\ \omega\ C^{(i.a_m)}}(R_i))$$

where $R_i.a_n$ is a randomly-selected indexed column in R_i; $R_i.a_m$ is a randomly-selected column in R_i; $\theta \in \{<, >\}$; $\omega \in \{=, \neq, <, >\}$ subject to (1) ω cannot be "$=$" if $R_i.a_m$ is indexed and (2) ω can only be "$\neq$" if $R_i.a_m$ is clustered-indexed; $\alpha^{(i)}$, $C_1^{(i.a_n)}$, $C^{(i.a_m)}$ are randomly-selected project list, constant in the domain of $R_i.a_n$ and constant in the domain of $R_i.a_m$, respectively. The cost formula derived for $G_{1\ 4}$ on ORACLE 7.0 is as follows:

$$0.354301 \ + \ 0.105255e-2 * TN \ + \ 0.32336e-2 * RN \ + \ 0.852187e-4 * RZ$$

where TN is the cardinality of the operand table; RN is the cardinality of the result table; and RZ is the result table length, i.e., the product of the tuple length and the cardinality of the result table. The details for drawing sampling queries from and deriving cost formulas for a query class can be found in [21] and [22], respectively. Statistical measures such as coefficient of multiple determination, standard error of

[g] The cost for the first execution of each query was used in the experiments. The cost models could also be built based on observed costs for the repeated (e.g., 2nd) execution of queries to take the cache effect into consideration. However, most applications would not require to execute the same query repeatedly at one time.

estimation and F-test were used to check the significance of derived cost formulas. Table 7 lists the statistical measures for the cost formulas we derived for different query classes. From the table, we can see that:

Table 7. Statistical Measures for Cost Formulas

local DBMS	query class	F-statistic (critical value at $\alpha = 0.01$)	coefficient of multiple determination	standard error of estimation (S)	average observed cost (A)	S/A %
ORA-CLE	$G_{1\ 1}$	204.06 (> 4.88)	0.83610	0.43747e-1	0.16869	25.9%
	$G_{1\ 2}$	56.76 (> 3.97)	0.65675	0.10578	0.20406	51.8%
	$G_{1\ 3}$	1576.66 (> 4.89)	0.97647	0.17882e+1	0.81627e+1	21.9%
	$G_{1\ 4}$	1161.46 (> 4.29)	0.96751	0.27357e+1	0.11360e+2	24.1%
	$G_{1\ 5}$	15397.70 (> 3.97)	0.99810	0.87345	0.13595e+2	6.4%
	$G_{2\ 1}$	4884.95 (> 4.27)	0.99201	0.34305e+1	0.18630e+2	18.4%
	$G_{2\ 2}$	3732.28 (> 4.28)	0.98992	0.14961e+1	0.60868e+1	24.6%
	$G_{2\ 3}$	1483.19 (> 7.06)	0.92457	0.51609e+3	0.75323e+3	68.5%
	$G_{2\ 4}$	980.69 (> 3.52)	0.97670	0.15275e+1	0.71334e+1	21.4%
EM-PRESS	$G_{1\ 2}$	183.20 (> 4.78)	0.77561	0.90392e-2	0.13729	6.6%
	$G_{1\ 4}$	381.49 (> 2.16)	0.91213	0.36670	0.10986e+1	33.3%
	$G_{1\ 5}$	34699.02 (> 3.98)	0.99917	0.57877e-1	0.21943e+1	2.6%
	$G_{2\ 2}$	2433.78 (> 3.83)	0.98833	0.14592e+1	0.62054e+1	23.3%
	$G_{2\ 3}$	37250.82 (> 4.72)	0.99847	0.40477e+2	0.32979e+3	12.3%
	$G_{2\ 4}$	110669.7 (> 7.08)	0.99966	0.32884e+2	0.64514e+3	5.1%
DB2 /6000	$G_{1\ 2}$	116.60 (> 3.64)	0.81481	0.40375e-1	0.10401	38.8%
	$G_{1\ 4}$	9472.52 (> 2.66)	0.99485	0.86129	0.90427e+1	9.5%
	$G_{1\ 5}$	63925.47 (> 3.97)	0.99954	0.44142	0.14863e+2	3.0%
	$G_{2\ 2}$	538.93 (> 4.29)	0.93252	0.43075e+1	0.93068e+1	46.3%
	$G_{2\ 3}$	334.32 (> 3.18)	0.94578	0.97540	0.33816e+1	28.8%
	$G_{2\ 4}$	552.43 (> 3.52)	0.95936	0.24401e+1	0.88202e+1	27.7%

