ANALYSIS OF DISTRIBUTED DATA BASE PROCESSING STRATEGIES

by

Robert Epstein

Memorandum No. UCB/ERL M80/25

14 April 1980

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Research sponsored by the Air Force Office of Scientific Research under Grant 78-3596 and the Army Research Office under Contract DAAG29-79-C-0182.

ANALYSIS OF DISTRIBUTED DATA BASE PROCESSING STRATEGIES

by

Robert Epstein

BRITTON-LEE, INC.

ALBANY, CA.

and

Michael Stonebraker

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

UNIVERSITY OF CALIFORNIA

BERKELEY, CA.

ABSTRACT

In this paper we report on query processing experiments that were performed in one distributed data base environment. In this environment we compared the strategy produced by a collection of algorithms on the basis of number of bytes moved. Among other conclusions we found that limited search algorithms do not perform very well compared to algorithms which exhaust all possible processing plans.

I INTRODUCTION

There have been a large number of strategies proposed for processing high level language data base commands. For centralized data bases these include [BLAS76, GRIF79, STON76, WONG76, YOUS78]. Moreover, there have recently been attempts to extend such algorithms to handle distributed data bases [BERN79, EPST78, GOOD79, HEVN78a, HEVN78b, WONG77].

All the above algorithms attempt to decompose a query, Q, into a collection of subqueries, Q1,...,Qn in such a way that the summation of the costs of the individual subqueries is minimized. In general, the collection of subqueries is at least partially ordered in the sense that Qi uses the result of query Qj for some $j < i$. As a result the cost of query Qi depends on the size of the result produced by query Qj. We now do an example to illustrate this issue.

Consider the three relations

SUPPLIER(S#, SNAME, SLOCATION)
PARTS(P#, PNAME, PSIZE)
SUPPLY(S#, P#, QUANTITY)

with the obvious interpretation, each on a different site in a distributed data base. Consider the query to find the names of suppliers who supply bolts, which is expressed in QUEL [HELD75] as follows:

```
RANGE OF S IS SUPPLIER
RANGE OF P IS PARTS
RANGE OF Y IS SUPPLY
RETRIEVE (S.SNAME) WHERE
  S.S# = Y.S# AND
  Y.P# = P.P# AND
  P.PNAME = "bolts"
```

Clearly, we can restrict PARTS to a temporary relation TEMP1 containing only the part numbers for those parts named bolts and pay no transmission cost at all. As a result we are left with the query:

```
RANGE OF P IS TEMP1
RANGE OF S IS SUPPLIER
RANGE OF Y IS SUPPLY
RETRIEVE (S.SNAME) WHERE
  S.S# = Y.S# AND
  Y.P# = P.P#
```

To process this resulting query, one possible tactic would be to move TEMP1 and SUPPLY to the site where SUPPLIER exists and process the above query at that site.

Alternately, we could perform the query:

```
RANGE OF P IS TEMP1
RANGE OF Y IS SUPPLY
RETRIEVE INTO TEMP2(Y.S#) WHERE
  Y.P# = P.P#
```

by moving TEMP1 to the site of SUPPLY. Subsequently, we can execute

```
RANGE OF T2 IS TEMP2
RANGE OF S IS SUPPLIER
RETRIEVE (S.SNAME) WHERE
  S.S# = T2.S#
```

by moving TEMP2 to the site where SUPPLIER exists. This second approach seems a sure winner, at least in bytes moved, because TEMP2 is guaranteed to be smaller than

SUPPLY.

However, there are other possible ways to decompose the query. For example, we could process the query

```
RANGE OF S IS SUPPLIER
RANGE OF Y IS SUPPLY
RETRIEVE INTO TEMP3(S.SNAME, Y.P#) WHERE
  S.S# = Y.S#
```

by moving either SUPPLIER OR SUPPLY. Thereafter, we could process

```
RANGE OF T IS TEMP1
RANGE OF T3 IS TEMP3
RETRIEVE (T3.SNAME) WHERE
  T3.P# = T.P#
```

by moving either TEMP1 or TEMP3. It is unlikely (but possible) that this option will be attractive. In general, the choice of the best strategy depends crucially on the sizes of the intermediate relations {TEMP1, TEMP2, TEMP3}.

We can now categorize query processing algorithms along the following four dimensions.

1) What information is used to estimate the sizes of intermediate relations?

2) What technique is used to perform the estimation?

3) Are all possible query processing plans evaluated or only a subset?

4) Is plan evaluation done at run time or prior to run time?

We now discuss each dimension in turn.

1) Information available to estimate sizes of intermediate results

Query processing algorithms assume certain information is available for estimating the size of intermediate results. For example, INGRES assumes that only the cardinalities of the relations in question are known [WONG76, STON76]. Alternatively, [GRIF79, GOOD79] assume this information and also the number of unique values in each field. Lastly, [HEVN78a, HEVN78b] require all this information plus the proportion of any relation that will participate in a join with any other relation.

There is a continuum of possible information from none at all to perfect information, i.e. the knowledge of the exact size of any intermediate relation that might be constructed.

2) Estimating technique

Based on some amount of information each algorithm must then estimate the size of any intermediate temporary relation. For example, the algorithm in [WONG76] suggests using a "worst case" estimate. Hence, the result size of any join is estimated to be the size of the cross product of the two participating relations. In particular, if R1 with N1 tuples is joined to R2 with N2 tuples, then the size of the result is estimated to be N1*N2 tuples. Alternately,

[GRIF79] suggests an estimating technique of "worst case/10". For example, the above join would be estimated at N1*N2/10 tuples.

3) Limited versus exhaustive search

Many algorithms (e.g. [WONG78, GRIF79, GOOD79]) attempt to limit the number of choices examined. A popular tactic is to evaluate the possible processing steps which can be taken next and choose the best one without concern for the ultimate consequences of the action taken.

