QUERY PROCESSING TECHNIQUES

FOR DISTRIBUTED, RELATIONAL DATA BASE SYSTEMS

by

Robert Samuel Epstein

Memorandum No. UCB/ERL M80/9

15 March 1980

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

QUERY PROCESSING TECHNIQUES FOR DISTRIBUTED, RELATIONAL DATA BASE SYSTEMS

Robert S. Epstein

ABSTRACT

The principle goal of this thesis is to develop and analyze algorithms for processing queries on distributed, relational data bases. Specifically, this thesis addresses the question: how does one determine an efficient sequence of processing steps for a query written in a non-procedural, high level query language?

A distributed data base has relations that are fragmented across one or more computer sites. The computer sites are connected by a communications network. This thesis considers two types of networks: site-to-site networks such as the ARPANET, and broadcast networks such as the ETHERNET.

There are two fundamental issues present in distributed data bases that are not found in centralized data bases. One is the communications network. As the time

Research Sponsored by the Army Research Office Grant #DAAG29-79-C-0182
and by the Air Force Office of Scientific Research Grant #78-3596

required to communicate between computer sites increases, it becomes more important to minimize the amount of information transmitted. Secondly, total processing time can be reduced by distributing the processing to many computer sites. This increases the amount of work that can be performed in parallel. These two goals, minimizing communications and maximizing parallelism sometimes conflict; that is, increasing communication costs can increase parallelism. Each of the tactics presented in this thesis are examined with respect to minimizing both the amount of network communication and total processing time.

Two fundamental tactics are developed: (1) the Fragmented Processing technique (FP technique) which is a general strategy for processing any query, and (2) the Query Splitting technique which divides a query into a sequence of subqueries. Each subquery is processed by the FP technique. The two techniques raise many important questions including: (1) How well do "locally optimal" or "greedy" algorithms compare to exhaustively examining every possible strategy? (2) How much information should be kept about each relation? (3) What method should be used for estimating the number of tuples that will qualify from a query? (4) What are the trade-offs between choosing a processing strategy once at compile time (static decision making) compared with continually reevaluating the pro-

cessing strategy during execution time (dynamic decision making)? (5) What role does a relation's physical structure play in choosing strategies?

The processing tactics are studied using both analytical techniques and a simulation program. The simulation program computes the performance of the algorithms under a variety of assumptions.

Among the conclusions are (1) "greedy" algorithms perform poorly, (2) dynamic decision making is slightly better that static decision making but potentially has a very large overhead, (3) The physical structure of a relation fragment is not useful if the fragment is moved, and (4) processing strategies are very sensitive to the accuracy of estimation.

The results of this thesis provide a framework for designing a distributed data base management system, and help to identify the many interrelated decisions that must be made when deciding how to process a query on a distributed data base.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1. Distributed Data Bases

This thesis describes a variety of techniques to process queries, written in a high level query language, on distributed, relational data bases. The word "query" will refer to both retrieval and updates. By "distributed", it is meant that portions of the data base may reside on different computer systems in a manner that is transparent to the user of the data base system.

1.2. Motivations

Recently, considerable attention has been paid to distributed data bases [LBL76, LBL77, LBL78]. One of the advantages of distributed data bases is sharing of data across different sites. For example, some computer applications are by their very nature geographically distributed. Consider a bank with many branches, or a manufacturer with many similar plants. Some personnel are concerned only with data that directly relates to their location; others are concerned with the data base as a whole. Distributed data bases allow data to be moved closer to the people most likely to need it, and allow for distributed administration of the data base.

Another advantage of distributed data bases is increased reliability due to redundancy. Failure of a single site need not affect people performing interactions local to other sites.

Still another advantage is incremental growth. A new site can be added to the system as necessary. The computing capacity at a site can be increased as required. It is even possible to split a site into two sites to accommodate growth.

Distributed data bases have the potential for increased speed. One large data base can be divided into N pieces on N computer systems. If the data base can be designed correctly, then all N systems can cooperate in solving a query in parallel. An example of such a system can be found in [STON79].

1.3. Relational Distributed Data Bases

We will only consider data bases where each site is organized using the relational [CODD70] model of data. A relation is a collection of records (called tuples in relational systems). For example, the user's view of employee data might be expressed in a relation called "employee:"

number	name	salary	manager
157	Jones, Tim	12000	199
1110	Smith, Paul	6000	33
35	Evans, Michael	5000	32
129	Thomas, Tom	10000	199
4901	Bailey, Chas M.	8377	32

A user or an application program will query the data base using a non-procedural query language such as QUEL [HELD75a]. For example, the user might ask for the names of all personnel who earn more than $10,000 and are managed by either manager number 199 or 33. This query can be expressed in QUEL as:

```
range of e is employee
retrieve (e.name)
        where
                e.salary > 10000
        and
                (e.manager = 199 or e.manager = 33)
```

The query illustrates two common operations performed on relations: restriction and projection. Their meaning can easily be explained using the above example. The statement

```
e.salary > 10000
```

restricts the employee relation to those employees whose salary is greater than 10000. The statement

```
retrieve (e.name)
```

projects the "name" domain out of the employee relation. To project means to select those domains of interest. The

list of domains being retrieved or being changed is also referred to as the target list. The terms "restriction" and "projection" will be used throughout this work. A formal definition of these two operations can be found in [DATE77].

Another example query would be to give a ten percent raise to all people who work for manager 199.

```
range of e is employee
replace e(salary = 1.1 * e.salary)
        where
                e.manager = 199
```

To the user entering the two queries above, the employee relation appears as if it were one, single relation. In a distributed data base environment, the employee relation might actually be fragmented into several employee relations located at different sites. For example, if the data base was distributed at three locations, Berkeley, Tahiti, and Paris, then the distribution of the employee relation tuples might be

number	name	salary	manager	location
157	Jones, Tim	12000	199	Tahiti
1110	Smith, Paul	6000	33	Berkeley
35	Evans, Michael	5000	32	Paris
129	Thomas, Tom	10000	199	Tahiti
4901	Bailey, Chas M.	8377	32	Paris

To the user entering the two queries, the physical location of the tuples is transparent. While there may be

other models that merit consideration this view of the data base is the only one that will be considered for this thesis.

1.4. Problems in Processing Queries

Finding an efficient sequence of local processing and data movement is the key to processing a query on a distributed data base. In the preceding examples, the problem was trivial because the query involved only one relation. For example, the retrieval could be processed as follows: The sites where data are stored are determined and the query is broadcast to those sites. Next, each site executes the query in parallel and transmits the results back to the site where the query originated. As data arrives at the originating site, it is accumulated and returned to the user.

Processing a query is very complex when multiple relations are involved and each relation is split up into fragments and stored on different sites. One obvious solution is to move all data to one site and process the query using centralized processing techniques. It is unlikely that this tactic would be optimal because it would tend to move the maximum amount of data and data transmission across a network can be slow. Furthermore, once all the data is assembled processing time can be lengthy. It is crucial to reduce the amount of data that

must be transmitted and to involve as many of the computer sites as possible in order to process the query with greater parallelism. This thesis explores techniques that will minimize data communications and increase parallel processing. It will also examine the trade-offs between the time required to communicate across a network and the "local" processing time where local processing time is defined as the time required by a site to process the query once all the data is assembled. Sometimes an increase in communication time can result in a significant decrease in the local processing time and vice-versa.