- The statistical F-tests at the significance level $\alpha = 0.01$ show that all the derived cost formulas are useful for estimating the costs of local queries.

- Most cost formulas capture over 90% variability in query cost, as indicated by the coefficients of total determination. On the average, they capture about 93.5% variability.

- The standard errors of estimation for the cost formulas are acceptable, compared with the magnitudes of the relevant average observed costs of the sample queries. On the average, the standard error of estimation is only about 23.8% of the corresponding average observed cost.

These statistical measures show that the cost formulas derived by our query classification are quite good.

To further verify the usefulness of the derived cost formulas, we used them to estimate the costs of some randomly-generated test queries for each query class. For example, the test queries used for $G_{1\ 4}$ are of the following forms:

$$\pi_{\alpha(i)}(\sigma_{R_i.a_n\ \theta\ C^{(i.a_n)}}(R_i)) \ ,$$

$$\pi_{\alpha(i)}(\sigma_{R_i.a_n\ \theta\ C^{(i.a_n)}\ \wedge\ R_i.a_m\ \omega_1\ C^{(i.a_m)}}(R_i)) \ ,$$

$$\pi_{\alpha(i)}(\sigma_{R_i.a_n\ \theta\ C^{(i.a_n)}\ \wedge\ (\ R_i.a_p\ \omega_2\ C^{(i.a_p)}\ \vee\ R_i.a_q\ \omega_3\ C^{(i.a_p)})}(R_i)) \ ,$$

where $\omega_1,\ \omega_2,\ \omega_3\ \in\ \{\ =,\ \neq,\ <,\ \leq,\ >,\ \geq\ \}$; $\theta \in \{\ <,\ \leq,\ >,\ \geq\ \}$. The qualifications in the test queries satisfy the requirements of the corresponding query class; for example, $R_i.a_n$ is (non-clustered) indexed for $G_{1\ 4}$.

The comparisons of estimated costs and observed costs of test queries in our experiments indicated that the estimated costs of the majority of test queries had relative errors within 30%. A typical comparison of costs is shown in Fig. 5 (for $G_{1\ 4}$ on ORACLE) where the estimated costs for 83% of the test queries have relative

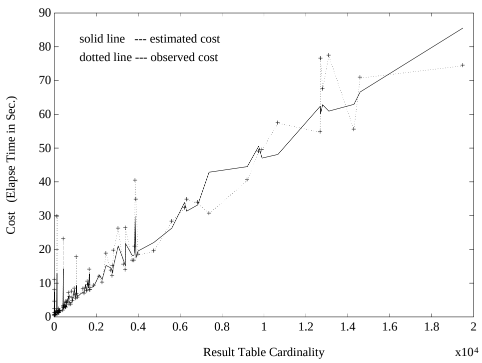

Fig. 5. Estimated and Observed Costs for Test Queries in $G_{1\ 4}$ on ORACLE 7.0

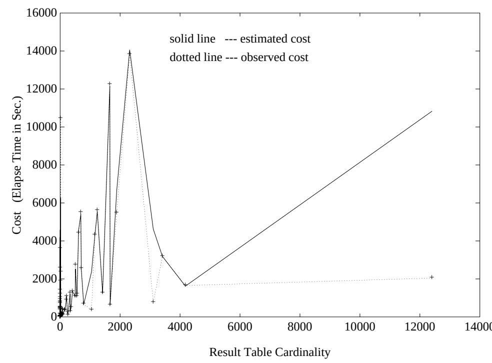

Fig. 6. Estimated and Observed Costs for Test Queries in $G_{2\ 3}$ on ORACLE 7.0

errors within 30%.