These algorithms never select plans which result from applying locally expensive tactics. As such, they will not find a global optimum solution unless the global optimum has the property that each step is the locally most advantageous thing to do.

Alternately, one can perform an exhaustive search through all possible plans to find the best one.

4) Static versus dynamic decision making

The algorithms in [GRIF79] suggest that a complete query processing plan be constructed in advance (in fact, at compile time). Because the actual result sizes are not available until run time, each decision is based on estimates of result sizes. Consequently, estimation errors will be propagated to subsequent decision steps. Alternately, one can do dynamic decision making. Here, a step is not performed

until the result of the previous step is completed. Exact information from the previous step can be used in selecting the next step.

Each of these dimensions appears to be very important. The amount of assumed information is significant because a data base system must keep reasonably accurate estimates for needed statistics. Obviously, the cost of doing so increases with the complexity and number of such statistics. To the extent that such statistics allow better strategies they are clearly valuable.

The estimating technique is significant because it is crucial for static decision making where incorrect estimates propagate through the remainder of the selection of a query processing plan. Even for dynamic decision making, poor estimates may result in an inferior strategy being adopted.

Many algorithms suggest limited search to avoid paying the CPU overhead of examining all possible query processing possibilities. The general feeling is that one obtains a "good" strategy anyway. Consequently, the extra cost of examining all possible plans is not offset by the savings inherent in a significantly better strategy.

Lastly, generating a query processing strategy at compile time has the obvious benefit of saving run time overhead. Again the question arises "Is the penalty for possibly inaccurate estimates persisting throughout the selection of the

plan small enough to justify the savings in overhead?"

In this paper we present experiments concerned with these questions. Hence, in Section 2 we indicate the environment chosen and the experiments performed. Then in Section 3 we give the results of our experiments and interpret some of the rather startling results. Lastly, Section 4 contains our conclusions.

II THE EXPERIMENTAL TECHNIQUE

We are interested in exploring query processing in a distributed environment and have suggested a collection of algorithms in [EPST78]. In order to present and interpret our results we must briefly review these algorithms. The environment assumed is a distributed data base system where each site contains a subset of the tuples in each relation, i.e. a fragment [ROTH77]. The placement of tuples is controlled by a distribution criteria which specifies the way a relation is partitioned among the sites.

For example the SUPPLY relation might be distributed as:

```
SUPPLY where P# < 500   AT site_1
SUPPLY where P# >= 500  AT site_2
```

Beginning with the query

```
RANGE OF R1 is REL_1
        .
        .
        .
```

```
RANGE OF Rn is REL_n
RETRIEVE TL(R1,...,Rn) WHERE Q(R1,...,Rn)
```

one first runs an algorithm "choose piece". This algorithm chooses a subset of the variables R1,...,Rn, say R1,...,Rk, and then breaks the query into two pieces. The first piece is directly executed, while the second is presented to the algorithm to be possibly split further.

The two pieces are respectively:

```
1) RANGE OF R1 is REL_1
        .
        .                                  (1)
        .
   RANGE OF Rk is REL_k

   RETRIEVE INTO TEMP TL'(R1,...,Rk) WHERE Q'(R1,...,Rk)

2) RANGE OF T IS TEMP
   RANGE OF Rk+1 IS REL_K+1
        .
        .                                  (2)
        .
   RANGE OF Rn IS REL_n

   RETRIEVE TL''(T,Rk+1,...,Rn) WHERE Q''(T,Rk+1,...,Rn)
```

Here, TL' consists of those fields which are actually in the ta ⌣et list of the query as well as those which appear in Q''. Moreover, Q' consists of those terms in Q which involve only variables R1,...,Rk. TL'' contains those fields desired as output while Q'' contains all clauses of Q not in Q' and has tuple variables changed to reflect the presence of TEMP.

The previous section contained two examples of "choose

piece" at work splitting a query on (SUPPLIER, SUPPLY, TEMP1). The first split was into the pair {(TEMP1, SUPPLY) and (TEMP2, SUPPLIER)} while the second was into {(SUPPLIER, SUPPLY) and (TEMP3, TEMP1)}.

The first piece of the query is executed by choosing one of the relations, Rp and the number of processing sites, L. Then, all tuples in relations designated by R1,...,Rk except Rp are moved in such a way that they are present at all L processing sites. The relation Rp is left fragmented among the L sites. Consequently, each of the L sites can run the query (1) above to produce a TEMP at its site. The composite of all L TEMP relations is the desired result.

To illustrate the above processing, suppose SUPPLY is split between sites 1 and 2 as noted above while PARTS is at site 3 and SUPPLIER at site 4. The example query is to find the names of bolt suppliers as in Section 1. As noted earlier we can restrict PARTS to the relation TEMP1 at site 3 and pay no data movement. Thereafter, we might choose to process the portion of the query involving TEMP1 and SUPPLY, i.e.

```
RANGE OF P IS TEMP1
RANGE OF Y IS SUPPLY
RETRIEVE INTO TEMP2(Y.S#) WHERE
  Y.P# = P.P#
```

If we choose L = 2 and Rp = SUPPLY, then we must move TEMP1 from site 3 to sites 1 and 2. Then, we can run the above query at these sites each of which will produce a fragment

of the result, TEMP2.

A complete set of tactics for "choose piece", Rp and L is given in [EPST80].

The above algorithm is oriented toward the possibility that L>1. One environment in which a distributed data base system will run is that of several machines connected over a local network which allows efficient broadcasting [ROWE79]. In such an environment one usually wishes to choose L > 1 [EPST80]. Moreover, the algorithm makes no use of "semi-join" tactics [BERN79]. Such tactics may be very attractive in some environments.

To study this algorithm under several alternate scenarios, we coded a simulation program to calculate the cost of various possible plans. This program used the following assumptions:

1) The cost function to be minimized is the number of bytes of data moved. Although it is argued in [EPST78] that this is not always the sole parameter of interest, it is an important one which is easily measured.

