HEURISTIC ALGORITHMS FOR DISTRIBUTED QUERY PROCESSING[1]

P. Bodorik

School of Computer Science,
Technical University of Nova Scotia,
P.O. Box 1000, Halifax, Nova Scotia,
B3J 2X4, Canada

J.S. Riordon

Dept. of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, K1S 5B6, Canada

ABSTRACT

This paper examines heuristic algorithms for processing distributed queries using generalized joins. As this optimization problem is NP-hard heuristic algorithms are deemed to be justified. A heuristic algorithm to form/formulate strategies to process queries is presented. It has a special property in that its overhead can be "controlled": The higher its overhead the better the strategies it produces. Modeling on a test-bed of queries is used to demonstrate that there is a trade-off between the strategy's execution and formulation delays. The modeling results also support the notion that simple greedy heuristic algorithms such as are proposed by many researchers are sufficient in that they are likely to lead to near-optimal strategies and that increasing the overhead in forming strategies is only marginally beneficial. Both the strategy formulation and execution delays are examined in relation to the number of operations specified by the strategy and the total size of partial results.

1. INTRODUCTION

Although researchers are currently turning their attention to multi-query optimization [Cellery, 1980; Ounegbe, 1983; Carey, 1985; Reuter, 1986; Su, 1986; Kim, 1986; Sellis, 1988] and dynamic query processing [Nguyen, 1981; Yu, 1983, 1986; Wong, 1986; Bodorik, 1988b, 1988c], most research on distributed query processing assumes a single query environment and static processing. In a single query environment, performance of a single query is optimized, while static processing implies that an optimized strategy is not modified (it remains static) once its execution commences. Furthermore, most research concentrates on an important class of queries, the Select-Project-Join (SPJ) queries [Ceri, 1984].

Reducing the size of relations by semi-joins has received a great deal of attention [Hevner, 1979; Bernstein, 1981; Apers, 1983; Perizzo, 1984; Yu, 1982a, 1982b, 1985; Ceri,1986]. Although the use of semi-joins has generally been accepted as a good processing tactic[2], it has also been recognized [Bernstein, 1981; Epstein, 1980] that semi-join and generalized join processing tactics will have to be integrated.

Epstein et al. [1978] proposed one of the earliest methods which uses joins as processing tactic. Their algorithm which is an adaptation of the optimizing algorithm for the INGRES relational DBMS to the distributed environment [Epstein, 1986]. A join processing heuristic algorithm for the ADD [Mahmoud, 1979; Toth, 1982], first decomposes a query into a "Class A" sub-queries which produce results processed by a sub-optimal heuristic. [Daniels, 1982; Selinger, 1980; Lohman, 1985; Mackert, 1986a, 1986b] describe optimization of queries in System R*. The optimizer generates alternative strategies as is performed for the System R, but modified for the distributed environment. Dynamic programming is used to generate alternative strategies, while detailed cost calculations, which include both the network and CPU processing costs, are used to evaluate alternatives for executing a join.

To optimize a distributed query implies that an optimizing algorithm formulates/forms a strategy, which is a sequence of relational operations and locations for their execution, to process a given query. As various sub-problems dealing with optimizing queries processed by semi-join or join processing tactics have been shown to be NP-hard [Hevner, 1980; Yu, 1982a, 1982b; Gavish, 1982, Segev, 1986; Bodorik, 1987], most algorithms to optimize strategies are heuristic. Although various heuristic algorithms have been proposed, in particular greedy [Horowitz, 1985] heuristic algorithms, relatively little attention has been paid to their "performance". Some of the algorithms were evaluated to determine the delay induced through execution of strategies they produce[1]. None of the algorithms were evaluated, however, to determine how close execution delays of "their" strategies are in comparison to the delays of optimal strategies. Moreover, for all of these algorithms it is assumed that the strategy formulation delay is negligible in comparison to its (the strategy's) execution delay. It is these two issues that are primarily investigated in this paper for a DDB in which partitioned relations are permitted.

For the purpose of this investigation it is assumed that the query response time includes not only delays due to execution of strategies but also delays due to their formulation. An algorithm with a property, adopted from Artificial Intelligence research, in that its overhead can be "controlled" is presented. Modeling on a test-bed of sample queries is used to examine (i) existence of a trade-off between the strategy's performance/cost and overhead expended in its formulation and (ii) a notion that simple greedy heuristic algorithms, such as those that have been proposed by many researchers, are sufficient in that they are likely to lead to near-optimal strategies. Both the strategy formulation and execution delays are examined in relation to the number of operations specified by the strategy and the total size of partial results.

[1]This research has been supported in part by a grant from the Natural Science and Engineering Research Council of Canada.

[2]There are some exceptions such as Lohman [1985].

[1]For example, Black's [1982] heuristic algorithm to formulate semi-join programs is claimed to produce strategies which induce lower delay than that of strategies produced by the algorithm for SDD-1 [Bernstein, 1981]. An algorithm proposed for semi-join programs in [Yu, 1983] is claimed to produce strategies which have a lower execution delay than that of strategies produced by algorithms discussed in [Bernstein, 1981] and [Black, 1982].

The following section presents assumptions for distributed query processing. The presentation of the heuristic algorithm is followed by the description of the modeled distributed application and the sample queries. Finally, presentation and analysis of the modeling results is followed by the summary and conclusions. The Appendix includes further details describing the parameters of the modelled application.

2. DISTRIBUTED PROCESSING MODEL

This section outlines a distributed query processing model[1] in order to establish a framework for presentation and treatment of the heuristic algorithms. As is usual with the treatment of processing distributed queries, several assumptions on the query form and its representation are made.

RELATIONS AND RELATIONAL OPERATIONS

Relations may be *partitioned* horizontally and/or vertically. Composition of a vertically partitioned relation is achieved by an equi-join. Composition of a horizontally partitioned relation is achieved through a union. A join of two horizontally partitioned relations is always preceded by a union to compose at least one of the horizontally partitioned relations. A *fragmented* join is available for a join of a non-fragmented relation with a horizontally fragmented relation. The non-fragmented relation is broadcast to the fragments of the partitioned relation. A "local" join is executed as soon as the non-partitioned relation arrives to a fragment's location. Consequently, a fragmented join is a collection of joins of the non-partitioned relation with partitions of the other relation. The result is a relation which is horizontally partitioned.

DISTRIBUTED QUERY PROCESSING

A query is assumed to be in the conjunctive normal form such that each term/formula has at most two relational variables. The two-variable terms are processed by joins. As one-variable terms are processed first in order to reduce the size of relations which are operands of expensive joins, it is also assumed that all of the query's terms are two-variable. Each term has a selectivity factor which is used in estimation of partial results' sizes [Epstein, 1980; Bernstein, 1981; Wong, 1982]. A query satisfying these assumptions can be represented by a hyper-edge graph in which vertices represent relations and edges represent the two-variable terms/formulas.[2]

Let
$Q = (V, E)$ ... represent the query where
$V = \{R_1, R_2, ..., R_n\}$... is the set of relations referenced by the query;
$E = \{P_1, P_2, ..., P_m\}$ ... is the set of two-variable terms;
n ... denotes the number of relations referenced by the query and
m ... denotes the number of two-variable terms.

Each term P_k is of the form $(R_i.a\ \theta\ R_j.b)$, where $R_i, R_j \in V$; $R_i.a$ and $R_j.b$ are attributes of R_i, R_j, respectively; and $\theta \in \{ =, <, >, \leq, \geq \}$ is an arithmetic relational operator. Also let

f_k ... denote the selectivity factor of the term P_k;

$c[R_i]$ ... denote the cardinality of the relation R_i;
$w[R_i]$ ... denote the "width" of the relation R_i, i.e., the total number of bytes taken by attribute values of each tuple in R_i; and
$s[R_i]$ ... denote the size of the relation R_i; $s[R_i] = c[R_i] * w[R_i]$.

For convenience in determining the size of partial results it is assumed that the values of attributes are distributed uniformly and independently of each other. The query's terms/formulae/predicates are also assumed to be independent in that no predicate is implied through others via transitivity. For each of the two-variable terms/predicates, referring say to relations R_i and R_j, there is a selectivity factor which determines the expected fraction of tuple pairs from R_i and R_j which satisfy it. These assumptions imply that a partial result of evaluating a set of predicates has the cardinality which is the product of the predicates' selectivities and cardinalities of relations referred to by these predicates [Ibaraki, 1984].