As mentioned before, a cost formula can be further improved by refining the relevant query class in the top-down classification approach. For example, if we want to improve the cost formula for $G_{2\ 3}$ on ORACLE which has the worst standard error of estimation (68.5% of the corresponding average observed cost), we can divide $G_{2\ 3}$ into smaller query classes and then derive new cost formulas for the subclasses. To demonstrate such an improvement, let us consider two subclasses of $G_{2\ 3}$:

$$\begin{aligned} G'_{2\ 3} \ = \ \{ & \ join\ queries\ in\ G_{2\ 3}\ whose\ qualifications\ have\ at\ least\ one \\ & conjunct\ R_i.a_n = C\ where\ R_i.a_n\ is\ a\ clustered-indexed\ column \\ & in\ one\ operand\ table\ R_i,\ but\ no\ index-usable\ conjunct\ for\ the \\ & other\ operand\ table\ R_j\ \} \\ G''_{2\ 3} \ = \ \{ & \ join\ queries\ in\ G_{2\ 3}\ whose\ qualifications\ have\ at\ least\ one \\ & conjunct\ R_i.a_n\ \theta\ C\ where\ R_i.a_n\ is\ a\ clustered-indexed\ column \\ & in\ one\ operand\ table\ R_i\ and\ \theta \in \{>,\ \geq,\ <,\ \leq\},\ but\ no \\ & index-usable\ conjunct\ for\ the\ other\ operand\ table\ R_j\ \} \end{aligned}$$

These subclasses are more homogeneous than the original class. Applying multiple regression analysis, we derived a cost formula for each of $G'_{2\ 3}$ and $G''_{2\ 3}$. The

Table 8. Statistical Measures for Two Subclasses of $G_{2\ 3}$ on ORACLE

query class	F-statistic (critical value at $\alpha = 0.01$)	coefficient of multiple determination	standard error of estimation (S)	average observed cost (A)	S/A %
$G'_{2\ 3}$	5122.89 (> 3.82)	0.99427	0.62906	0.76499e+1	8.2%
$G''_{2\ 3}$	1360.23 (> 4.27)	0.97190	0.89347e+2	0.36239e+3	24.7%

statistical measures for these two formulas are given in Table 8. From the table, we can see that the standard error of estimation has been improved to 8.2% and 24.7% of the average observed costs for $G'_{2\ 3}$ and $G''_{2\ 3}$, respectively. The coefficient of multiple determination has also been improved significantly for the two subclasses (from 92.3% to 99.1% and 97.2%, respectively). Using the derived cost formulas to estimate the costs of randomly-generated test queries, we found that 79% of the test queries had an estimated cost with a relative error within 30% by using the original formula for $G_{2\ 3}$ (see Fig. 6); while 98% of the test queries had an estimated cost with a relative error within 30% by using the new formula for $G'_{2\ 3}$ (see Fig. 7), and 92% of the test queries had an estimated cost with a relative error within 30% by using the new formula for $G''_{2\ 3}$ (see Fig. 8). Therefore, cost formulas can be significantly improved by refining a query classification.