Familiarity with the techniques used on centralized data bases is important as background for understanding processing on distributed data bases. We will briefly describe the techniques used in centralized data bases.

1.4.1. Centralized Data Bases

Data base management systems utilize access methods to reduce the amount of data that must be examined to process a query on a relation. (For an introduction to access methods see [DATE77].) An access method takes a key (say the "name" field in the employee relation) and determines from the key the physical location of the tuples in the relation. If one wishes to minimize the processing time for performing a local query, then the distributed query processing tactics must consider access methods when

choosing strategies. Access methods are roughly divided into "hashing" techniques and "indexing" techniques. Hashing [KNUT73] works by taking the key value, applying a randomizing function to it, and generating an address where the tuple physically belongs on secondary storage. Index techniques [KNUT73] determine where the tuple belongs by looking up the key in an index. For a thorough treatment of the use and implementation of access methods for relational systems see [HELD75b]. This thesis will consider hashing [HELD75b], and two indexing techniques ISAM [IBM66], and B-trees [BAYE70]. Throughout this thesis, we shall call a relation well structured if it has a useful access path for the specific query under consideration; otherwise, it will be called unstructured.

Processing queries on centralized data bases has been studied at length. A study of various processing techniques can be found in [YOUS78a, BLAS76, GRIF79]. One of the most difficult problems is that of computing a "join" between two or more relations. A join is one of the three common operations performed on relations. The other two, projection and restriction, have already been introduced. Two relations with a common domain can be joined on that domain. The result is a new relation in which each tuple consists of a tuple from the first relation and the second relation which satisfy the "join predicate". For example,

consider a query expressed in QUEL involving the employee relation and department relations:

```
employee(number, name, salary, manager)
dept(number, name, store, floor, manager)

range of e is employee
range of d is dept
retrieve (e.name, d.name)
        where
                e.manager = d.manager
```

This query asks for a list of each employee and their department. In this example the "employee" relation is being joined with the "dept" relation on the "manager" domain. The join predicate is

```
e.manager = d.manager
```

When the join predicate uses equality ("="), the join is sometimes refered to as an "equi-join". A reader not familiar with joins is referred to [DATE77] for an introduction.

For the purpose of this thesis we shall be concerned with three methods for computing joins: (1) sort-merge join [BLAS76], (2) tuple substitution [WONG76, STON76], (3) reformatting [STON76, YOUS78b]. In the previous example, the join between the two relations on the "manager" domain might be computed using any of the above three techniques depending on which was the most cost effective. The sort-merge join would first sort both relations on the "manager" field (if they were not already sorted). Then

one of the relations would be read sequentially and for each tuple the other relation would be incrementally searched for a matching manager. With a minor amount of bookkeeping it is usually possible to compute the join by reading each relation only once.

Tuple substitution chooses one of the two relations for substitution. Then, one at a time, each tuple is read from the chosen relation and its value substituted into the original query. After substituting, the remaining query in this example involves only one relation. The technique differs from the sort-merge because the relations do not have to be sorted. Tuple substitution performs best when one relation is chosen for substitution and the other relation is well structured on the joining domain. If the second relation is not well structured, however, it can perform poorly.

Reformatting is a technique used in addition to tuple substitution. If the relation not chosen for substitution is not well structured, it may be more effective to reformat that relation to a useful structure before starting the substitution process.

The usefulness of the three techniques and the manner in which they apply to distributed data bases will be examined at length in chapter 4. These three techniques have been studied in great depth [YOUS78b, GRIF79] and

represent effective, general purpose techniques for computing joins. There are other techniques but for the purpose of this thesis, we shall limit the analyses to sort-merge, tuple substitution, and reformatting.

1.5. Survey of Previous Results

A survey of the significant research in distributed data bases can be found in [ROTH77b]. Four main areas of research particular to distributed data bases are (1) distributed concurrency control, crash recovery and multiple copies, (2) query processing, (3) network communications' handling, and (4) data base design. This thesis is concerned only with how to process a query on a distributed data base. Research on distributed concurrency control, crash recovery and multiple copies can be found in [RIES79, STON78, ELLI77, THOM75, LAMP76, ROTH77a, CHU76]. Network communications for distributed data bases is discussed in [ROWE79, LBL76, LBL77, LBL78]. Some issues related to distributed data base design are discussed in [LEVI75, ROTH77b].

Query processing on distributed data bases has been examined by [WONG77], [STON77], [EPST78], [HEVN78a], and [HEVN78b]. In [STON77], Stonebraker proposes an extension to tuple substitution which can be used to process an arbitrary query on a distributed relational data base. The extension was for a distributed data base version of

INGRES.

An algorithm for use on the ARPANET [ROBE70] for the SDD-1 distributed data base system build by Computer Corporation of America [ROTH77a] is proposed in [WONG77]. The algorithm considered minimizing the amount of data moved over the communication network as the primary optimization criterion.

Hevner and Yao proposed an algorithm [HEVN78a] which optimized processing for a distributed data base on an ARPANET type network with one relation per computer site and one joining domain. It was basically a limited extension of Wong's work. They later extended the algorithm [HEVN78b] to handle arbitrary queries but assumed that relations did not span more than one site.

Epstein, Stonebraker and Wong proposed algorithms [EPST78] for processing queries on two different models of computer networks - ARPANET types and broadcast networks. It also specifically dealt with relations fragmented across multiple sites. This thesis includes the work presented in [EPST78] and is a continuation of that work.

1.6. Overview

This thesis will examine a variety of techniques which are important for processing queries on distributed data bases. In the next four chapters, we shall present a

model of the distributed data base environment, the proposed tactics for processing queries, the analysis of the tactics, and finally conclusions and suggestions for further research.

Models for network communication and data organization are presented in chapter two. Only the relational model of data is considered. A user defined distribution criterion specifies the distribution of relations into fragments across the computer sites. Two models of network communication, broadcast model and site-to-site model, are considered. At the end of chapter two, the optimization criteria are presented. These include minimizing the number of messages sent on the network, minimizing the number of bytes transmitted on the network, and minimizing response time.

In chapter three, the processing strategies and estimation methods are presented. Some of the tactics are extensions of algorithms used on centralized data bases. Others are new tactics that specifically apply to distributed data bases. We will concentrate on those issues which apply specifically to distributed data bases. Several of the techniques make decisions based on estimates of how much data will result from a specific query. Chapter three describes several estimation procedures.