The cost of a network data transfer is assumed to be a linear function of the volume of transferred data. The unit cost of a network transfer is represented by a matrix. A matrix element represents the cost/delay to transfer a unit of data (byte) between two network locations.

As restrictions and projections are assumed to be processed as soon as possible, a strategy is a specification of join operations[1] on relations and the network locations of their execution. Obviously, some joins have operands which are results of other join executions. A relational operation is executed as soon as its operands are ready. A relation which is a result of a relational operator execution is transferred to another network location only when it is completely formed. Consequently, pipelining, which may improve the query response time is not considered. Semi-joins, as a processing tactic, are not considered either.

The strategy's execution cost is measured in term of its delay. Any time an operation is under consideration, the delay for the resulting relation is evaluated. It depends on the time when the operation can commence, i.e., when its operands are available, and on the delay to perform the operation itself. An operation can be either transferring a relation or relational operator execution. The delay to execute a relational operation depends on the resulting access to the secondary storage devices; thus it depends not only on the average delay to access such devices but also on the size of operands which must be retrieved, the size of the result which must be stored and the size of any intermediate results which must be formed. For example, to perform a projection on the relation R, the tuples of R must be retrieved, sorted and only then duplicate tuples may be removed. The delay to sort is included in the calculation of the projection's result delay.

3. HEURISTIC ALGORITHMS

To address the question of whether or not a trade-off between the strategy formulation overhead and the strategy execution delay exists an optimizing algorithm with a controllable overhead has been developed. It is based on a simple idea that the solution space, the space of strategies, can be developed in "levels" as is proposed in [Reingold, 1977]. The remainder of this section is devoted to the description of this algorithm. First, the representation of the space of strategies by

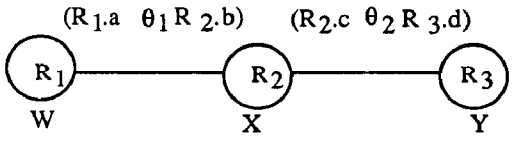
[1] For details see [Bodorik, 1985].

[2] It is convenient to combine terms applying to the same two relational variables (relations) into a conjunction of these terms and represent it by one edge in the graph. In this way hyper-edges can be removed.

[1] Discussion of unions is postponed to the end of section 3.

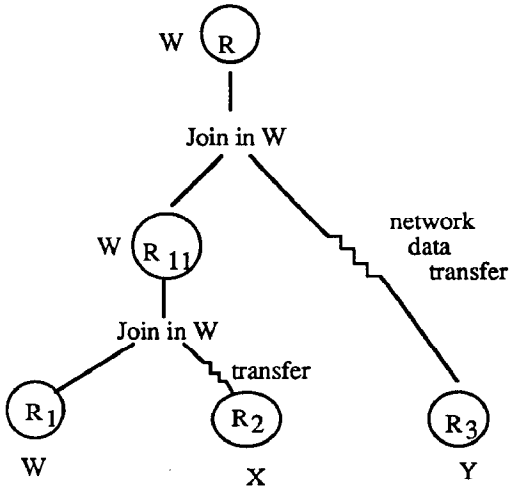
a graph is presented. The graph is a collection of non-disjoint trees, such that each tree represents a strategy. It is organized in levels such that each element of a level i represents a partial result of i join executions. The space of strategies is non-redundant, that is, each graph element represents a result of a unique strategy/sub-strategy.

The algorithm OPT_GV, which constructs this complete space of strategies level by level, is described first. The presentation of the heuristic algorithm, which is a modification of the algorithm OPT_GV, then follows. The section concludes by discussing implications of existence of horizontally partitioned relations and inclusion of the union operator which is necessary for their processing.

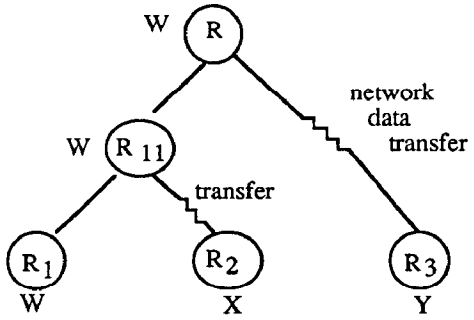


W, X, Y are network locations

(a) Query graph for (R 1.a θ1 R 2.b) Λ (R2.c θ2 R 3.d)

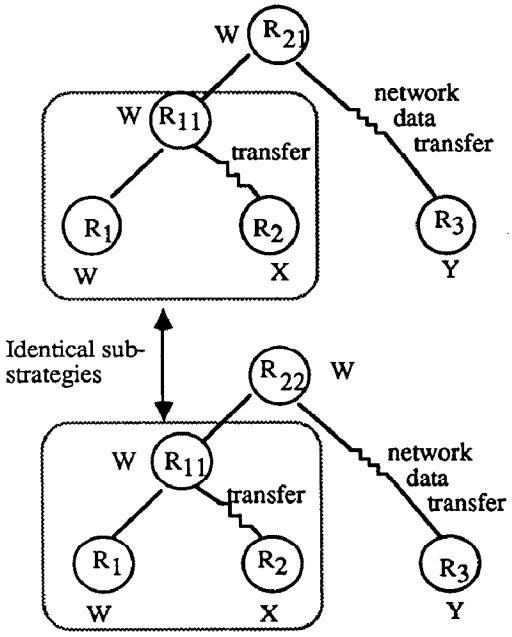


(b) Strategy representation with explicit joins

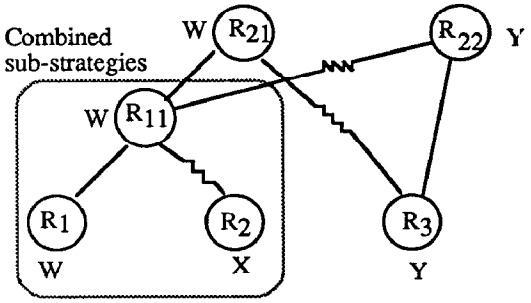


(c) Strategy representation with implicit joins

Figure 1 Query graph and strategy representation



(a) A common sub-strategy for two strategies



(b) Strategies with merged trees (sub-strategies)

Figure 2 Graph representation of strategies

SPACE OF STRATEGIES

Recall that a strategy specifies the sequence and network locations of join executions. The space of strategies can therefore be represented by a collection of trees, which are not disjoint, such that each tree represents a strategy. Figure 1(b) shows the usual representation of a strategy by a tree. Since restrictions and projections are assumed to be processed as soon as possible, they are excluded from the tree representation. As the only relational operations under considerations are joins, the graphical representation can be simplified by their removal. Figure 1(c) shows the strategy's simplified tree representation. The leaves are assumed to be relations referenced by the query while non-leaf nodes are results of join operations. Figure 2 shows a query and two strategies to process it. The strategies are such that they have a common sub-strategy to derive an identical partial result, that is, an identical relation stored by the same information processor. Instead of representing each strategy separately, their identical portions can be merged together to obtain a graphical representation of both strategies as

is shown in Figure 2(c). In this way a space of strategies can be developed such that each unique strategy/sub-strategy appears only once in the solution space.

ALGORITHM FOR A COMPLETE SPACE OF STRATEGIES

The space of strategies can be simply viewed to consist of levels 0, 1, ..., n-1. Each level has elements which represent relations. An element of level i represents a relation in a particular network location which is a result of a unique sub-strategy containing exactly i joins. The elements of the level zero represent "original" relations referenced by the query, which are assumed to have been "processed" by restrictions and projections in order to reduce their size. Elements of the level n-1, called final elements, represent the query result, each derived by a unique strategy. Recall that n is the number of relations referenced by the query and n-1 is the number of joins required to process it.