2) T istributed environment is a collection of nodes connected by a "contention net" network [METC76, ROWE79]. Hence, the cost to send data to all sites is equal to the cost to send it to any site. Such an assumption closely models the ETHERNET, COCANET or other local network. The

purpose of this assumption is to limit the number of parameters in the study. It is shown in [EPST80] that except in very unusual circumstances: (1) L > 1 and (2) the amount of data moved is independent of the initial distribution of fragments and independent of the number of processing sites (L above). Hence, these parameters will not appear in the simulation.

3) Only retrieval commands are considered. In INGRES [STON76, WONG76] all updates are turned into retrievals followed by lower level processing. The slight extensions required in a distributed environment are given in [EPST80]. Hence, it is appropriate to investigate retrievals.

4) The commands to be processed may involve multiple relations. If so, the relations are joined by an equi-join of the form

relname_1.domain_name = relname_2.domain_name_2

Moreover, it is assumed that the two domain names are 4 bytes wide and do not appear anywhere else in the command.

The purpose of this assumption is to simplify the bookkeeping concerning how wide partial answers are.

5) The cardinality of each relation involved in a command is varied over the set of values {10 tuples, 100 tuples, 1000 tuples}. Moreover, each relation can either have fields present in the target list or not. If a relation has fields

in the target list, it is assumed to contribute 10 bytes per tuple to the answer to the query. Notice that the results which we will document scale in the obvious way to other relation sizes. Hence, the important point is the 100 to 1 range that is examined.

6) The actual size of any possible temporary relation that might be constructed is available. Algorithms use this information in various ways.

As a result of the above assumptions, the simulation program accepts as input the following:

1) the number of relations involved

2) a graph of the way the joining terms interconnect the relations

3) the true size of any possible join which might be performed

The program then is run for 6**n different combinations of relation sizes and presence or absence of a contribution to the target list, where n is the number of relations involved.

This results from:

3 relation sizes per relation

2 widths of contribution to the target list per relation

In our simulation we concentrated on queries spanning 4 relations; hence there are 6**4 or 1296 cases to consider. In fact, 81 of these cases have a null target list and are uninteresting. Consequently, we are left with 1215 to process.

Our strategy was to choose benchmark queries along with data for the sizes of all possible joins and then to run through all 1215 test cases. For each case we computed the cost, in bytes moved, of the query processing plan produced by 14 different algorithms. They correspond to every reasonable combination of the following options.

-information available-

i1) perfect information

Here we assume that the algorithm has access to the actual size of any temporary which it might wish to create and can use this information in its cost calculations.

i2) size of relation plus a uniqueness indicator

Here we assume that the algorithm must estimate the size of any temporary result and has at its disposal the size of the participating relations plus one bit per field indicating whether each row contains a unique value or not (or something close). Should an algorithm wish to estimate the maximum size of any join, it can do so as shown in Table 1.

Operation	Estimated Result Size
R1 join R2 (both joining fields non unique)	N1*N2 tuples
R1 join R2 (joining field in R1 unique)	N1 tuples
R1 join R2 (joining field in R2 unique)	N2 tuples
R1 join R2 (both joining fields unique)	min(N1, N2) tuples

Maximum Size Estimates

Table 1

-estimating technique-

e1) Any temporary result is estimated to be the maximum size from Table 1.

e2) Any temporary result is estimated to be the maximum size divided by 2.

e3) Any temporary result is estimated to be the maximum size divided by 10.

-limited versus exhaustive search-

l1) Exhaustive search is performed. In particular, we iterate over all possible ways that a piece can be chosen by "choose piece". For each such piece we then iterate over all possible ways the second piece can be processed. Consequently, exhaustive search evaluates all possible splits and then chooses the plan with lowest estimated total cost. The

best choice for Rp is always the relation with largest cardinality. Hence, this is the one selected.

The number of plans which exhaustive search examines is:

$$e(1) = 1$$

$$e(2) = 2$$

$$e(n) = \sum_{i=2}^{n} \left\lceil \begin{matrix} n \\ i \end{matrix} \right\rceil * e(n-i+1)$$

where "n" is the number of variables in the query. For each plan, the number of variables in the first piece, i, is chosen and then the remaining query requires exhaustively examining all combinations of the remaining n - i + 1 variables. The number of actual choices is limited by the user's query, i.e. by the interconnections of the variables in the graph. All possible combination must be tried. For example, the number of plans evaluated for a four variable query is 29 and for a five variable query 336. Notice that many of the choices may not have a clause in piece 1 of the split query. As such they are degenerate and need not be considered. Hence, the real number of possibilities examined is much less than the maximum amount possible.

12) Limited search is performed. For any query M, limited search estimates the cost of any split into pieces M1 and M2 as:

cost(M) = cost(M1) + cost(M2)

The cost of M1 is computed as the number of bytes moved.

Moreover, the cost of M2 is computed AS IF M2 IS NOT SPLIT FURTHER. The lowest cost split is the one selected.

Exhaustive search would look through all possible ways to split M2 in order to find a minimum cost first split, whereas limited search does not perform this "look ahead". Therefore, the number of plans which limited search examines is:

$$e(n) = \sum_{i=2}^{n} \left\lceil \begin{matrix} n \\ i \end{matrix} \right\rceil$$

For example, it only needs to examine at most 11 cases for a four variable query and 26 for a five variable query.

-static versus dynamic decision making-

s1) Static decision making is performed. Here, a complete access plan is found prior to run time. Exhaustive search is run once to find the best expected plan. Limited search, on the other hand, finds a choice for M1 and the expected size of the resulting TEMP. Then, it is run on M2 and must select a piece to process using only its initial information and the expected size of TEMP. The algorithm is run until a piece is left which cannot be split.

s2) Dynamic decision making is performed. Either exhaustive search or limited search is run to find a first piece to process. Then, the query M1 is run and the ACTUAL SIZE of the TEMP produced is available. Either algorithm is then rerun on the query M2 with this extra information.