Chapter four presents the analysis of the tactics described in chapter three. When analyzing a query, it is frequently beneficial to break the query into subqueries to be executed in a particular order. Chapter four begins by examining subqueries which involve only one relation. Next a general algorithm (called the FP technique) is presented that can process an arbitrary query on a distributed data base. The next processing tactic determines when a query involving three or more relations should be divided into subqueries. The three tactics just described comprise the major strategies proposed for processing queries on distributed data bases. There are many specific details which are particularly important for distributed data bases. These include determining what information to transmit, how to encode joins during transmission, the movement of the result relation, and static verses dynamic decision making. Finally in chapter four, we compare the proposal of this thesis to those of [WONG77] and [HEVN78b]. Lastly, chapter 5 contains conclusions and suggestions for future research.

CHAPTER 2

PROCESSING MODELS

This chapter introduces the framework for the processing tactics. The distributed relational model is presented and the notation which will be used throughout this thesis is defined. The notion of distribution criteria is presented and the specific distribution criteria model is explained. Next two communication models are discussed and the processing model is presented. Finally, three optimization criterion are discussed.

2.1. Distributed Relational Model

The data base model consists of a data base on a known number of sites called $S_1, S_2, \ldots, S_n$. The index "j" will be used when referring to a collection of sites (e.g. S_j). The data base contains a collection of relations $R_1, R_2, \ldots, R_n$. The index "i" will be used when referring to a collection of relations (e.g. R_i). To the user of a distributed data base system relations are divided into two classes: local and distributed. A "local" relation is known only to the site where it was created and is accessible only at that site. The query processing strategy, however, treats a local relation like a distributed relation which exists at only one site. A "distributed" rela-

tion is known to all sites in the data base and can be accessed by any site. For distributed relations, the instance of relation R_i at site S_j shall be referred to notationally as R_i^j. We shall call it the fragment of relation i at site j. The syntax for creating local and distributed relations for a distributed version of the INGRES system is shown in the examples below:

```
create local supplier (snum = i2, sname = c10,
                       address = c30)
create distributed employee (enum = i2,
                       ename = c20, salary = i2, manager = i2)
```

The first example creates a local relation called "supplier" with domains snum, sname, and address. "Snum" is a two byte integer; "sname" is a ten byte character; and "" address is a thirty byte character. The second example is a distributed relation called "employee" with domains enum, ename, salary and manager.

2.2. Distribution Criteria

Distribution criteria allow the data base administrator to assign tuples to specific sites based on one or more domain values. For example, the employee relation and its distribution from chapter 1 can be expressed as:

```
range of e is employee
distributed e at
        berkeley where e.manager = 32,
        paris where e.manager = 199,
        tahiti where e.manager = 33
```

For this thesis, we shall assume that a distribution criterion applies only to a single relation and that it unambiguously maps tuples onto a single unique site. The reasoning behind such a choice can be found in [RIES78].

2.3. Communication Model

A communication model will be used to determine the cost measured in time required to send a specific number of bytes of data to one or more sites. The delay in general is a linear function such as

$$DELAY = C_0 + C_1X$$

where X is the number of bytes to be sent and C_0 and C_1 are network dependent constants. Most computer communications networks split a stream of bytes into a set of fixed size packets or messages. Thus C_1X might be better modeled as a step function but for the purpose of this thesis we will approximate its value using the above formula.

It will be shown throughout this thesis that distributed data bases frequently require the same information to be sent to multiple sites. Because of the usefulness of broadcasting to multiple sites, we will use two different models of communication costs. The models will be called "site-to-site" and "broadcast" models.

Let $C_n(X)$ be the cost to send X bytes of data to n sites. Using the "site-to-site" model, it is assumed that the cost to send a message to n sites is equal to n times the cost to send to one site.

$$C_n(X) = nC_1(X)$$

This model resembles the ARPANET [ROBE70]. The ARPANET consists of a large collection of computer sites which are connected from point to point by communication lines. If site A wishes to communicate with site B, there exists either a communications line directly connecting A and B, or A must route the message through one or more other sites to get to B. The time delay to send a message between two sites on the ARPANET depends on the route the message must take. We make the simplifying assumption that to communicate between any two arbitrary sites in the network is the same cost. A further simplification involves messages sent to multiple sites. It is assumed that the delay is linear, that is,

$$C_n(X) = nC_1(X)$$

Our model assumes that messages are not transmitted in parallel. In fact, if transmissions can occur in parallel, there are many algorithms [DALA78] for which

$$C_n(X) < nC_1(X)$$

Our second model is the broadcast model. Every site on the network listens to every message. A message can be addressed to a single site or a set of sites. The cost of communication is independent of the number of sites, thus:

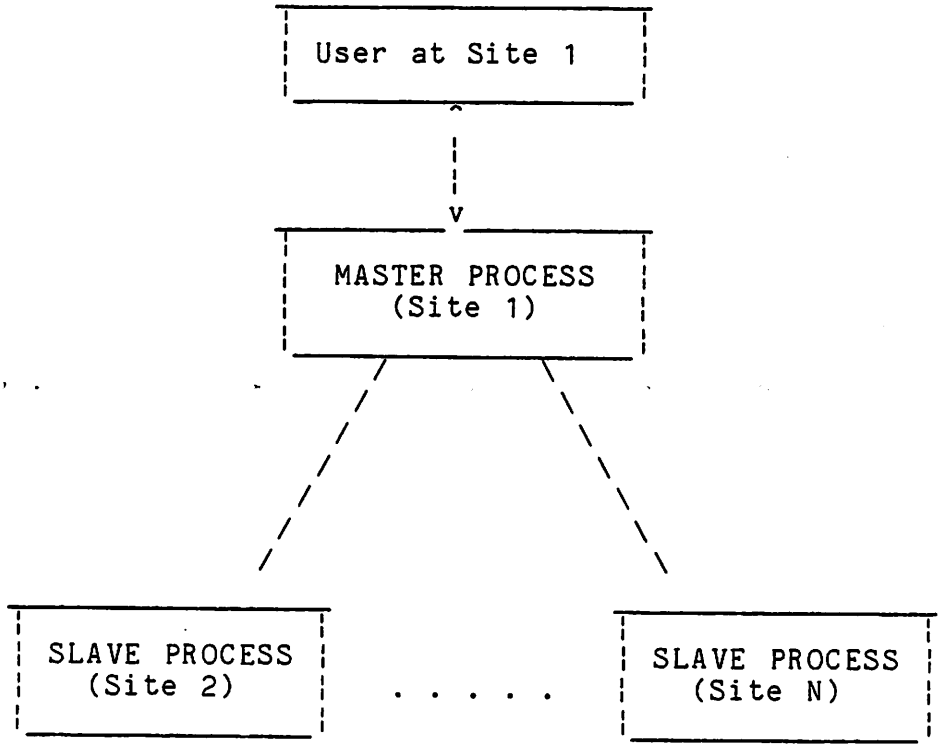
$$C_n(X) = C_1(X)$$

This model resembles the ETHERNET [METC76]. An ETHERNET consists of a single common line to which all sites are connected in parallel. It is used for local networks and can handle a one kilometer network at a speed of three megabits per second [METC76]. Our model is dissimilar from the ETHERNET in that the ETHERNET does not allow arbitrary subsets of sites to be addressed. COCANET [ROWE79], is an example of a broadcast network which also allows broadcasting to arbitrary subsets of sites.