INPUT: Query q in a conjunctive normal form, selectivities, statistical information about the relations, network locations of relations, and delays due to CPU processing and network data transfer per unit of data.
OUTPUT: Space of strategies represented by a graph.
METHOD: The graph of strategies consists of elements organized in levels 0, 1, ..., m, where m is the number of joins required to process the query. Elements of the level 0 represent relations referred to by the query. An element of a level p represents a relation which is a result of a unique strategy consisting of exactly p joins. The level p is created from elements/relations of levels p_x and p_y, such that $p = 1 + p_x + p_y$, that is from two relations such that one is a result of p_x while the other the result of p_y join executions, respectively.
STEP 1: Create elements of the level 0, which represent relations referred to by the query. Calculate their cost (delay) and size resulting from the execution of restrictions and projections. Projections remove attributes which are neither target nor joining (not referred to be any of the query's two-variable terms).
STEP 2: Do Step (3) for p = 2, 3, ..., m, to create the level p on each iteration. Each level contains elements which represent relations which are results of p join executions.
STEP 3: Insert elements representing results of p join executions into the level p of the graph of strategies by executing Step (4) for p_x = p-1, p-2, ..., $\lfloor p/2 \rfloor$.
STEP 4: Let the graph level p_y be such that $p = 1 + p_x + p_y$. For each element, representing a relation R_i of the level p_x, search the level p_y for elements representing relations which can be joined with R_i. For each such element, representing say a relation R_j, create z new elements, each representing the result of the join of R_i with R_j executed at one of the z unique network locations (executed by one of the z possible information processor). Calculate their size and delays and insert them in the level p of the graph of strategies and save information about the join operands used to create them. Each element represents the result of a unique strategy consisting of p joins.

(NOTE that the graph's level m contains elements which represent the query result, each obtained by a unique strategy. To find the optimal strategy, the level m is searched for the minimum cost element/relation.)

Figure 3 Algorithm OPT_GV

The graph can be built in levels. Elements of the level 0 are created first; they represent the relations referenced by the query such that they are assumed to have been processed by restrictions and projections. Then levels p = 1, 2, ..., n-1 are built in that order. An element of a level p, p > 0, represents a partial result of p join executions, located in a specific network location. It can be constructed from two elements of levels p_x and p_y, $p_x \geq 0$ and $p_y \geq 0$, such that $p = 1 + p_x + p_y$. The two elements represent partial results of p_x and p_y joins, respectively; they are operands of the join execution resulting in the relation represented by the level p element. An outline of the algorithm, called OPT_GV (OPTimal for a Global View)[1], is shown in Figure 3. For details see [Bodorik, 1985]. Once the space of strategies is constructed, the optimal strategy can be found by searching the level n-1 for the minimum cost element. The level n-1 contains elements representing the query result such that each is obtained by a unique strategy.

Note that the algorithm develops a non-redundant space of strategies in that each of its elements is a result of a unique strategy/sub-strategy. The whole solution space, however, has to be stored to find the optimal strategy. If the solution space were represented by a decision tree and developed through a tree search such as the depth-first search, some sub-strategies would be examined/built many times over, but the entire tree would not have to be stored at one time.

HEURISTIC ALGORITHM

Consider now a greedy heuristic for the minimization problem under consideration. Since the problem is to decide on the sequence and network locations for executing joins, the heuristic chooses the least expensive choice regardless of consequences on the choices for the remainder of the strategy. Thus, the heuristic chooses that join execution which incurs the minimal cost. The query now can be modified to represent this choice. The two operands of the selected join are removed and replaced by the join result and the process repeats; the heuristic again chooses the "cheapest" join. This repeats until the query result is obtained. In the context of the previously described space of strategies this heuristic procedure implies that only the first two levels of the solution space are developed. The minimum cost partial result of one join is chosen, the query is modified and the process repeats.

A simple modification to the algorithm which builds the solution space also yields the greedy heuristic. In fact, the modification can be such as to yield a range of heuristic algorithms with different levels of overhead. Instead of developing only the levels 0 and 1 of the solution space, the algorithm may develop the levels 0, 1, ..., N; N < n - 1. The level N represents all possible results of N join executions. The cheapest result is chosen from this level and the sub-strategy to derive it is adopted as a part of the query processing strategy. The relations and two-variable terms processed by the sub-strategy are removed from the query and are replaced by the sub-strategy's result. The process continues until the query result is obtained. This algorithm, called HEURISTIC, is outlined[2] in Figure 4 and an example of one of its iterations is shown in Figure 5. It should be realized that if N is sufficiently large, i.e., N ≥ n - 1, where n - 1 is is the number of joins required to process the query, the algorithm develops the whole solution space on the first iteration and finds the query's optimal strategy.

[1]Global View [Mahmoud, 1979] of a distributed database is such that it does not contain relations which are partitioned horizontally and/or vertically.
[2]The efficiency of the algorithm is further improved by retaining, whenever possible, information (graph elements) from one iteration to another.

INPUT: N, the number of levels of the partial graph of strategies to be created on each iteration. Query q in a conjunctive normal form, selectivities, statistical information about the relations, network locations of relations, and delays due to CPU processing and network data transfer per unit of data.

OUTPUT: A strategy to process the given query.

METHOD: There are three repeatedly executed phases:

a) Create the levels 0, 1, ..., N of the graph of strategies using the algorithm OPT_GV.

b) Find the minimum cost partial result of the level N. Save the sub-strategy for this relation (it is adopted as part of the query processing strategy).

c) Modify the query by (i) removing relations and two-variable terms processed by the selected sub-strategy and (ii) adding the relation, the result of the selected sub-strategy and modifying the two-variable terms. Repeat (a) to (c) until the strategy to process the query is completely derived.

STEP 1: Create a partial graph of strategies consisting of levels 0, 1, ..., N, possibly by the algorithm OPT_GV adapted for this task.

STEP 2: Search the level N for the minimum cost relation. Save the sub-strategy to derive this relation; it will become the part of the query processing strategy.

STEP 3: Remove any relations and two-variable terms processed by the selected sub-strategy from the query. Augment the query by the result of the sub-strategy and modify the remaining two-variable terms. Assume for example, that the selected sub-strategy processes two-variable terms (R_1.a θ_1 R_2.b) and (R_2.c θ_1 R_3.d) by two joins resulting in the relation R'. These two terms and relations R_1, R_2 and R_3 are removed from the query, the query is augmented by the relation R', the sub-strategy's result. In addition, any of the query's two-variable terms which refers to exactly one of the relations removed from the query, say R_k, is modified to refer to the sub-strategy's result, the relation R'. For example, a two-variable term (R_1.a θ_3 R_k.e) is modified to (R'.a θ_3 R_k.e).

Steps (1) to (3) are repeated until only one relation remains (the query result).

Figure 4 Algorithm HEURISTIC

The parameter N specifies the number of levels to be developed on each iteration. It is expected that as the value of N increases, execution delays of strategies decrease. This is because the amount of information used in their formulation is increased; however, overhead also increases.

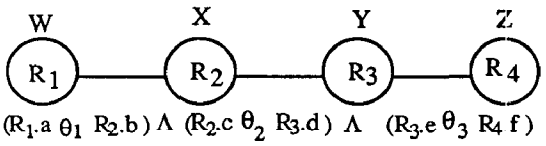
The algorithm just described was used on sample queries of a particular application. The modeled application, queries, and modeling results are described in subsequent sections. As the modeled application includes relations which are horizontally partitioned, their implications on the space of strategies and the algorithms are briefly discussed.

HORIZONTALLY PARTITIONED RELATIONS

A horizontally partitioned relation, or a fragmented relation for short, is assumed to be processed in the following way. Any restrictions and/or projections are performed on each partition/fragment as soon as possible. A join of two fragmented relations is always preceded by a union of fragments of at least one of the operand relations. A join of a non-partitioned relation with a partitioned one leads to a "fragmented" join. A copy of the non-fragmented relation is delivered to each of the fragments for a local join. The result of a fragmented join is a fragmented relation which is ready for a union of partitions or another join.