The actual algorithms which we tested are the following:

1) (perfect information, exhaustive)

This algorithm examines all possible ways to process the query and has perfect information. Hence, it must necessarily find the optimum plan. Of course, this information would never be available in practice; rather this answer represents the best which could be done and is reminiscent of the OPT algorithm for buffer management [MATT70].

2) (limited information, maximum size, exhaustive, static)

The number of bytes moved by an algorithm which enumerates all possible plans is found. However, this algorithm ESTIMATES the size of any TEMP relation from Table 1. Moreover, this algorithm finds a complete access plan in advance of running any pieces of the query so it can obtain no feedback on the actual size of any relation that it created.

3) (limited information, maximum size/2, exhaustive, static)

This algorithm is the same as algorithm 2) except that it estimates the size of any TEMP relation as the size from Table 1 divided by 2.

4) (limited information, maximum size/10, exhaustive, static)

This algorithm is the same as 3) above except it uses a fac-

tor of 10 instead of a factor of 2.

5) (limited information, maximum size, exhaustive, dynamic)

This algorithm behaves like 2) in many ways. It enumerates all possible decisions to find the best processing strategy. Moreover, it uses the same estimating procedure as that used by algorithm 2). Then, it performs the first step of the plan and obtains the ACTUAL size of the TEMP relation which it created. Subsequently, it uses this additional information to create a new plan for processing piece 2.

It is true that algorithms 2) and 5) will choose the same initial piece to process. However, 5) obtains the true size of TEMP while 2) must use its estimate. Hence, subsequent steps may diverge between the two cases.

6) (limited information, maximum size/2, exhaustive, dynamic)

This is the same as 5) except estimates are set to be the maximum size divided by 2.

7) (limited information, maximum size/10, exhaustive, dynamic)

This is the same as 6) except a factor of 10 is used.

8) -14) (limited search)

Algorithms 1) to 7) were repeated for a search strategy

which used the limited search tactic mentioned above.

In the next section we present a sampling of our results.

III EXPERIMENTAL RESULTS

Limited and exhaustive search will differ in the number of cases considered only if the number of relations involved in a query is four or more. Hence, we consider the following representative four relation command:

```
RETRIEVE (TL)
WHERE   R1.e = R2.f and
        R2.g = R3.h and
        R3.i = R4.j
```

(3)

Notice that (3) uses pairwise natural joins as the qualification. In the target list, a 10 byte field from each relation can optionally be present. As noted earlier, the presence of such a field is a simulation variable.

Table 2 indicates the two sets of test data which we use.

	A test data	B test data
R1 join R2	10	100
R2 join R3	20	200
R3 join R4	5	2

Number of Tuples in the Various Joins

Table 2

For example, using the A test data, if one chooses to process:

RETRIEVE INTO TEMP(R3.e) WHERE R3.f = R4.j

then TEMP will be a relation containing 5 tuples.

These join sizes have been assumed constant regardless of the sizes of R1,...,R4. In this way we are implicitly varying the percentage of the cross product which remains in the join between 100 percent and .0002 percent.

For both test situations we assume that the joining fields are unique values. Consequently, maximum size estimates were obtained from r w 4 of Table 1.

We now present some of our results for the A test data. First, we treat the issue of limited versus exhaustive search with perfect information by comparing algorithms 1 and 8. Both strategies will find a query processing plan for each of the 1215 cases examined. In Figure 1 we present 100 of the 1215 points sorted in descending order of the difference in magnitude f bytes moved between the two algorithms. The first 20 cases show an order of magnitude difference between limited and exhaustive search!

Figure 2 summarizes limited versus exhaustive search in a

different way. Here, we plot the percentage of cases in which limited search performs within a given factor of exhaustive search. Note that 59 percent of the time the two strategies yield the same algorithm and achieve the same cost (relative performance of limited search is 0 percent worse). However, in only 75 percent of the cases does limited search come within a factor of two (relative performance 100 percent worse) of exhaustive search. Lastly, in only 90 percent of the cases can it come within a factor of three (relative performance 200 percent worse).

In order to intuitively understand what is happening, we examine in detail the worst of the 1215 cases. This case corresponds to relations R1, R3 and R4 having cardinalities of 1000 and target list sizes of 10; and R2 having a cardinality of 10 and no target list. In that case, the cost to perform the query using exhaustive search was 380 bytes. The cost for limited search was 14,200 bytes. Exhaustive search broke the query into three, two variable pieces. First, R2 was joined to R3; then the result was joined to R4 and finally the second result was joined to R1 producing the answer. On the other hand, limited search broke the query into two pieces, R1 joined with R2 followed by the remainder of the query as one piece.

In more detail exhaustive search creates the following algorithm.

step	procedure	cost in bytes moved
join R2 and R3	move R2	80
join result and R4	move result	180
join result and R1	move result	120
		380

In order for limited search to find the optimal strategy, it would have to find that R2 joined to R3 was the first piece to process. The cost for that split would be

step	procedure	cost
join R2 and R3	move R2	80
join result to R1 and R4	move result and R1	180 + 14000
		14260

However, the strategy it chose was

step	procedure	cost
join R3 and R4	move R4	14000
join result to R1 and R2	move result and R2	80 + 120
		14200

Thus the minimum cost for a 2-3 split was to start with the (R3, R4) piece. Notice that by doing so, limited search does an expensive move for the first piece but leaves itself with two inexpensive moves for the second piece. Because limited search does not break down all possible ways to process the second piece until AFTER it chooses the first piece, it makes an expensive error. The heart of the problem is that it has an inaccurate estimate for the cost of

processing the second piece.