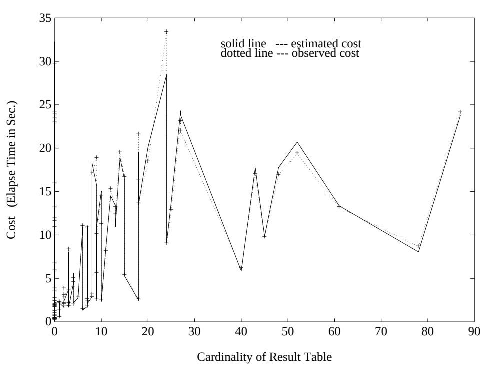

Fig. 7. Observed and Estimated Costs for Test Queries in $G'_{2\ 3}$ on ORACLE 7.0

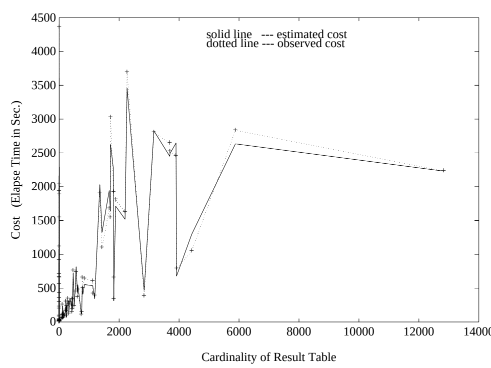

Fig. 8. Observed and Estimated Costs for Test Queries in $G''_{2\ 3}$ on ORACLE 7.0

In fact, a satisfactory classification of local queries can be obtained via either the top-down approach or the bottom-up approach. The former progressively converts an unsatisfactory one (in terms of accuracy of derived cost formulas) into a satisfactory one (in terms of both accuracy and maintenance overhead of derived cost formulas) by refining undesirable classes into smaller and smaller subclasses (for example, from $G_{2\ 3}$ to $G'_{2\ 3}$, $G''_{2\ 3}$, ...). The latter progressively converts an unsatisfactory classification (in terms of maintenance overhead of derived cost formulas) into a satisfactory one (in terms of both maintenance overhead and accuracy of derived cost formulas) by merging some classes into larger and larger superclasses (for example, from $G'_{2\ 3}$, $G''_{2\ 3}$, ..., to $G_{2\ 3}$). A satisfactory classification usually represents a reasonable balance between accuracy and maintenance overhead. In the top-down approach, accuracy is getting better, while maintenance overhead is getting higher, during the classification procedure. In the bottom-up approach, maintenance overhead is getting lower, while accuracy is getting worse, during the classification procedure. Clearly, the classification procedure should terminate when both accuracy and maintenance overhead reach a satisfactory level. In general, the two approaches can produce the same classification if sufficient rules are provided. However, the most efficient classification approach is the hybrid one as described in Section 4.3. A good classification can usually yield a set of good cost models, from which the query optimizer can choose a good execution plan for a query that, in turn, improves the performance of the query.

7. Conclusions

Global query optimization in a multidatabase system has attracted more and more researchers in the database area recently. One of the major challenges for global query optimization in an MDBS is that the required local cost information may not be available at the global level. In our previous work, we proposed a query sampling method to tackle this challenge. The key idea is to classify local queries and then derive a cost estimation formula for each query class by performing regression analysis on observed costs of sample queries. Query classification is clearly crucial in this method. In this paper, we discuss the details of several classification approaches.

The objective of query classification is to group a given set of queries into a number of classes so that the costs of queries in each class can be estimated accurately by the same cost formula. Furthermore, the characteristics of queries in each query class should be identified so that sample queries can be efficiently generated for the query classes according to their characteristics during derivation of cost formulas by using the query sampling method. The classification principle employed to achieve the above goal is to put all the queries that are likely to be executed by the same access method into the same class because their performance behavior usually follows the same pattern. In other words, their costs can be well described by the same formula.