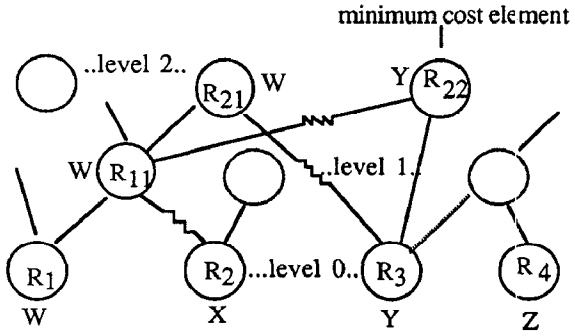
The above assumptions regarding processing of fragmented relations have the following implications on the space of strategies and algorithms. A join of two non-fragmented relations or a union of fragments is considered to be one operation. A fragmented join is also considered to be one operation. The space of strategies is still viewed to be organized and built in levels. A level p contains elements, such that each represents the result of a unique sub-strategy which consists of exactly p operations.

The algorithms under discussion were used in modeling of an application of a Distributed Data Base (DDB) [Bodorik, 1985]. The results of this modeling are described in the following sections. Algorithms as used in modeling had one additional restriction in that a join of non-fragmented relations, or a union of fragments can be executed only at a network location of one of its operands.

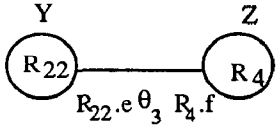


W, X, Y and Z are network locations

(a) Query graph



(b) Levels (portions of) 0, 1 and 2 of the graph of strategies



(c) Modified query to reflect the adoption of the sub-strategy which produces R_{22}

Figure 5 One iteration of the algorithm HEURISTIC with N = 2

4. MODELED APPLICATION

This section contains a brief outline of the modeled application of the DDB. A more detailed outline can be found in Appendix, while detailed description can be found in [Bodorik, 1985].

The modeled application of the DDB is claimed to be realistic and one for which available networking and processing resources are assumed to be modest. The data base is of a type needed to support an information management system for a medium size company having five geographically separated offices. The company utilizes services of a public network to which it is connected through rather slow connections ranging in data transfer rate from 1200 to 9600 bits/sec. Information processors are minicomputers in the VAX 750 to 785 range.

The 21 modeled queries are of a type which might be utilized in a management information system. They range from simple queries referencing only one or two relations to queries which are processed through a significant number of joins and/or unions. They reference relations, some of which are horizontally partitioned, with cardinalities ranging from a few tuples to tens of thousands of tuples.

5. MODELING RESULTS

Recall that the heuristic algorithm HEURISTIC has a parameter N which is used to control the amount of information for "look-ahead" when making decisions on joins and/or unions and their locations of execution. When deciding on which join/union and where it should be executed, N controls the number of levels that are considered in each iteration. An increase in the value of N results in an increase in overhead, but it is also anticipated that the execution delay of strategies would be lower. If N is sufficiently large, the algorithm develops a complete solution space and determines the minimum delay strategy. To examine the relationship between the strategy formulation and execution delays for the modeled application, the algorithm HEURISTIC was used on a sample of 21 queries which were chosen in such a way as to give a representative range for the number of operations which are required for their execution. From the 21 modeled queries:

3 require 2 operations;	6 require 3 operations;
5 require 4 operations;	3 require 5 operations;
2 require 6 operations; and	2 require 7 operations.

One would expect that, in general, most queries would require between two to five joins and/or unions while the number of queries which require six or seven operations would be smaller. There would be a few queries requiring one or more than seven joins and/or unions; none of the modeled queries fall into this category.

Let

m ... denote the number of modeled queries (m = 21);

q_i ... denote the i^{th} query;

$T_f[i, N=x]$ ... denote the delay to formulate a strategy for the query q_i such that the algorithm uses the value of x for the parameter N;

$T_e[i, N=x]$ ... denote the execution delay of the strategy, formulated by the algorithm using N=x, to process the query q_i;

$A_f[N=x]$ ... denote the average delay to formulate strategies with the parameter N having the value of x for each query:

$$A_f[N=x] = \sum_{i=1}^{m} T_f[i, N=x] / m$$

$A_e[N=x]$ ... denote the average execution delay of strategies formulated by the algorithm with the parameter N having the value of x:

$$A_e[N=x] = \sum_{i=1}^{m} T_e[i, N=x] / m$$

$T_t[i, N=x]$ ... denote the query response time for the query q_i which is processed by a strategy formulated by the algorithm using N = x:

$T_t[i, N=x] = T_f[i, N=x] + T_e[i, N=x]$

$A_t[N=x]$ ... denote the average query response time over all m queries, such that each query is processed by a strategy formulated with the value of x for the parameter N:

$$A_t[N=x] = \sum_{i=1}^{m} T_t[i, N=x] / m = A_f[N=x] + A_e[N=x]$$

EXECUTION AND FORMULATION DELAYS

Figure 6 shows the average execution delay, $A_e[N=x]$, for strategies formulated with N = 1, 2, ..., 7. The delays are shown in the graphical form and also in the form of a table. Recall that for any query which requires n - 1 operations, the algorithm HEURISTIC finds the minimum execution delay strategy if it is used with a parameter N such that

x	1	2	3	4	5	6	7
$A_e[N=x]$	24.79	23.76	21.20	20.96	20.36	20.36	20.36

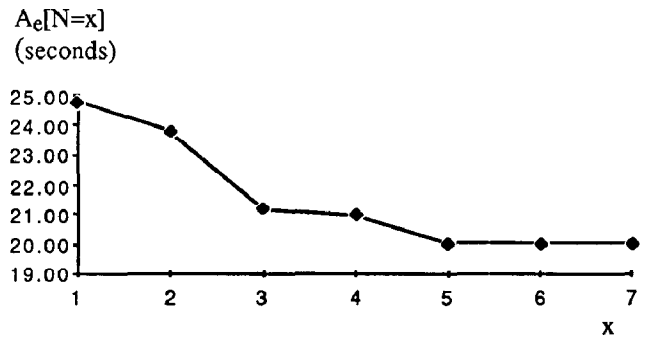


Figure 6 Average strategy execution delay, $A_e[N=x]$

$N \geq n - 1$. The modeled queries are such that the algorithm finds optimal strategies for all queries when N = 7. In other words, each of the modeled queries requires at most 7 multi-operand relational operations (joins and/or unions) for its execution.

It is evident from Figure 6 that, as expected, an increase in N results in strategies of lower execution delay. It may therefore be tempting to choose a sufficiently large N so that the algorithm finds the minimum execution delay strategies. The high strategy formulation delay, however, makes this prohibitive. Figure 7 shows the average delay to formulate the strategies as a function of N. Clearly, it is unreasonable to accept an average delay of about 26 seconds in formulating strategies which have an

average execution delay of 20 seconds. These values are shown in figures 6 and 7, respectively, for N = 7. Increasing the value of N does decrease the strategies' execution delay, but this is at the expense of increased delay in their formulation.

QUERY RESPONSE TIME

It was already stated that the modeled application includes information processors comparable to the VAX family of computers. As the formulation of strategies and modeling was performed using a VAX-785 computer system, it is reasonable to calculate the average query response time using the formulation delay derived in modeling. Figure 8 shows the total strategy formulation and execution delay, $A_t[N=x]$, for x = 1, 2, ..., 7. The figure indicates that for this application the heuristic algorithm HEURISTIC should be used with the parameter N = 3.