Figure 3 contains the same information as Figure 1 for algorithms 1, 5 and 6. Here, we are comparing the estimation technique. Algorithm 1 requires perfect information to operate. However, algorithm 5 makes maximum size estimates while 6) uses maximum size/2. Notice that 1) dramatically outperforms 5) but is only marginally better than 6). Presumably, this is because maximum size/2 is a much more accurate estimate of the true result size than maximum size. On the basis of this more accurate information it should be able to perform better.

We now turn to a comparison of static and dynamic decision making. Figure 4 compares the performance of algorithms 5 and 12. Notice that there are cases when dynamic decision making does significantly better than static. The overall difference, however, is not very large.

To assign an overall rating to each of the 14 algorithms mentioned previously, a single number was derived for each one. This was done by averaging the cost for all 1215 cases for query (3). This represents a very crude performance measure since it assumes that each situation has an equal probability of occuring. Table 3 shows the average performance for A and B test da .

A test data

		exhaustive	limited
1	perfect information	508	1086
2	maximum size, static	860	1070
3	maximum size/2, static	545	1257
4	maximum size/10, static	536	1058
5	maximum size, dynamic	755	927
6	maximum size/2, dynamic	537	1252
7	maximum size/10, dynamic	531	1053

B test data

		exhaustive	limited
1	perfect information	551	1036
2	maximum size, static	1017	1210
3	maximum size/2, static	781	1420
4	maximum size/10, static	774	1229
5	maximum size, dynamic	995	1128
6	maximum size/2, dynamic	746	1403
7	maximum size/10, dynamic	740	1207

Average Performance in Bytes Moved

Table 3

The results indicate that exhaustive search performs

consistently better than limited search for both sets of test data. Moreover, the performance of exhaustive search is very sensitive to the estimation procedure used. A maximum size estimate was significantly worse than either maximum size/2 or maximum size/10. This indicates that maximum size estimation is simply too pessimistic to use. On the other hand, note that maximum size is a viable tactic when limited search is employed.

Dynamic decision making tends to do somewhat better than static decision making but the difference is around 10 percent. Hence, dynamic decision making is only cost effective if it is very inexpensive.

We ran several alternate queries with similar results. In Table 4 we show values for one of them:

```
RETRIEVE (R1.a, R2.b, R3.c, R4.d)
WHERE    R1.e = R2.f and
         R2.g = R3.h and
         R3.i = R4.j and
         R4.l = R1.m
```

(4)

Here, a joining term linking R4 to R1 is added to the qualification of query (3) and we use the A test data augmented by:

R4 join R1 = 20 tuples

The last experiment which we report uses query (3) and the B test data. However, we assume that the join fields are not

A test data

		exhaustive	limited
1	perfect information	631	1340
2	maximum size, static	1259	1289
3	maximum size/2, static	724	2128
4	maximum size/10, static	724	1475
5	maximum size, dynamic	1185	1183
6	maximum size/2, dynamic	721	2128
7	maximum size/10, dynamic	721	1471

Average Performance in Bytes Moved

Table 4

unique. Hence, from Table 1 maximum size estimates will be the full cross product and very pessimistic. In addition, we include maximum size/1000 as a tactic in place of maximum size.

The results are shown in Table 5. Notice that maximum size/1000 is the only estimation technique which produces a reasonable plan.

IV CONCLUSIONS

The following conclusions hold in the environment which we have simulated:

B test data

		exhaustive	limited
1	perfect information	508	1086
2	maximum size/2, static	3300	3549
3	maximum size/10, static	3206	3319
4	maximum size/1000, static	876	1290
5	maximum size/2, dynamic	3372	3541
6	maximum size/10, dynamic	3307	3179
7	maximum size/1000, dynamic	852	1240

Average Performance in Bytes Moved

Table 5

1) limited search performs poorly

There is a dramatic difference between the quality of the plans produced by limited and exhaustive search. For four and five relation queries the cost of exhaustive search in CPU time is sure to be well worth it.

2) good result size estimates are crucial

Maximum size estimates are dramatically inferior to maximum size/2 or maximum size/10, at least for exhaustive search. Maximum size/10 (as assumed by [GRIF79]) appears to be a generally good tactic, although there are situations where

it is too pessimistic.

3) dynamic decision making is beneficial

Dynamic decision making does consistently better than static decision making; however, Because dynamic decision making has greater run-time cost, it may not be a big winner.