Using the site-to-site and broadcast models we will develop algorithms that minimize the transmission cost measured in delay time. In both models we ignore the real network problems of transmission errors and retransmissions.

2.4. Processing Model

The logical organization of the processing model is illustrated below. The "master" process is a program running at the site where the "user" logs in. A "slave" process runs on each site in the data base involved in the



query. The physical communications network can be any configuration that allows all data base sites to communicate with all other data base sites. The master process directs all activity in the slaves. The slaves do not initiate any activity on their own.

The master process has two high level commands which it can give to slaves or execute itself:

(1) Execute the local query Q on sites $S_1, S_2, \ldots, S_n$

(2) Move the fragment of relation R_i^j on site S_j to $S_k, \ldots, S_n$.

This is the process view that will be used for this thesis. There are other possible processing models. The choice of processing model is important because it precludes some algorithms. For example, one might consider an algorithm which gives a query to a slave for which the slave then becomes the master. This would evolve into a complicated tree control structure potentially with multiple processes on each site. Such a proposal was made in [STON77]. The motivation for this master-slave model is its simplicity for implementation and debugging. The flow of control is centralized and bookkeeping is simplified. Only the master needs to know the distribution of data and what each slave is doing.

2.5. Optimization Criterion

As stated in the previous section there are two commands that the master process can issue: execute a local query or move a fragment of a relation. We must choose a sequence of these commands which satisfy some optimization criterion. We will look at optimizing

(1) the total number of messages,

(2) the number of bytes transmitted across the network,

(3) the response time (wall clock time).

For each user, one QUEL statement is optimized at a time. We are not trying to optimize multiple statements from

multiple users because the usefulness of doing so is strongly application dependent. We shall assume that each query is unique and different from the previous queries; therefore, no attention will be paid to keeping partial results from previous queries in anticipation that they will be reused.

We shall frequently refer to the "cost" to perform some operation. Cost should always be equated with time. The cost to perform a query is the time elapsed when the query is received until the query is complete.

2.5.1. Minimizing the Number of Messages

The cost to send a message includes passing the message through the various software layers on the sending and receiving machines. In addition the transmission method may induce its own delay. For example, a satellite communication may have a several hundred millisecond delay until the message begins but it may have an extremely high bandwidth This implies that if the cost is a linear function

$$C_0 + C_1 \lceil X \rceil$$

and C_0 is large compared to $C_1 \lceil X \rceil$, it will be important to minimize the number of messages.

2.5.2. Minimizing the Number of Bytes Transmitted

Minimizing the amount of data transmitted makes sense in a majority of cases. If the transmission speed is very slow it will make sense to reduce the amount of data that must be transmitted. Transmitted data has some hidden costs; that is, the receiving sites must store the data, presumably on disk. Assuming an infinite speed network, the transfer of data looks very much like a disk to disk transfer. This is a non-negligible local cost. It can be minimized by reducing the amount of data moved. Networks such as the ARPANET transfer data at a rate of roughly 6K bytes/sec. This is about 25 times slower than a local disk to memory transfer which is roughly 30 milliseconds for a 4k byte transfer or 130K bytes/sec. Thus the time delay for network communication is significantly greater than for local processing.

2.5.3. Minimizing the Response Time

The response time is the sum of the processing delay and the communication delay. Minimizing just one does not necessarily yield an overall minimization. On high speed networks (such as the ETHERNET) transmission time is significantly shorter than the time required to transfer data from a local disk to memory (roughly three times faster if the ETHERNET operates at three megabits per second). Careful attention must be paid to local processing delay.

2.6. Conclusions

This chapter has presented a model of a distributed data base environment. Relations consist of fragments with are distributed across a network of computer sites. The sites are connected either by a broadcast network or a site-to-site network. The processing tactics must make efficient use of the communication network and the processing sites. Specifically they must consider the number of messages, the amount of data transmitted and the response time. The processing tactics will be presented in chapter three.

CHAPTER 3

PROCESSING STRATEGIES AND ESTIMATING RESULT SIZES

A variety of processing strategies are presented in this chapter. Each tactic is based on the models presented in chapter two. The tactics are presented here and then in chapter four, they are analyzed with respect to the optimization criteria presented in section 2.5.

3.1. One Variable Detachment and Distribution Criteria

On multi-variable queries, a decision must be made whether to detach and separately execute one variable subqueries. As an example, consider the query:

```
range of e is employee
range of d is dept
retrieve (e.name)
        where
                e.manager > 15
        and
                e.dept = d.dept
        and
                d.floor = 1
        and
                e.salary > 8000.
```

There are one variable clauses on both e and d. The clauses are:

```
e.manager > 15 and e.salary > 8000
d.floor = 1
```

These clauses can be compared with the distribution cri-

terion using a simple propositional calculus theorem prover. For each fragment, it can be determined that:

(1) no tuples in the fragment satisfy the user's query,

(2) all tuples in the fragment satisfy the user's query,

(3) Those tuples in the fragment that satisfy the user's query cannot be determined in advance.

For example, given the distribution criterion:

```
range of e is employee
distribute e at
        berkeley where e.manager <= 10,
        tahiti where e.manager > 10
                    and e.manager <= 20,
        paris where e.manager > 20
```

and the one variable subquery

```
range of e is employee
retrieve (e.name,e.dept)
        where
                e.manager > 15
```

it is possible to detect that no tuples from Berkeley will satisfy the query, some tuples from Tahiti will satisfy, and all tuples from Paris will satisfy. Thus the distribution criterion can immediately eliminate the fragment at Berkeley and can eliminate the need to check the tuples at Paris.

A separate decision must be made concerning whether to detach and execute the one variable subqueries:

```
(1)  retrieve into e'(e.name,e.dept)
            where
                    e.manager > 15
            and
                    e.salary > 8000

(2)  retrieve into d'(d.dept)
            where
                    d.floor = 1
```

Executing the subqueries separately from the main query will restrict the size of e and d. The remaining query would then be

```
retrieve (e'.name)
        where
                e'.dept = d'.dept
```

The cost to detach and execute one variable restrictions is the cost to transmit the command plus the local cost of executing the restriction.

3.2. Fragmented Processing Technique

The Fragmented Processing (FP) technique applies to all queries involving two or more relations. Suppose we have a join between relations R_1 and R_2. On a distributed data base this requires the join of every fragment of R_1 with every fragment of R_2 or

$$R_1^i(i=1,n) \text{ JOIN } R_2^k(k=1,m)$$

The FP technique is a method for performing n times m joins. Rather than viewing the problem as nm joins, it can be viewed as n or m joins. Make one of the two relations fully redundant at each site S_j holding a fragment

of the other relation. If R_1 is made fully redundant, each site S_j will compute

$$R_1 \text{ JOIN } R_2^j$$

If R_2 is made fully redundant, each site S_j will compute

$$R_1^j \text{ JOIN } R_2$$

To generalize this technique to N variables, the algorithm is to fully replicate N - 1 variables; however, one variable, call it R_p, is not moved. Then each site, S_j, will have

$$R_1, R_2, \ldots, R_p^j, \ldots, R_n$$

This tactic can be further extended by allowing complete freedom in selecting processing sites. Then two interrelated decisions must be made, choosing R_p, the relation to remain fragmented, and choosing the number of processing sites K, where K is less than or equal to the number of data base sites.