Let

$\hat{N}$ ... denote that value of N for which

x	1	2	3	4	5	6	7
$A_f[N=x]$	0.19	0.35	0.79	2.01	5.81	13.59	23.56

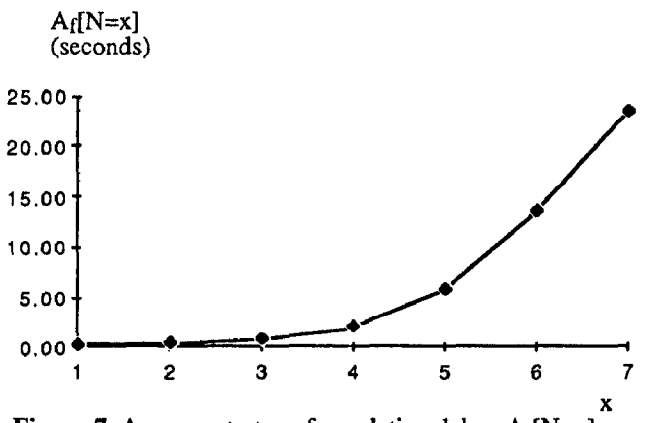


Figure 7 Average strategy formulation delay, $A_f[N=x]$

x	1	2	3	4	5	6	7
$A_t[N=x]$	24.98	23.11	21.99	22.97	26.17	33.95	43.92

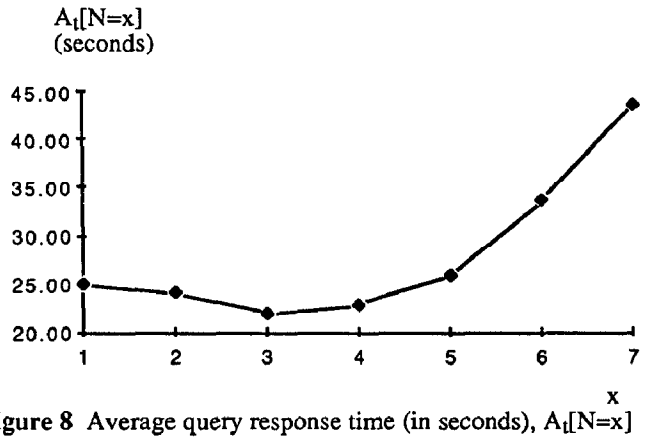


Figure 8 Average query response time (in seconds), $A_t[N=x]$

$$A_t[\hat{N}] = \min_{\{x\}} \{ A_t[N=x], x = 1, 2, ..., 7 \}$$

For this application,

$$A_t[\hat{N}] = A_t[N=3] = 21.99 \text{ sec.}$$

Using N = 3 results in the minimum query response time from all $A_t[N=x]$, x = 1, 2, ..., 7. On the average about 4 percent of the query response time is spent in the formulation of strategies while 96 percent is spent in their execution. This average response time is only 1.08 times higher than the minimum execution delay of strategies which is $A_e[N=7]$ = 20.36 seconds.

BEST ACHIEVABLE RESPONSE TIME

It has been assumed thus far that the algorithm HEURISTIC is used with the same value for its parameter N for all of the queries. This does not lead to the minimum query response time as different values of N may be best for different queries. Let

$\hat{T}_t[i]$ ... denote the minimal query response time for the query q_i, which can be achieved using the algorithm HEURISTIC. It is the minimal delay which can be achieved for the query when both the strategy formulation and execution delays are under consideration:

$$\hat{T}_t[i] = \min_{\{x\}} \{ T_t[i, N=x], x = 1, 2, ..., 7 \} ;$$

$\hat{T}_f[i]$, $\hat{T}_e[i]$ ... denote the respective formulation and execution components of the total delay $\hat{T}_t[i]$:

$$\hat{T}_t[i] = \hat{T}_f[i] + \hat{T}_e[i]$$

$\hat{T}_t[N^*]$ ... denote the average query response time such that each query q_i is processed by a strategy formulated with that value of N which leads to $\hat{T}_t[i]$, i.e.,

$$\hat{T}_t[N^*] = \sum_{i=1}^{m} \hat{T}_t[i] / m \text{ and}$$

$\hat{T}_f[N^*]$, $\hat{T}_e[N^*]$ ... denote the average delays spent in formulating and executing such strategies, respectively:

$$\hat{T}_t[N^*] = \hat{T}_f[N^*] + \hat{T}_e[N^*], \text{ where}$$

$$\hat{T}_f[N^*] = \sum_{i=1}^{m} \hat{T}_f[i] / m \text{ and } \hat{T}_e[N^*] = \sum_{i=1}^{m} \hat{T}_e[i] / m$$

For the modeled application the above delays in seconds are:

$$\hat{T}_t[N^*] = 21.52 ; \quad \hat{T}_f[N^*] = 0.53 ; \text{ and } \quad \hat{T}_e[N^*] = 20.99$$

The above average query response time, $\hat{T}_t[N^*]$ = 21.52 seconds, is the best that can be achieved with the algorithm HEURISTIC for the modeled application of the DDB. On the average, 2.5 percent of the query response time is spent in formulating strategies while the remaining 97.5 percent is spent in their execution. It should be noted, however, that to achieve this response time the optimal value of N must be known for a given query before the algorithm starts formulating its strategy. Although in reality the optimal value of N is not known, it was

already shown that when used with the specific value of N = 3, the average query response time of the modeled queries is $A_t[\hat{N}] = A_t[N=3] = 21.99$ seconds. This is only 1.02 times higher than the best possible response time which can be achieved using the algorithm HEURISTIC, which is $\hat{T}_t[N^*] = 21.52$ seconds. Consequently, when the algorithm is used with N = 3 for all queries, the average query response time, $A_t[\hat{N}] = A_t[N=3] = 21.99$ seconds, is not only close to the best achievable response time, $\hat{T}_t[N^*]$, but also to the minimum execution delay of strategies which is $A_e[N=7] = 20.36$ seconds.

GREEDY HEURISTIC

When the algorithm HEURISTIC is used with the parameter N = 1, it becomes a simple hill climbing heuristic which is similar in approach to those proposed in the scientific literature to optimize processing of distributed queries. Its average delay (i) due to the formulation of strategies can be found in Figure 6 and (ii) due to the execution of strategies it formulates in Figure 7. Its average query response time can be found in Figure 8. Although its query response time is marginally higher than that of the algorithm when used with N = 2, 3 or 4, it is doubtful that the higher complexities arising due to a search through a more complete space of strategies are worth the effort. This is even more apparent when it is realized that the space of strategies is developed using estimation techniques to predict (i) delays due to transfer of data over the network, (ii) delays due to access to the secondary storage devices and, in particular, (iii) sizes of results of relational operations.

DELAYS, OPERATIONS AND SIZE OF RESULTS

This section concludes with examining the relationship between the number of multi-operand relational operations (joins and unions) which are required to execute a query and its formulation and execution delays. The dependance of execution delays on the total size of partial results is also examined. The total size of partial results includes the size (in bytes) of all relations accessed and produced by the strategy used to derive the query result. That is, it includes the size of relations referred to by the query, any intermediate results/relations of executing joins/unions, and also the relation which is the query result. Let

Q_k ... denote the set of indices for those queries which require k operations for their execution and

m_k ... denote the number of queries which require k operations (joins and unions) for their execution. In other words, m_k is the cardinality of the set Q_k: $m_k = |Q_k|$.

Recall that there are m = 21 queries such that

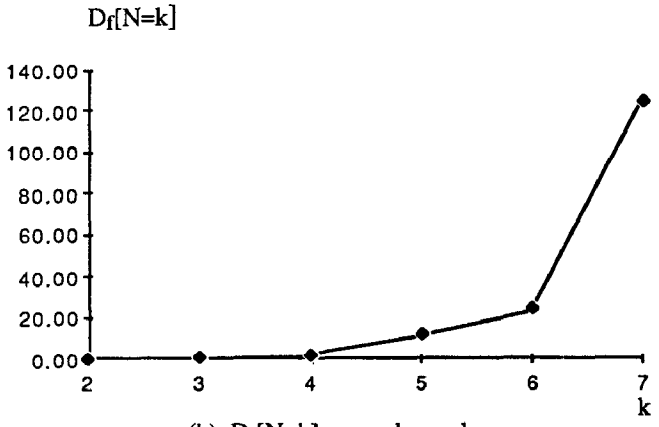
3 require 2 operations ($m_2 = 3$);
6 require 3 operations ($m_3 = 6$);
5 require 4 operations ($m_4 = 5$);
3 require 5 operations ($m_5 = 3$);
2 require 6 operations ($m_6 = 2$); and
2 require 7 operations ($m_7 = 2$),

for their execution, where operations are assumed to be joins and/or unions. Let

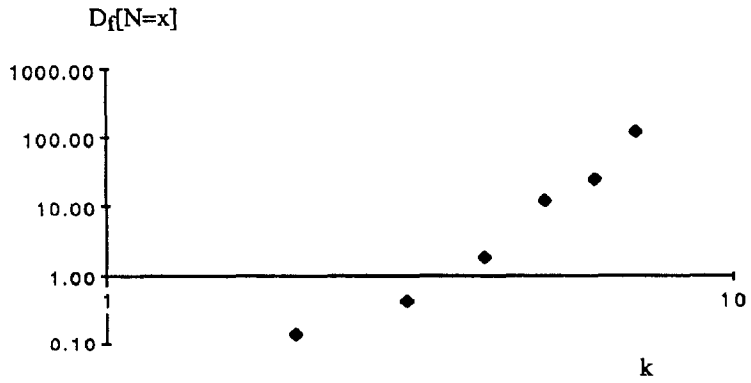
$D_f[N=k]$ ... denote the average delay, of the heuristic algorithm HEURISTIC using the parameter N = k, to formulate strategies for the m_k queries which require k operations for their processing:

k	2	3	4	5	6	7
$D_f[N=k]$	0.14	0.41	1.79	11.87	24.16	124.22

(a) $D_f[N=k]$ -- table form



(b) $D_f[N=k]$ -- regular scale



(c) $D_f[N=k]$ -- semi-log scale

Figure 9 Average formulation delay (in seconds), $D_f[N=k]$

$$D_f[N=k] = \sum_{i \in Q_k} T_f[i, N=k] / m_k \text{, where}$$

$T_f[i, N=k]$ is the delay incurred by HEURISTIC using N = k in formulating the strategy for the query q_i, $i \in Q_k$;