Based on relevant knowledge of query optimizers in DBMSs, a set of classification rules are identified, which can be categorized into three types. The rules of the first type consider the effect of changing parameter values (e.g., operand table(s), referenced columns and constants) for a query on the selection of its access method. We notice that, for a given query (with a given format), a query optimizer chooses an access method according to its parameter values as well as their relevant physical properties and statistics (if any). Based this observation, three parameter substitution rules are introduced to describe equivalence among queries at three levels for query classification; namely, equivalence with identical statistics/constants, equivalence with similar statistics/constants, and equivalence regardless of statistics/constants. The rules of the second type consider the effect of changing the format of a query on the selection of its access method. The similar predicates merging rule, the project lists merging rule and the join order changing rule belong to this category. The rules of the third type incorporate the common policies used by query optimizers in many DBMSs for choosing an access method for a given query. Classification rules may need to be adjusted according to the actual policies used by a specific DBMS. In general, the more information we have about a local DBMS (policies), the more as well as the better classification rules we can get, and therefore a better classification would result.

To apply multiple rules to classify a query set, two issues are considered; that is, how to compose multiple rules to yield a composite classification rule and how to eliminate a redundant rule implied by one or more others. A query classification is defined by the final composite rule obtained by composing a set of rules without redundancy.

Two types of classification approaches are suggested: bottom-up and top-down. The bottom-up approach starts with a classification in which each query comprises a separate class and then repeatedly merges classes into larger superclasses until a satisfactory classification is obtained. An algorithm for this approach is presented, and its correctness is proven. The bottom-up approach provides a theoretical insight into the query classification problem. The top-down approach, on the other hand, provides a practically feasible solution to the query classification problem. It starts with a classification containing a sole class of all queries and then repeatedly divides a class into smaller subclasses until a satisfactory classification is obtained. To compose multiple classification rules, the OR logic is used in the bottom-up approach, while the AND logic is adoted in the top-down approach. Theoretically speaking, a satisfactory classification can be achieved via either the bottom-up approach or the top-down approach if sufficient classification rules are provided. However, the most efficient approach in pratice is usually the hybrid one that combines the bottom-up and top-down approaches.

To test which class a query belongs to, an efficient query membership test method based on ranks is proposed. The rank of a qualification condition in a query is calculated in such a way that the access method likely to be employed for the query is indicated. It is shown that this method can effectively test the membership of a query for a classification obtained from the common classification rules that are applicable for many DBMSs.

Our experimental results demonstrate that a query classification obtained by applying the techniques suggested in this paper can be used to derive good local cost formulas in an MDBS environment.

The query classification techniques presented in this paper can be used not only for estimation of local cost parameters in an MDBS but also for performance analysis of a DBMS, simulation of query optimization, and evaluation of existing cost models. In the future, we plan to generalize our techniques for non-relational local database systems (such as object-oriented one) in an MDBS and also investigate how to classify queries based on the performance patterns observed from the costs of user queries at runtime.

Acknowledgements

We would like to thank Grant E. Weddell, Frank W. Tompa, Amit Sheth, Kenneth Salem, M. Tamer Özsu, and Frank Olken for their insightful suggestions and comments. We would also like to thank Lauri Brown and Wendy Powley for their help in setting up the experimental environments.