In summary, the FP technique chooses the set of sites, K, and a relation to be left fragmented, R_p. The remaining relations, $R_i, i \neq p$, are replicated at all K processing sites. In chapter four we will discuss choosing R_p and K.

3.3. Splitting a Query

Queries with three or more variables can be processed in "pieces", For example,

```
range of e is employee
range of d is dept
range of s is supply

retrieve (e.name, d.dname, s.item)
        where
                e.manager = d.manager
        and
                e.number = s.number
```

can be processed in two pieces, each involving two variables:

```
(1) retrieve into temp(e.name, e.number, d.dname)
        where
                e.manager = d.manager
```

followed by

```
(2) range of t is temp
    retrieve (t.name, t.dname, s.item)
        where
                t.number = s.number
```

Intuitively, such a split is advantageous because it delays the transmission of one or more relations. For example, to process query (1) above, either "e" or "d" has to be moved. To process query (2) either "t" or "s" has to be moved. Processing the three variables all at once would require moving either "e" and "d", or "e" and "s", or "d" and "s". Instead of moving "e" and "d", move "e" process the first piece, then move "t" and process the second piece. That split will be cost effective if "t" is

smaller than "d".

This approach can be extended to any query of three or more variables. Given a query involving variables $V_1, V_2, \ldots, V_n$

$$Q(V_1, V_2, \ldots, V_n)$$

the query can be split into two queries, the first involving x variables:

$$Q'(V_1, \ldots, V_x) \quad x \geq 2$$

This produces a new result variable which shall be called V_y. The remaining query involves V_y and the remaining n - x variables:

$$Q''(V_y, V_{x+1}, \ldots, V_n)$$

This technique can be applied recursively to Q' and Q'' if either has three or more variables. Query splitting is a generalization of a technique called reduction, proposed by Wong and Youssefi [WONG76], [YOUS78a]. In reduction, the variable V_y is composed of domains from at most one relation. Query splitting relaxes this requirement.

3.4. Transmission of Fragments

The FP technique described in section 3.2 requires fragments to be moved between sites. As described in section 1.4.1, a fragment commonly has some keyed structures. For example, a fragment may have a primary ISAM structure

on domain one and a hashed secondary structure on domain two. These structures exist for performance reasons. A mechanism exists [STON76, GRIF79] for examining a query and choosing whether to access a relation through a primary or secondary structure, or to sequentially read the relation. The optimal choice will depend on a variety of factors including the distribution of key values, the size of the relation (measured both in pages and in number of tuples), which keys are specified in the qualification of the query, etc. The access path selection mechanism for distributed databases must be expanded to include transmission cost and other factors which arise when several fragments of the same relation are brought together. The access path selection will determine what is transmitted when a fragment is moved. We will examine three choices for transmitting a fragment:

(1) projecting only the needed domains, sorting to remove duplicates, and then sending the fragment.

(2) sending the complete relation with its primary structure (ISAM, Btree or Hash).

(3) sending the relation and a useful secondary structure.

The choice depends on the optimization criterion. If one is optimizing for minimum transmission cost then clearly

choice (1) is always better than (2) or (3). If one is optimizing for response time, then transmitting the relation structure might make the local processing more efficient. The increased efficiency must be compared with the additional transmission cost of sending the structure. This is an example of trading increased communication cost for reduced local processing cost.

3.5. Join Encoding Techniques

In section 3.3, we explained that splitting a query into pieces can improve overall performance. The example in that section included creating a new relation from the equi-join of two existing relations.

```
retrieve into temp (e.name, e.number, d.dname)
        where
                e.manager = d.manager
```

The new relation "temp", might then be transmitted to other sites. Whenever the target list has more than one variable, the result can potentially be as large as the cross product of the relations in the target list. There are several join encoding techniques which can be used to reduce the cost of transmitting the new relation. The problem is basically a coding problem and we will examine three techniques for encoding a join. The three techniques are:

1) Physically form the join.

2) Encode the join as one tuple from relation R_i followed by one or more tuples from relation R_j which match the R_i tuple.

3) Evaluate the join and form two new relations R_1' and R_2'. R_1' will contain those tuples from R_1 which are part of the join, and similarly for R_2'. The two new relations are then transmitted and the join is recreated at the destination site.

In all three cases, only the domains which are needed in subsequent processing are kept. Cases 1 and 2 are relatively straight forward. Case 3 can be clarified with an example. If the example presented at the beginning of this section were processed using case 3, the following steps would occur:

```
step 1:
  retrieve into e' (e.name, e.number, e.manager)
        where e.manager = d.manager

  retrieve into d' (d.dname, d.manager)
        where e.manager = d.manager

step 2:
  move e' and d' to the required site(s)

step 3:
  retrieve into temp (e'.name, e'.number, d'.dname)
        where e'.manager = d'.manager
```

Relations e' and d' are guaranteed to have only tuples which are part of the join. Note that is not difficult to compute e' and d' concurrently in step 1. This technique

can reduce the transmission cost if there is a many-to-many relationship between the two relations, but it also involves non-trivial processing in step 3 to reconstruct the join at the destination site. (This idea is similar to what Bernstein refers to as a "semi-join" [BERN79]).

The three codings represent trade-offs between processing costs and transmission costs. The best choice depends on the optimization criterion. These issues will be examined in section 4.5.

3.6. Movement of Result Relation

Typically there is a required location for the result of a query. For a "retrieve" the results must ultimately end up at the user's site. For updates (append, delete, replace) the tuples must first be identified and then changed. The change must be reflected at the site where the tuples reside. Suppose there is a query:

```
replace e(salary = 1.1 * e.salary)
        where
                e.manager = d.manager
        and
                d.dname = "toy"
        and
                e.salary < 1000
```

Solving this query using the FP technique requires moving either e or d. Suppose e is moved. Each processing site S_j will have a complete copy of R_e (composed of all fragments R_e^j $j=1,2,..n$) and its own fragment R_d^j. Now each

processing site will find some tuples from R_e which must be updated. To update a tuple from R_e, the change must be performed at the site S_j where the tuple originated. Additional network communication will be required to direct the non-local updates to their correct sites.

Now suppose d is moved instead of e. Each processing site S_j will have its own fragment of R_e^j and a complete copy of R_d (composed of the fragments R_d^j $j=1,2,\ldots n$). Since only R_e is being updated and each processing site has only its own local R_e^j fragment, no additional communications are required.

The FP technique will favor moving the smaller relation in order to reduce communication costs. If the smaller relation is the relation being updated, the additional cost of the non-local updates must be considered.