$D_e[N=k]$ ... denote the average execution delay of strategies, formulated by HEURISTIC using the parameter N = k, for queries which require k operations for their processing:

$$D_e[N=k] = \sum_{i \in Q_k} T_e[i, N=k] / m_k \text{, where}$$

$T_e[i, N=k]$ is the execution delay of the strategy, formulated by HEURISTIC using the N = k, for the query q_i, $i \in Q_k$.

Note that since queries q_i, where $i \in Q_k$, require k operations for their processing the algorithm HEURISTIC with N = k formulates the minimum delay (optimal) strategies for their execution. $D_f[N=k]$ and $D_e[N=k]$ are the formulation and

execution delays of those strategies. The average formulation delay, $D_f[N=k]$, is shown in Figure 9(a) as a function of the number of operations, k. It is also shown using the semi-log scale in Figure 9(b). It appears to have an exponential shape and thus empirically supports the theorem that the problem under the consideration is NP-hard [Bodorik, 1987] and that the search for optimal strategies is not desirable.[1]

k	2	3	4	5	6	7
$D_e[N=k]$	21.04	31.46	7.46	6.46	27.27	32.20

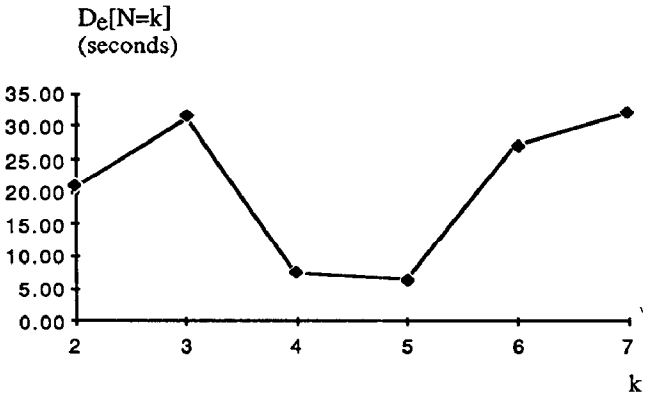


Figure 10 Execution delay $D_e[N=k]$

The average execution delay, $D_e[N=k]$, is shown in Figure 10. The strategy execution delay does not appear to be a function of the number of operations which are required to process a query. This is somewhat surprising in that one would expect that queries which require a higher number of joins and/or unions would also have a higher strategy execution delay. A brief reflection on this observation suggests a likely and also obvious explanation that the execution delay depends not only on the number of operations but also on the size of relations, which include intermediate partial results, accessed in processing the query. An examination of execution delays of optimal strategies and the total size of partial results they create supports this conclusion. Let

S[i] ... denote the total size of partial results for the query q_i when it is processed by the minimum delay strategy and also recall that

$\hat{T}_e[i]$...denotes the delay of the minimum execution delay strategy for the query q_i.

Figure 11 shows execution delays ($\hat{T}_e[i]$) and also the total size of partial results (S[i]) for each query which is assumed to be processed by the minimum delay strategy. It also shows the execution delay versus the total size of partial results using a scatter diagram in a log-log scale. A least-squares line, calculated using the standard equations

$$\sum_{i=1}^{m} \hat{T}_e[i] = m\,a + b \sum_{i=1}^{m} S[i] \quad \text{and} \quad \sum_{i=1}^{m} (\hat{T}_e[i]\, S[i]) = b \sum_{i=1}^{m} S[i]^2 ,$$

[1]Recall that the search for optimal strategies is through a complete and non-redundant space of strategies. That is, the space of strategies represents only unique strategies/sub-strategies.

i ... index of query q_i, i = 1, 2,..., 21

$\hat{T}_e[i]$ ...delay (in seconds) of the minimum execution delay strategy for q_i

S[i]...the total size of partial results produced by the minimum execution delay strategy for the query q_i

i	S[i]	$\hat{T}_e[i]$
1	562	0.29
2	3072	1.05
3	4162	1.98
4	6655	6.12
5	8112	2.56
6	4777	2.50
7	18743	4.77
8	19676	10.37
9	6660	5.59
10	5244	2.29

i	S[i]	$\hat{T}_e[i]$
11	2900	6.08
12	55350	27.44
13	135200	52.57
14	16386	10.06
15	76312	61.66
16	119000	161.59
17	88712	46.69
18	3663	2.9
19	29197	13.36
20	950	0.12
21	45675	7.06

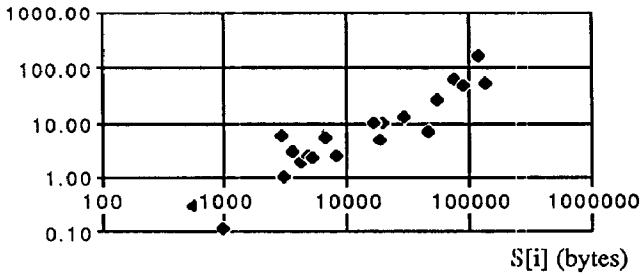


Figure 11 $\hat{T}_e[i]$ versus the total size of partial results

has a slope of b = 0.00075 and the "axis S[i]" intercept of a = -0.33. The correlation coefficient of the least square fit is r = 0.82. Whereas the execution delay did not appear to have been dependent on the number of operations required to process the query, it seems to be dependent on the size of partial results. This is rather obvious in that the total size of partial results not only affects the volume of data accessed in the secondary storage devices, but it is also likely to affect the volume of data transferred over the network. Consequently, the strategy's execution delay is likely to depend on the total size of partial results regardless of the information processor and network parameters.

The two results discussed in the for-going discussion provide support for a two-phased approach to formulating strategies to process distributed queries [Bodorik, 1987, 1988a]. The first phase finds that sequence or relational operations which minimizes the size of partial results. The second phase determines the network locations for their execution using an objective, such as minimization of the network data transfer delay, which is deemed to be appropriate for the intended environment.

It should be stressed that the results presented herein are claimed to be applicable only to the environment of the modeled application and only for the described algorithms and their assumptions. Obviously, to generalize these results further

work is necessary. It should also be pointed out, however, that evaluating heuristic algorithms is extremely difficult. Optimally, evaluation should be analytical and show that a heuristic algorithm is guaranteed to produce solutions which are within certain bounds of an optimal solution. Failing that, the evaluation should be probabilistic and show that generated solutions fall with a high probability within certain bounds of optimal solutions. To the best of the authors' knowledge none of these two approaches were used in evaluating heuristic query optimizers proposed in the scientific literature. The next approach to consider may be statistical, either through modeling or, if at all possible, through implementing the algorithm and observing its performance within an environment of a real system. As is the case with modeling reported herein, however, to generalize such results is difficult because of the high number of parameters describing the distributed environment.

6. SUMMARY AND CONCLUSIONS

This paper examined use of heuristic algorithms which are used to formulate strategies to process distributed queries by joins. A heuristic algorithm with a special property in that its overhead can be "controlled" was presented. Modeling on a test-bed of queries was used to examine whether or not there exists a trade-off between the strategy's performance and the overhead expended in its formulation. The results indicate that although there is a trade-off between the strategy formulation and execution delays, a simple greedy heuristic leads to near-optimal strategies and increasing overhead in forming a strategy appears to be only marginally beneficial. It thus confirms an often made assumption that a simple heuristic algorithm is sufficient in optimizing distributed queries.