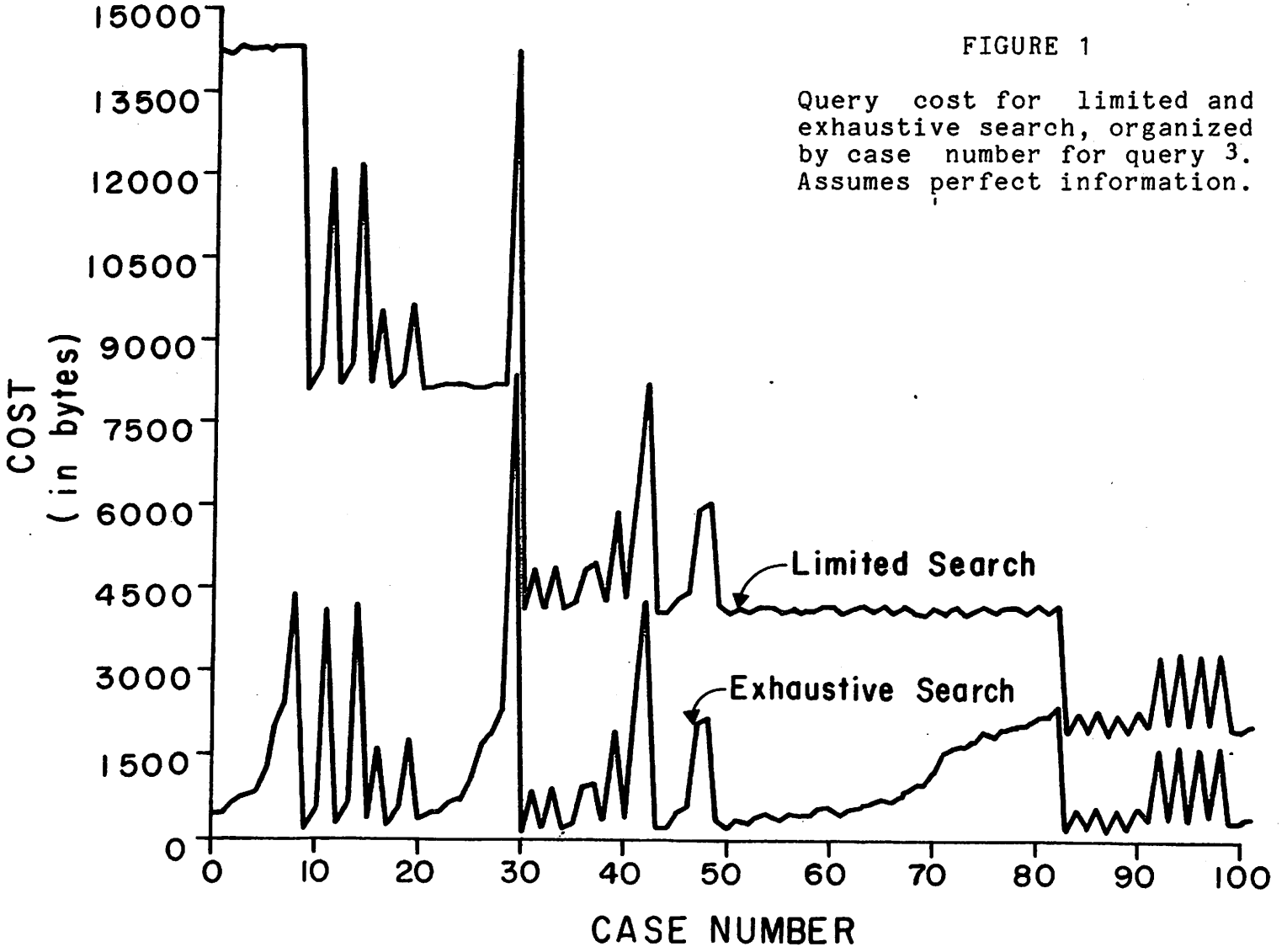


FIGURE 1

Query cost for limited and exhaustive search, organized by case number for query 3. Assumes perfect information.