3.7. Shuffling Strategies for Updates

There are several cases when all the processing sites will have to distribute data to other processing sites. This can happen

(1) on updates (append, delete, replace) when the result relation is moved by the FP technique,

(2) on a "replace" when a domain is changed which occurs in the distribution criterion.

(3) when the distribution criterion itself is changed,

(4) on an "append" when appending to a relation from one or more other relations.

An example of (1) was given in section 3.6 when "e" was moved. For examples of the other cases, suppose the distribution criterion is

```
distribute e at
        berkeley where e.salary < 10000,
        paris where e.salary >= 10000
                and e.salary < 20000,
        tahiti where e.salary >= 20000
```

and there is a relation

```
salchange(number, newsalary)
```

Case (2) would occur with the query

```
range of s is salchange
range of e is employee
replace e (salary = s.newsalary)
        where
                e.number = s.number
```

Each employee identified in "s" would get a new salary and might have to be moved to another site in order to satisfy the distribution criterion. Case (3) would happen if the distribution criterion on the employee relation were changed, for example,

```
range of e is employee
distribute e at
        berkeley where e.manager = 13,
        paris where e.manager = 27,
        tahiti where e.manager != 27
                and e.manager != 13
```

As an example of case (4), suppose we want to create a

relation with a distribution based on age.

```
create oldemp(name=c20, salary=i4, age=i2)
range of o is oldemp
distribute o at
        berkeley where o.age < 40
        paris where o.age >= 40 and o.age < 60
        tahiti where o.age >= 60
```

If we appended to the "oldemp" relation from the "employee" relation, the qualifying tuples from "employee" would have to be distributed based on the "age" domain.

```
range of e is employee
append oldemp (e.name, e.salary, e.age)
        where e.age > 25
```

In each case an efficient way must be determined to redistribute or "shuffle" the data. We will consider two methods for shuffling: centralized control and decentralized control. In centralized control all sites identify and process tuples which belong to their own site. Next they transmit their remaining tuples to one centralized site. The centralized site partitions and transmits the remaining tuples to the sites where they belong. Using decentralized control each site processes its own data, partitions the remaining data, and then transmits it directly to the correct site.

Decentralized control requires less data movement and achieves greater parallelism; however, it generates substantially more message traffic on the network. These issues are examined in section 4.6.

3.8. Static Versus Dynamic Decision Making

In static decision making the processing strategy, i.e., the tactics and their order of execution, is decided in advance of any actual query processing. Dynamic decision making, on the other hand, makes only one decision at a time. It decides what the tactic should be, performs it and only then chooses the next tactic. Both methods base decisions on estimates of the result sizes. Static decision making has the undesirable property that errors from bad estimates accumulate. For example, consider the query

```
retrieve (p.pname)
        where
                p.pnum = s.pnum
        and
                s.snum = 475
        and
                s.shipdate = "79-10-21"
```

Two decisions must be made:

(1) whether to detach and execute the one variable subquery on s

(2) whether to move s or p.

Suppose detaching and executing the one variable subquery is a good tactic. In static decision making, the choice between moving p and s must be decided based on the size of p and the estimated size of the restricted s. In dynamic decision making, the decision of whether to move s or p is deferred until after the subquery on s has been run. Thus dynamic decision making will determine what to

do based on two known values while static decision making must decide based on one known value and one estimate.

The advantage of dynamic decision making is that it has more information and therefore, can potentially make a better sequence of decisions. The benefits of dynamic decision making are analyzed in section 4.3.

In distributed systems, dynamic decision making can require increased communication cost since the status of each tactic must be returned to the master site before the next tactic can be determined. This information might otherwise be unnecessary. Static decision making generally requires less communication since all sites can know all processing steps in advance. Static decision making has the property that decisions are made at "compile time" and thus there is no run time overhead. The extra overhead associated with dynamic decision making is examined in section 4.7.

3.9. Estimating Result Sizes

The ability to estimate the number of tuples which will satisfy a query is crucial in deciding whether to split a query (section 3.3), in deciding whether to move a result relation (section 3.6), and in choosing between static and dynamic decision making (section 3.8). Many researchers [YAO77, HAMM76, DUHN78] have studied the prob-

lem of estimating the size of a result from a given query based on statistical information. It is a very difficult problem which is typically solved by using a simplified model of the problem. Such models are inevitably based on assumptions about the distribution of data values within a domain, and assumptions about the independence between domains. Reasonable results can be obtained if the model is an accurate representation of the data.

The accuracy of an estimate depends on the amount of statistics kept about the relations and their attributes. There can be a significant overhead in maintaining accurate statistics. Thus one is motivated to keep as little information as possible yet still allow reasonably accurate estimates. We will consider maintaining three levels of statistics

case 1: relation cardinality (number of tuples in the relation)

case 2: relation cardinality plus 1 bit of information per domain

case 3: relation cardinality plus more than 1 bit per domain

We will show how estimates can be computed for each of the three cases. For example, assume we maintain case one, that is, the cardinality of relations e and d. What is the estimated number of tuples which satisfy the query:

```
retrieve (e.name)
where
        e.dept = d.dept
and
        e.salary > 10000
```

Although this query might examine the cross product of e and d, we know the cardinality of the result can not exceed the cardinality of e. Now suppose the query were

```
retrieve (e.name,d.dname)
where
        e.dept = d.dept
and
        e.salary > 10000
```

Knowing only the cardinalities without knowing any semantic or statistical information about the domains being joined, it is impossible to estimate the size of the result. The minimum result size is zero and the maximum is the product of the two cardinalities. Consider case 2. We have 1 bit of information per domain and that bit, if set, tells us that every value in the domain is unique. If the bit is clear it means that two or more values in the domain are the same. For example, a domain holding unique numbers such as employee numbers or social security numbers would have its bit set. If both e.dept and d.dept had their "bits set" then we know the result from either query can not exceed the cardinality of the smaller relation. If only one domain had its bit set then the result size could not exceed the cardinality of the larger relation. In addition to one bit of information, the target

list plays an important role. If only one variable is in the target list, then the result cannot exceed the cardinality of the relation in the target list.

If more than one bit of information is available about each domain (case 3) the estimates could be computed with greater accuracy. The role of the estimation procedure is examined in section 4.3. Also included in that section is an examination of the sensitivity of the splitting algorithm to the accuracy of the estimation procedure. Note that the techniques presented in this thesis are independent of the actual method of estimating result sizes.

3.10. Summary

In this chapter we have presented a variety of techniques which can be applied to queries on distributed data bases. The effectiveness of each tactic depends on trade-offs which can be very difficult to analyze. The goal of this thesis is to develop algorithms which perform well for a large class of applications. Chapter 4 contains an analysis of each technique.

CHAPTER 4

ANALYSIS OF PROCESSING STRATEGIES

In this chapter we will analyze how each of the tactics presented in chapter 3 satisfy the optimization criteria identified in chapter 2. One goal of this chapter is to prove, whenever possible, how a tactic will perform. A second goal is to provide intuitive insight into the properties of each tactic. Most of the tactics have been implemented, or are being implemented as part of the distributed data base version of INGRES.