Both the strategy formulation and execution delays were examined in relation to the number of operations and the total size of partial results. The formulation delay appears to be an exponential function of the number of joins and/or unions required to process the query and it confirms the fact that the problem of optimizing distributed queries is NP-hard. The strategy execution delay, at least for the modeled application, surprisingly does not appear to depend on the number of multi-operand relational operations used to process the query. It was shown, however, to be directly proportional to the total size of partial results. These results support the previously described two-phased formulation of strategies in which the first phase minimizes the total size of partial results.

REFERENCES

Apers P., Hevner A., Yao S.B., 1983;
"Algorithms for Distributed Queries", IEEE TOSE, Vol. SE-9, No. 1, January 1983, 57-68.

Bernstein P., et al., 1981;
"Query Processing in a System for Distributed Databases (SDD-1)", ACM TODS, Vol. 6, No. 4, Dec. 1981, 602-625.

Bertino E., 1987;
"An Evaluation of Precompilation and Interpretation in Distr. DB Management Systems", Comp. Jour., Vol. 30, No. 6, 1987, 519-528.

Black P., Luk W.S., 1982;
"A New Heuristic for Semi-Join Programs for Distr. Query Processing", Proc. of COMPSAC 82, IEEE Comp. Society's 6th Int. Conf., 1982, 581-588.

Bodorik P., 1985;
"Query Processing Strategies in a Distributed Database", Ph.D. thesis, Carleton University, Ottawa, Canada, 1985.

Bodorik P. and Riordon J.S., 1987;
"Decomposition in Processing Distributed Queries", Technical Report, School of Computer Science,Technical University of Nova Scotia, Halifax, Nova Scotia, Canada. November 1987.

Bodorik P. and Riordon, J.S., 1988a;
"Distributed Query Processing Optimization Objectives", Proc. of the IEEE Fourth International Data Engineering Conference, Los Angeles, CA, February 2-4, 1988, 320-329.

Bodorik P. and Riordon, J.S., 1988b;
"A Threshold Mechanism for Distributed Processing of Queries", Proc. of the ACM CSC '88 Conference, Atlanta, GA, Feb. 23-25, 1988, 616-625.

Bodorik P. and Riordon J.S., 1988c;
Proc of the IEEE Int. Conference on Distributed Computing Systems, San Jose, California, June 13-17, 1988, 510-519.

Carey M., Livny M., Lu Hongjun, 1985;
"Dynamic Task Allocation in a Distributed Database System", Proc. of the 1985 IEEE Conf. on Distributed Comp. Systems, 282-291.

Cellery W., Meyer D., 1980;
"A Multi-query approach to Distributed Processing in a Relational Distributed Database Management System", Distributed Data Bases: Proc. Inter. Symp. on Distributed Data Bases, Edited by Delobel C., Litwin W., North Holland Publ. Co. March 1980.

Ceri S., Pellegati G., 1984;
Distributed Databases: Principles and Systems, McGraw Hill, 1984.

Ceri S., Gotlob G., 1986;
"Optimizing Joins between Two Partitioned Relations in Distributed Databases", Journal of Parallel and Distributed Computing, Vol. 3, 1986, 183-205.

Christodoulakis S., 1983;
"Implications of Certain Assumptions in Database Performance Evaluation", ACM TODS, Vol. 9, No. 2, June 1984, pp. 173-186.

Chu W., Hurley P., 1982;
"Optimal Query Processing for Distributed Database Systems", IEEE TOC, Vol. C-31, No. 9, September 9182, 835-850.

Daniels et al., 1982;
"An Introduction to Distributed Query Compilation in System R", In Distributed Data Bases, Schneider H.J., editor, North Holland, 1982, 247-290.

Epstein R., Stonebraker M., 1980;
"Analysis of Distributed Data Base Processing Strategies", 6th VLDB Conf., Montreal, Quebec, Canada, 1980, 92-101.

Gavish B., Segev A., 1982;
"Query Optimization in Distributed Computer Systems", in Management of Distributed Data Processing, North Holland Publ. Comp., 1982, 233-252.

Hevner A., Yao S.B., 1979;
"Query Processing in Distributed Data Base Systems", IEEE TOSE, Vol. SE-5, No. 3, May 1979, 177-187.

Hevner A., 1980;
"Query Processing in Distributed Data Base Systems", Ph.D. thesis, Univ. of Minnesota, 1980.

Horowitz, 1985;
Fundamentals of Computer Algorithms, Second Edition, Computer Science Press.

Ibaraki T., Kameda T., 1984;
"On the Optimal Nesting Order for Computing N-Relational Joins", ACM TODS, Vol. 9, No. 3, Sept. 1984, pp. 482-502.

Kim W., 1985;
"Global Optimization of Relational Queries: A First Step", in Query Processing in Distributed Data Base Systems, Edited by Kim, Reiner and Batory, Springer-Verlag, 1985, pp. 207-216.
Lohman G.M., et al., 1985;
"Query Processing in R*", In Query Processing in Database Systems, Edited by Kim W., Reiner D., Batory D., Springer Verlag, 1985, pp. 31-47.
Mackert L.F., Lohman G.M., 1986a;
"R* Optimizer Validation and Performance Evaluation for Distributed Queries", Proc. VLDB, 1986, 149-159.
Mackert L.F., Lohman G.M., 1986b;
"R* Optimizer Validation and Performance Evaluation for Local Queries", Proc. ACM SIGMOD, 1986, 84-95.
Mahmoud S.A., Riordon J.S., Toth K.C., 1979;
"Distributed Database Partitioning and Query Processing", IFIP-TC-2, Venice, Italy, 1979, 32-51.
Nguyen, N.G., 1981;
"Distributed Query Management for a Local Network", Proc. 2nd Int. Conf. on Distributed Computing Systems, Paris, France, April 1981, 188-196.
Ounegbe E., Rahimi S., Hevner A., 1983;
"Local Query Translation and Optimization in a Distributed System", Proc. NCC, July 1983, pp. 229-239.
Perizzo W., 1984;
"A Method for Processing Distributed Database Queries", IEEE TOSE, Vol. SE-10, No. 4, July 1984, 466-471.
Reingold E., 1977;
Combinatorial Algorithms: Theory and Practice, Prentice-Hall, 1977.
Reuter A., 1986;
"Load Control and Load Balancing in a Shared Database Management System", Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, Feb. 1986, pp. 188-197.
Segev A., 1986;
"Global Heuristics for Distributed Query Optimization", IEEE INFOCOM'86, 388-394.
Selinger P.G., Adiba M., 1980;
"Access Path Selection in Distributed Data Base Management Systems", Proc. of the First Int. Conf. on Data Bases, Aberdeen, 1980.
Sellis T.K., 1988;
"Multiple-Query Optimization", ACM TODS, Vol. 13, No. 1, March 1988, 23-52.
Su S.Y.W., et al., 1986;
"A Distributed Query Processing Strategy Using Decomposition, Pipelining and Intermediate Results Sharing Techniques", Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, Feb. 1986, pp. 94-102.
Toth K.C., Mahmoud S.A., Riordon J.S., 1982;
"Query Processing Strategies in a Distr. DB Architecture", Distributed Data Sharing Systems, North Holland Publ. Comp., 1982, 117-134.
Wong E., 1982;
"A Statistical Approach to Incomplete Information in Database Systems", ACM TODS, Vol. 7, No. 3, Sept. 1982, 470-488.
Wong E., 1986;
"Dynamic Rematerialization: Processing Distr. Queries Using Redundant Data", in The INGRES Papers: Anatomy of a Relational DB System, Stonebraker M., Ed., Addison-Wesley Publ. Comp. Inc., 1986, 215-224.
Yu C., et al., 1982a;
"Two surprising results in Processing Simple Queries in Distributed Databases", IEEE COMPSAC, Nov. 1982, 377-384.
Yu C., et al., 1982b;
"A Promising Approach to Distributed Query Processing", Berkeley Conf. on Distr. DBs, Feb. 1982, 363-390.
Yu C., et al., 1983;
"On the design of a Distributed Query Processing Strategy", Proc. of the ACM SIGMOD Conf., May 1983, pp. 30-39.
Yu C.T., 1985;
"Distributed Database Query Processing", Query Processing in Data Systems, Edited by Kim W., Reiner D., Batory D., Springer Verlag, 48-61.
Yu C., et al., 1986;
"Adaptive Techniques for Distributed Query Optimization", Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, Feb. 1986, pp. 86-93.