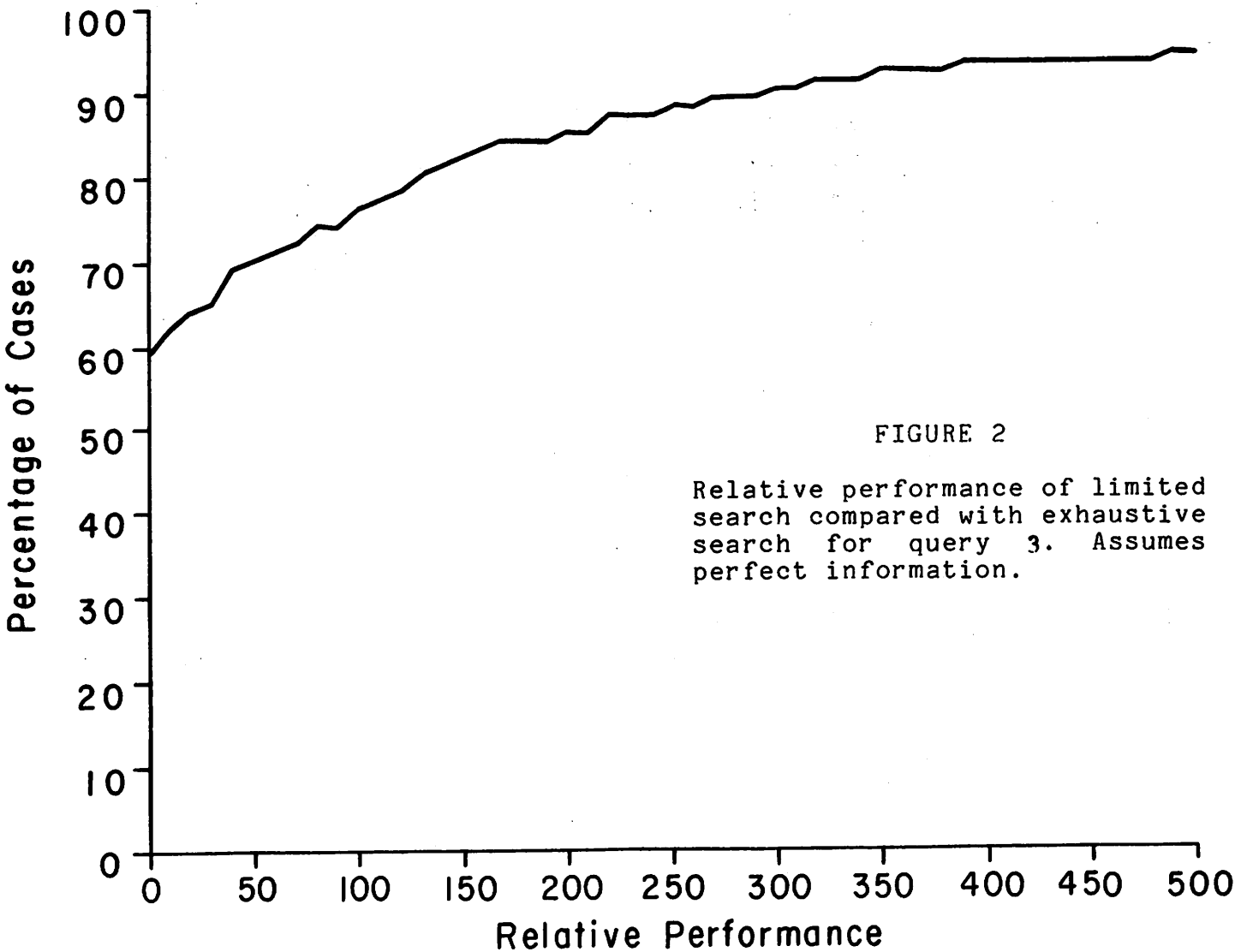
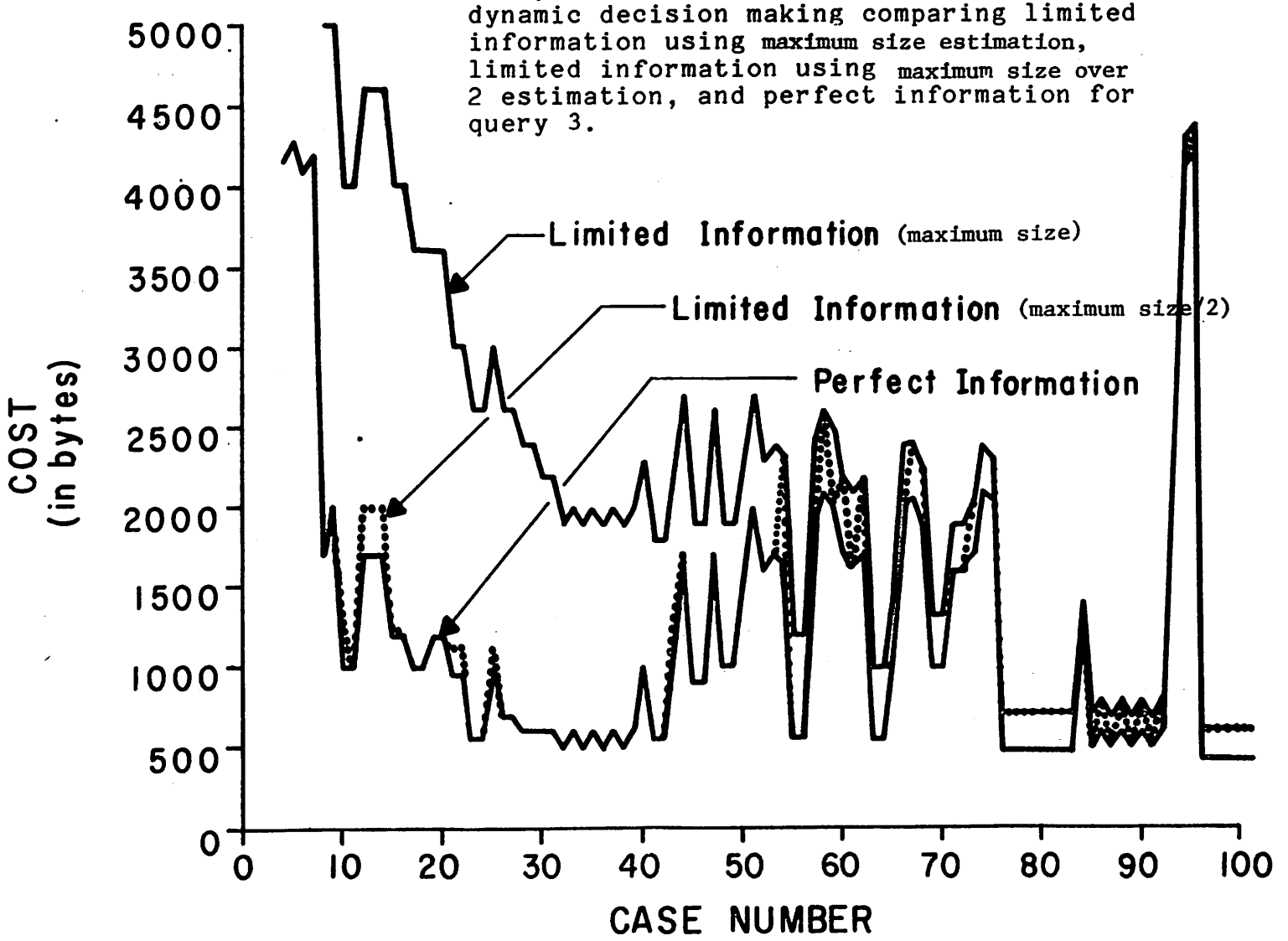


FIGURE 2

Relative performance of limited search compared with exhaustive search for query 3. Assumes perfect information.

FIGURE 3

Query cost using exhaustive search and dynamic decision making comparing limited information using maximum size estimation, limited information using maximum size over 2 estimation, and perfect information for query 3.