The chapter begins with the analysis of one variable restrictions, followed by an analysis of the FP technique. Next the query splitting technique is examined. In particular, we examine the relationships between the search strategy for finding the optimal split, the estimation procedure, and static and dynamic decision making. The remaining sections analyze moving a relation's structure, encodings of joins, shuffling strategies, estimating result sizes and decision making. Lastly we compare the results of this thesis and other published works.

4.1. Analysis of One Variable Detachment

The examination of one variable restrictions is logically broken into the use of distribution criteria and the

detachment of one variable restrictions.

A distribution criterion functions exactly like a coarse, top level index; breaking a relation into disjoint fragments by specifying physical locations for tuples. As such, it improves performance in two ways. First, it can limit the amount of data which must be examined and second it can reduce the number of sites required for processing the query. If the query can be done locally, it is reasonable to assume that it will be faster than if network communications are required. This strongly suggests that the distribution criteria should be quickly accessible at each site.

The distribution criteria should be used as follows: When a query is first received, each of the variables should be examined for restrictive clauses involving only that variable. For those variables with one variable clauses, get their distribution criteria (if any). Use the distribution criteria to eliminate those fragments for which no tuples satisfy the restriction. Identify those fragments for which all tuples satisfy.

The next step is to consider detaching and performing the one variable restrictions. In nearly all cases, doing so will reduce communication costs. Each variable will be restricted in size or at worst, remain the same size. The overall communication cost will be reduced if the cost to

transmit the restriction command is less than the cost to transmit the data that would be eliminated by the restriction. Whether this is true in practice is application dependent. It is quite likely that the command can be "piggy backed" to another command (for example it can be sent with the command to move the fragment). In that case, detaching one variable restrictions never increases communication costs. If the optimization criterion is to minimize communication costs then the restrictions should be detached and performed.

If one considers response time then there are cases when performing the restriction will be detrimental. Such cases are identical to those in centralized data bases. An analysis of one variable detachment on a centralized data base can be found in [YOUS78b]. The major argument against detachment is as follows. If the restriction is performed as described in [STON76], then the result of the restriction is saved in a new, temporary relation. There are cases when the original physical structure of a relation would be valuable for subsequent processing and the loss of the structure (caused by the restriction to a temporary) is not offset by the reduced size of the restricted relation. In such a case the local processing time can be adversely affected by considering only communication costs. Further examples will be identified in

section 4.4.

In summary, the use of distribution criteria to restrict a relation is an effective technique provided the cost to access the distribution criteria is low. This can be achieved by replicating the distribution criteria at all sites or caching the distribution criteria whenever it is accessed. Detaching and executing one variable restrictions is frequently a good idea. It is not a good idea when the loss of the physical structure degrades subsequent processing which could have used the structure.

4.2. Analysis of the Fragmented Processing Technique

The Fragmented Processing (FP) Technique can process any query with two or more variables. As mentioned in chapter three, the technique consists of choosing one relation which is not moved (R_p) and choosing K processing sites. The remaining relations, $R_{i,i\neq p}$, are moved to the K processing sites. Processing then begins on all K sites and the result is the union of results on the K sites. The analysis will show how to choose R_p and K with regard to the optimization criteria.

The analysis will proceed as follows. First a general formula is developed to measure communication cost. Next we present the solution for choosing R_p and K to minimize communications costs on broadcast and site-to-

site networks. Next we consider choosing R_p and K to minimize response time. Finally a heuristic is presented which balances communication cost with response time.

We begin by reviewing the notation that will be used throughout this chapter:

N = number of sites in the data base
n = number of relations, from 1 to n.
i = always used to index a relation (e.g. R_i)
j = always used to index a site (e.g. S_j)

R_i^j = fragment of R_i at S_j

M_i = number of sites holding a fragment of R_i

$|R_i|$ = the sum total in bytes of all fragments of R_i

$|R_i^j|$ = the size in bytes of R_i^j

The general formula for the number of bytes to be transmitted before processing can begin is derived from the following two facts:

(1) For each processing site, S_j, $R_{i,i\neq p}^j$ must be moved to all other K - 1 processing sites.

(2) For each non-processing site, S_j, $R_{i,i\neq p}^j$ must be moved to the K processing sites and R_p^j must be moved to one processing site.

To simplify the discussion, the processing sites will always be renumbered to be sites $S_1, S_2, \ldots, S_k$. K is an integer which represents the number of sites chosen. The formula for the number of bytes which must be moved is then:

$$\text{comm} = \sum_{j=1}^{K} C_{K-1} \left\lceil \sum_{i \neq p} \left| R_i^j \right| \right\rceil + \sum_{j=K+1}^{N} C_K \left\lceil \sum_{i \neq p} \left| R_i^j \right| \right\rceil + \sum_{j=K+1}^{N} C_1 \left\lceil \left| R_p^j \right| \right\rceil \qquad (4.1)$$

The first term comes from (1) above. It is the cost to transmit the relations from a processing site to the other processing sites. The second and third terms come from (2) above. It is the cost to transmit relations from the N - K non-processing sites.

We will first examine minimizing equation (4.1). This will be done for both the broadcast model and the site-to-site model.

4.2.1. Minimizing Communication Cost for the Broadcast Model

For the broadcast model, communication cost will be minimized by either doing all the processing at one site, or processing at all sites which have a fragment of the largest relation. Theorem 1 will prove this.

Intuitively, the situation is as follows. If one site has more data than the largest relation, then the data distribution is skewed heavily towards that site. In such a case, communication costs are minimized by choosing K = 1 and moving all data to the site with the most data.

Note that when K = 1, there is no R_p since no relation is left fragmented. If no site has more data than the largest relation, then R_p should be chosen to be the largest relation. This is because all relations except R_p will have to be moved once. Thus one wants to avoid moving the largest relation. Furthermore, all sites which have a fragment of R_p, no matter what size, should be chosen as processing sites. If a site, S_j, which had a fragment of R_p were not a processing site, the fragment R_p^j would have to be moved. This cost is avoided if S_j is allowed to be a processing site. As long as there is more than one processing site, all $R_{i,i\neq p}$ have to be moved, they can be moved to all sites for the same network cost as one site. This is true on a broadcast network because $C_k(x) = C_1(x)$. Thus there is no incentive to exclude a site.

THEOREM 1:

For a broadcast model, communication cost will be minimized by either choosing:

(1) K = 1 and choose as a processing site, the site with

$$\max_j \left\lceil \sum_i \left| R_i^j \right| \right\rceil$$

(2) or else choose

$$R_p = \max_i \left\lceil \left| R_i \right| \right\rceil$$

and choose $K = M_p$.