APPENDIX

The appendix outlines the modeled application in terms of relations, queries and parameters describing information processors and the network. The modeled application of the DDB is claimed to be realistic and one for which available networking and processing resources are assumed to be modest. The data base is of a type needed to support an information management system for a medium size company having five geographically separated offices. The company utilizes services of a public network to which it is connected through rather slow connections ranging data transfer rates from 1200 to 9600 bits/sec. Information processors are minicomputers in the VAX 750 to 785 range. It is assumed that a CPU processing delay depends primarily on the delay caused by access to the secondary storage devices. Disks are assumed to be comparable to RA81 disk packs.

The queries are of a type which might be utilized in a management information system. They range from simple queries referencing only one or two relations to queries which are processed through a significant number of joins and/or unions. The number of modeled queries is 21. They reference relations, some of which are horizontally partitioned, with cardinalities ranging from a few tuples to tens of thousands of tuples. Relations, information processor parameters and some sample queries are shown below.

NETWORK PARAMETERS

The delay in seconds and dollar cost of transferring a unit of data (byte) between any two information processors is shown below. The dollar cost is comparable to that of the DATAPAC network. Delays are based on the assumption that information processors are connected to the network using modems with rates of either 1200 or 2400 bits/second. Only the connection between MONAXW and MONAXC is a connection with a rate of 9600 bits/second.

DOLLAR COST TO TRANSFER A UNIT OF DATA
($ / 1000 bytes)

	MONW	MONC	MONR	HQHFX	HQEDM
MONW	0.0	0.089	0.015	0.03	0.1
MONC	0.089	0.0	0.089	0.095	0.07
MONR	0.015	0.089	0.0	0.03	0.1
HQHFX	0.03	0.095	0.03	0.0	0.13
HQEDM	0.1	0.07	0.1	0.13	0.0

DELAY TO TRANSFER A UNIT OF DATA (seconds/byte)

	MONW	MONC	MONR	HQHFX	HQEDM
MONW	0.0	0.001	0.0042	0.0083	0.0083
MONC	0.001	0.0	0.0042	0.0083	0.0083
MONR	0.0042	0.0042	0.0	0.0083	0.0083
HQHFX	0.0083	0.0083	0.0083	0.0	0.0083
HQEDM	0.0083	0.0083	0.0083	0.0083	0.0

INFORMATION PROCESSOR PARAMETERS

Information processors are assumed to be of capacities comparable to those of the VAX family of computers. The secondary storage devices attached to the processors are of storage capacities and delays comparable to those of RA81 disk pack with 28 msec average seek time, 8.3 msec average latency and 17 megabits/sec data transfer rate. Data is stored in pages of 1024 bytes each. It is assumed that each 512 bytes of retrieved data incurs one disc access with one track seek (28 msec) and 8.3 msec latency. This is prorated over one byte, giving a delay of approximately 0.000035 sec/byte due to access to secondary storage devices. Further slow down by a factor of 1.5 is assumed to be due to processing of data by other applications coexisting within the time-sharing environment. The dollar cost ranges between $0.9 to $1.2 per 100,000 bytes of data transferred to/from the secondary storage devices.

INFORMATION PROCESSOR	DOLLAR COST (in $)	DELAY (seconds)
MONW	0.00001	0.00005
MONC	0.000012	0.00004
MONR	0.0000095	0.00006
HQHFX	0.000009	0.00009
HQEDM	0.0000085	0.0001

RELATIONS

Relations are listed using the following format. The relation name is followed by its cardinality enclosed in parentheses and then by a list of attribute names. Note that "_C" at the end of an attribute name stands for a "_CODE".

OFFICE (5): OFF#, OFF_NAME, LOCATION
DEPT (30): DEPT#, DEPT_NAME
OFFICE_DEPT (30): OFF#, DEPT#
DEPT_POSITION (120): DEPT#, POSITION_C, POSITION_NUM, POS_NUM_FILL
EMP (240): EMP#, POSITION_C, PAY_RATE, START_DATE, END_DATE, TERMINATE_C, EXPERIENCE_C
DEPT_EMP (240): DEPT#, EMP#
JOB_CUSTOMER (10): JOB#, CUST#, JOB_NAME, JOB_LOCATION
JOB_OFFICE (10): JOB#, OFF#, HQ#
EMP_JOB (650): EMP#, JOB#, JOB_POS_C, PERCENT_TIME, START_DATE, END_DATE
JOB_PHASE (100): JOB#, PHASE_C, PROPOSE_DATE, START_DATE, END_DATE
JOB_MATERIAL (2500): JOB#, ITEM#, QTY, PHASE_C, EXP_PRICE
JOB_MANPOWER (300): JOB#, PHASE_C, POSITION_C, RATE_CHARGED, HOURS
ITEMS (7000): ITEM#, ITEM_NAME, WEIGHT, ITEM_DESCRIPTION
SUPPLY_ORDER (1000): JOB#, PO#
PO_SUPPLIER (1000): PO#, SUPPLIER#, PO_STATUS_C
PO (1000): PO#, ITEM#, QTY, ORDER_DATE, DELIVERY_DATE, PRICE
PO_RCVD (7000): PO#, ITEM#, QTY, DATE
QUOTES (10000): ITEM#, SUPPLIER#, QTY, QUOTE_PRICE, DATE
SUPPLY (10000): ITEM#,SUPPLIER#
SUPPLIER (50): SUPPLIER#, SUP_NAME, ADDRESS, STATUS_C
CUSTOMER (10): CUSTOMER#, CUST_NAME, ADDRESS, STATUS_C

QUERIES

Three out of the 21 modeled queries are shown in English and also in their conjunctive normal form.

Query 1: "List all currently employed programmers in offices MONAXW and MONREAL. List employee, department and office numbers and also the department name."

```
(EMP.POSITION_C      = "PGM"              ) AND
(OFFICE_DEPT.DEPT#   = DEPT_EMP.DEPT#    ) AND
(DEPT_EMP.EMP#       = EMP_EMP#           ) AND
(DEPT_EMP.DEPT#      = DEPT.DEPT#         ) AND
(OFFICE_DEPT.DEPT#   = DEPT.DEPT#        )
```

Target attributes: EMP.EMP#, DEPT.DEPT#, OFFICE_DEPT.OFF#, DEPT.NAME

Query 2: "Find employees who work on a job with a job number "JOB02". List employee, department and office numbers and also the department name."

```
(EMP.EMP#        = EMP_JOB.EMP#  ) AND
(DEPT_EMP.EMP#   = EMP.EMP#      ) AND
(DEPT.DEPT#      = DEPT_EMP.DEPT#) AND
(EMP_JOB.JOB#    = "JOB02")
```

Target attributes: DEPT_EMP.EMP#, DEPT.DEPT#, DEPT.NAME, OFFICE_DEPT.OFF#

Query 3: "Find jobs worked on by employees for a given department with the number "DEPT2". For each employee list the job numbers currently worked on, job phases, job position code, percent time and start and end dates. Also list the office number for the office managing the job, employee numbers and the department name."

```
(EMP_JOB.EMP#    = DEPT_EMP.EMP#     ) AND
(DEPT_EMP.DEPT#  = DEPT.DEPT#        ) AND
(EMP_JOB.JOB#    = JOB_OFFICE.JOB#   ) AND
(DEPT_EMP.DEPT#  = "DEPT2")
```

Target attributes: DEPT.DEPT_NAME, DEPT_EMP.EMP#, JOB_OFFICE.JOB#, EMP_JOB.JOB_POS_C, EMP_JOB.PERCENT_TIME, EMP_JOB.START_DATE, EMP_JOB.END_DATE