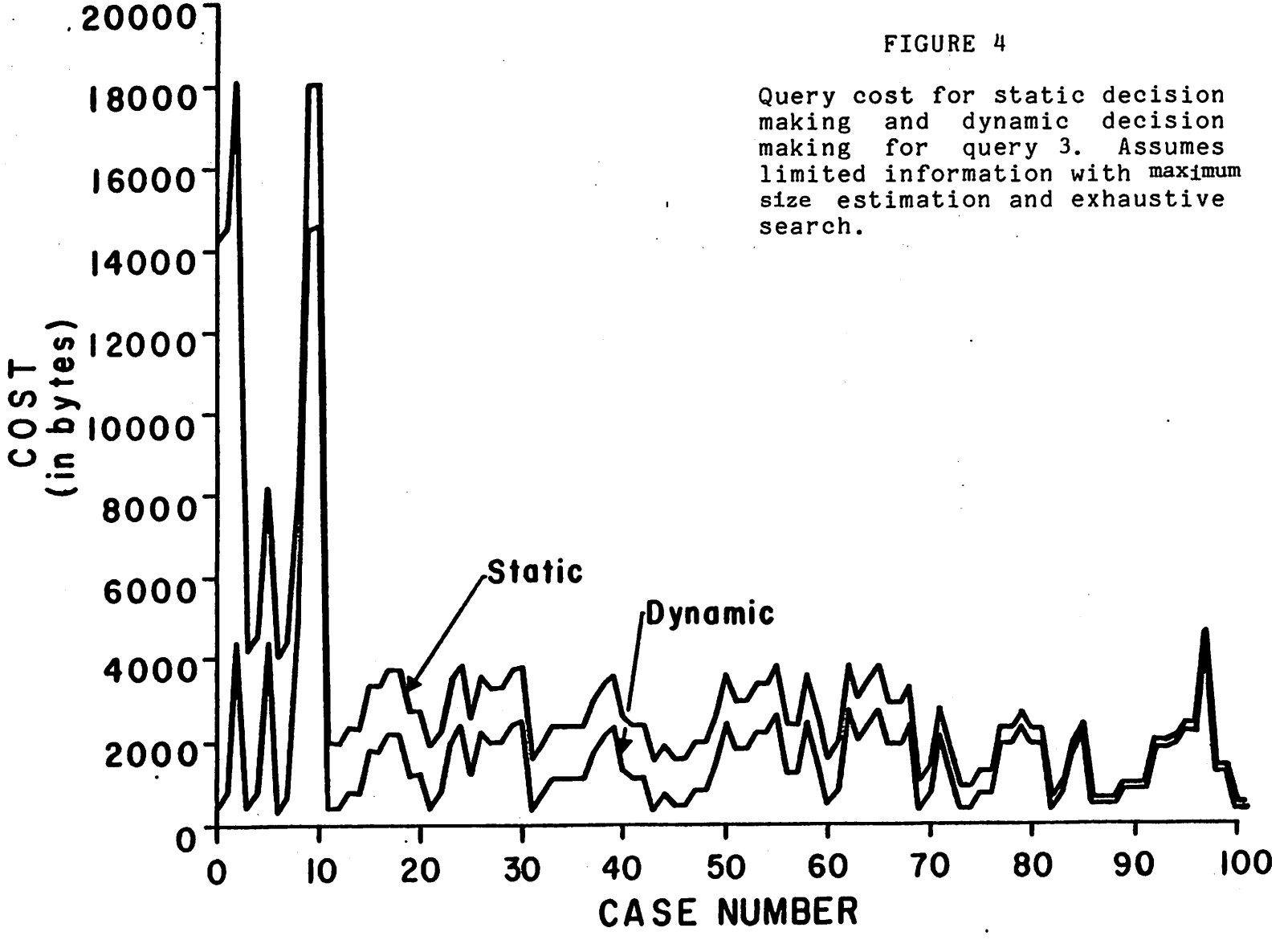


FIGURE 4

Query cost for static decision making and dynamic decision making for query 3. Assumes limited information with maximum size estimation and exhaustive search.

REFERENCES

[BERN79] Bernstein, P.A. & Chiu, D.W., "Using Semi-joins to solve Relational Queries", Unpublished paper, Harvard University, 1979.

[BLAS76] Blasgen, M.W. & Eswaran, K.P., "On the Evaluation of Queries in a Relational Data Base System", IBM Research Report RJ1745, April, 1976.

[EPST78] Epstein, R.; Stonebraker, M.; Wong, E; "Distributed Query Processing in a Relational Data Base Systems", Proc. 1978 ACM-SIGMOD Conference on Management of Data, Austin, Texas, June 1978.

[EPST80] Epstein, R., "Query Processing in a Distributed data base Environment," PhD Dissertation, University of California, Berkeley, Ca.

[GOOD79] Goodman, N., et. al., "Query Processing in SDD-1: A System for Distributed Data Bases," Technical Report 79-06, Computer Corp. of America, Cambridge, Mass., October 1979.

[GRIF79] Griffiths Selinger, P. et. all., "Access Path Selection in a Relational D ⌄ Base Management System", IBM Research Laboratory, San Jose,

California, RJ2429(33240), January, 1979.

[HELD75] Held, G.D., M.R. Stonebraker, and E. Wong; "INGRES - A Relational Data Base System," Proc. NCC vol. 44, 1975.

[HEVN78a] Hevner, A. & Yao, S.B., "Query Processing on a Distributed Data Base", Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, LBL-7953 UC-32, Lawrence Berkeley Laboratory, Berkeley, California, August 1978.

[HEVN78b] Hevner, A. & Yao, S.B., "Query Processing in a Distributed System", Dept. of Computer Science, Purdue University, August, 1978.

[MATT70] Mattson, R., et. al., "Evaluation Techniques for Storage Hierarchies," IBM Systems Journal, June 1970.

[METC76] Metcalf, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," CACM, vol. 19, no. 7, July 1976.

[ROTH77] Rothnie, J. B. and Goodman, N., "An Overview of the Preliminary Design of SDD-1: A System for Distributed Data Bases," Proc. 2nd Berkeley Workshop

on Distributed Data Bases and Computer Networks, Berkeley, Ca., May 1977.

[ROWE79] Rowe, L.A. & Birman, K.P., "Network Support for a Distributed Data Base System", Proceedings of the Fourth Berkeley Workshop on Distributed Data Management and Computer Networks, August, 1979, San Francisco, California.

[STON76] Stonebraker, M.R., E. Wong, P. Kreps and G.D. Held; "Design and Implementation of INGRES," ACM Trans. Database Systems, vol. 1, no. 3, Sept. 1976.

[WONG76] Wong, E. and K. Youssefi; "Decomposition - A Strategy for Query Processing," ACM Trans. Database Systems, vol. 1, no. 3, Sept. 1976.

[WONG77] Wong, E.; "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[YOUS78] Youssefi, K. & Wong, E., "Query Processing in a Relational Data Base Management System", Electronics Research Laboratory, UCB/ERL M78/17, University of California, Berkeley, California, March,

1978.