References

1. S. Adali, K.S. Candan, Y. Papakonstantinou and V.S. Subrahmanian, Query caching and optimization in distributed mediator systems, in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, Montreal, Canada, 1996, 137–48.
2. G.K. Attaluri, D.P. Bradshaw, N. Coburn, P.-Å. Larson, P. Martin, A. Silberschatz, J. Slonim and Qiang Zhu, The CORDS multidatabase project, *IBM Systems Journal*, **34** (1995) 39–62.
3. W. Du, R. Krishnamurthy and M. C. Shan, Query optimization in heterogeneous DBMS, in *Proc. of the 18th VLDB Conf.*, Vancouver, BC, Canada, August 1992, 277–91.
4. W. Du, M. C. Shan and U. Dayal, Reducing multidatabase query response time by tree balancing, in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, San Jose, CA, 1995, 293–303.
5. C. Evrendilek, A. Dogac, S. Nural and F. Ozcan, Multidatabase query optimization, *Distributed and Parallel Databases*, **5** (1997) 77–114.
6. G. Gardarin, F. Sha and Z.-H. Tang, Calibrating the query optimizer cost model of IRO-DB: an object-oriented federated database system, in *Proc. of the 22nd VLDB Conf.*, Mumbai (Bombay), India, 1996, 378–89.
7. W. C. Hou, et al., Statistical estimators for relational algebra expressions, in *Proc. of PODS*, 1988, 276–87.
8. J. I. Icaza, Adaptive Selection of Query Processing Strategies, PhD thesis, University of Waterloo, 1987.
9. R. Johnsonbaugh, *Discrete Mathematics* (Macmillan Publishing Company, 1994).
10. C. Lee and C.-J. Chen, Query optimization in multidatabase systems considering schema conflicts, *IEEE Trans. on Knowledge and Data Eng.*, **9** (1997) 941–55.
11. R. J. Lipton and J. F. Naughton, Practical selectivity estimation through adaptive sampling, in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 1990, 1–11.
12. H. Lu and M.-C. Shan, On global query optimization in multidatabase systems, in *Proc. of the 2nd Int'l workshop on Research Issues on Data Eng.*, Tempe, Arizona, USA, 1992, 217.
13. M. Muralikrishna and D. J. DeWitt, Equi-Depth histograms for estimating selectivity factors for multi-Dimensional queries, in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 1988, 28–36.
14. H. Naacke, G. Gardarin and A. Tomasic, Leveraging mediator cost models with heterogeneous data sources, in *Proc. of 14th Int'l Conf. on Data Eng.*, 1998, 351–60.
15. F. Olken and D. Rotem, Simple random sampling from relational databases, in *Proc. of the 12th VLDB Conf.*, 1986, 160–9.

16. M. T. Roth, F. Ozcan and L. M. Haas, Cost models DO matter: providing cost information for diverse data sources in a federated system, in *Proc. of the 25th VLDB Conf.*, Edinburgh, Scotland, 1999, 599–610.
17. G. P. Shapiro and C. Connel, Accurate estimation of the number of tuples satisfying a condition, in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 1984, 256–76.
18. D. K. Subramanian and K. Subramanian, Query optimization in multidatabase systems, *Distributed and Parallel Databases*, **6** (1998) 183–210.
19. Qiang Zhu, Query optimization in multidatabase systems, in *Proc. of the 1992 IBM CAS Conf., vol.II*, Toronto, Canada, Nov. 1992, 111–27.
20. Qiang Zhu and P.-Å. Larson, A fuzzy query optimization approach for multidatabase systems, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, **5** (1997) 701 – 22.
21. Qiang Zhu and P.-Å. Larson, Solving local cost estimation problem for global query optimization in multidatabase systems, *Distributed and Parallel Databases*, **6** (1998) 373–420.
22. Qiang Zhu and P.-Å. Larson, Building regression cost models for multidatabase systems, in *Proc. of the 4th IEEE Int'l Conf. on Parallel and Distributed Inf. Syst.*, Miami Beach, Florida, Dec. 1996, 220–31.
23. Qiang Zhu and P.-Å. Larson, A query sampling method for estimating local cost parameters in a multidatabase system, in *Proc. of the 10th IEEE Int'l Conf. on Data Eng.*, Houston, Texas, Feb. 1994, 144–53.
24. Qiang Zhu and P.-Å. Larson, Establishing a fuzzy cost model for query optimization in a multidatabase system, in *Proc. of the 27th IEEE/ACM Hawaii Int'l Conf. on Syst. Sci.*, Maui, Hawaii, Jan. 1994, 263–72.
25. Qiang Zhu, Yu Sun and S. Motheramgari, Developing cost models with qualitative variables for dynamic multidatabase environment, in *Proc. of the 16th IEEE Int'l Conf. on Data Eng.*, San Diego, CA, Feb. 2000, 413–24.