PROOF:

For the broadcast model $C_k[x] = C_1[x]$, therefore formula 4.1 can be written as:

$$\sum_{j=1}^{K} C_1\left[\sum_{i\neq p} R_i^j\right] + \sum_{j=K+1}^{N} C_1\left[\sum_{i\neq p} R_i^j\right] + \sum_{j=K+1}^{N} C_1[R_p^j] \tag{4.2}$$

The first and second terms can now be combined giving:

$$\text{comm} = \sum_{j=1}^{N} C_1\left[\sum_{i\neq p} R_i^j\right] + \sum_{j=K+1}^{N} C_1[R_p^j] \tag{4.3}$$

Now assume that $K > 1$. The first term is independent of the value of K. The second term is zero when $K = M_p$. Hence if $K > 1$, equation 4.3 is minimized by $K = M_p$. Any other solution would make the second term non-zero. With the second term zero, the communication cost from 4.3 becomes:

$$\sum_{j=1}^{N} C_1\left[\sum_{i\neq p} R_i^j\right] \tag{4.4}$$

Since equation 4.4 iterates over all possible sites, S_j, it can be simplified to

$$C_1\left[\sum_{i\neq p} R_i\right] \tag{4.5}$$

This can be rewritten as

$$C_1 \left\lceil \sum_i \lceil R_i \rceil - \lceil R_p \rceil \right\rceil \tag{4.6}$$

Since all relations must be moved except R_p, equation 4.6 is minimized by choosing R_p to be:

$$\max_i \left\lceil \lceil R_i \rceil \right\rceil \tag{4.7}$$

At this point it has been proven that if K > 1, K must be M_p and R_p should be $\max_i \left\lceil \lceil R_i \rceil \right\rceil$. If K = 1, only one site will be a processing site. Thus the first term of equation 4.1 is zero, since the cost to send X bytes to K - 1, or zero sites, is obviously zero. The communication cost is then

$$\sum_{j=K+1}^{N} C_1 \left\lceil \sum_{i \neq p} \lceil R_i^j \rceil \right\rceil + \sum_{j=K+1}^{N} C_1 \left\lceil \lceil R_p^j \rceil \right\rceil \tag{4.8}$$

Since no relations have to be moved <u>from</u> the processing site, only <u>to</u> it, there is no R_p. This simplifies equation 4.8 to

$$\sum_{j=2}^{N} C_1 \left\lceil \sum_i \lceil R_i^j \rceil \right\rceil$$

which can be rewritten as

$$C_1 \left\lceil \sum_i \lceil R_i \rceil - \sum_i \lceil R_i^1 \rceil \right\rceil \tag{4.9}$$

Equation 4.9 is minimized if the second term as large as possible, that is, by choosing S_j to be

$$\max_j \left\lceil \sum_i \lceil R_i^j \rceil \right\rceil$$

QED

Theorem 1 proves that there must either be one processing site or all sites which have a fragment of R_p must be processing sites. We will now show how to decide between the two cases. If one site holds more data than the largest relation, then K = 1, otherwise $K = M_p$.

THEOREM 2:

To minimize communication costs on a broadcast network, K should be 1 if

$$C_1\left[\max_j\left[\sum_i R_i^j\right]\right] > C_1\left[\max_j\left[R_j\right]\right]$$

and otherwise,

$$K = M_p$$

PROOF:

If K = 1, then by theorem 1 the communication cost is expressed by equation 4.9. If K > 1, then by theorem 1, the communication cost is expressed by equation 4.6. Compare equations 4.9 and 4.6

$$C_1\left[\sum_i\left[R_i\right] - \sum_i\left[R_i^1\right]\right] < C_1\left[\sum_i\left[R_i\right] - \left[R_p\right]\right] \qquad (4.11)$$

Subtracting common terms and multiplying by -1 yields:

$$C_1\left[\sum_i\left[R_i^1\right]\right] > C_1\left[\left[R_p\right]\right]$$

Replacing R_p and R^1 by their values gives:

$$C_1\left[\max_j\left[\sum_i R_i^j\right]\right] > C_1\left[\max_j\left[R_j\right]\right] \qquad (4.12)$$

If the inequality holds, then K = 1, otherwise K = M_p by theorem 1.

QED

In summary, to minimize communication cost for the broadcast model, choose K = 1 if one site has more data than the largest relation, otherwise, choose R_p to be the relation containing the most data and choose K to be all sites holding a fragment of R_p.

4.2.2. Minimum Communication Cost for Site-to-Site Model

The minimization of network traffic on a site-to-site model is very sensitive to the distribution of data among the sites. The general procedure for a site-to-site network is to order the sites according to the amount of data they have. The largest relation is chosen as R_p. Since there must be at least one processing site, the site with the most data is always chosen as a processing site. Each additional site is examined to see whether it would receive less data as a processing site than it would have to transmit as a non-processing site.

The proof is again divided into two steps. Theorem three proves that communication cost is minimized by choosing K = 1, or else choosing R_p to be the largest relation and making a site a processing site only if the amount of data it would have to receive as a processing site is less than the amount of data it would have to transmit as a non processing site.

THEOREM 3:

For a site-to-site network model, communication cost will be minimized by either

(1) choosing R_p to be $\max_i |R_i|$ and choosing K to include every site, S_j, for which

$$\sum_{i \neq p} |R_i| < \sum_i |R_i^j|$$

(2) choosing one site, S_j where j is

$$\max_j \left[\sum_i |R_i^j| \right]$$

PROOF:

The definition of the site-to-site model, states that $C_K(x) = KC_1(x)$. Since we assume that the cost to transmit x bytes is linear in the number of bytes, we need only minimize the number of bytes transmitted. This allows us to drop the cost function C(x), since minimizing x will also minimize C(x). Equation 4.1 can then be rewritten as

$$(K-1)\sum_{j=1}^{K}\sum_{i\neq p}|R_i^j| + (K)\sum_{j=K+1}^{N}\sum_{i\neq p}|R_i^j| + \sum_{j=K+1}^{N}|R_p^j| \quad (4.13)$$

Multiplying out terms and observing that:

$$\sum_{j=K+1}^{N}|R_p^j| = |R_p| - \sum_{j=1}^{K}|R_p^j|$$

yields:

$$(K)\sum_{j=1}^{K}\sum_{i\neq p}|R_i^j| - \sum_{j=1}^{K}\sum_{i\neq p}|R_i^j| + (K)\sum_{j=K+1}^{N}\sum_{i\neq p}|R_i^j| + |R_p| - \sum_{j=1}^{K}|R_p^j|$$

Combining the first and third terms yields:

$$(K)\sum_{j=1}^{N}\sum_{i\neq p}|R_i^j| - \sum_{j=1}^{K}\sum_{i\neq p}|R_i^j| + |R_p| - \sum_{j=1}^{K}|R_p^j|$$

Observe that

$$(K)\sum_{j=1}^{N}\sum_{i\neq p}|R_i^j| = \sum_{j=1}^{K}\sum_{i\neq p}|R_i|$$

Substituting the above and factoring out the $\sum_{j=1}^{K}$ gives:

$$\sum_{j=1}^{K}\left[\sum_{i\neq p}|R_i| - \sum_{i\neq p}|R_i^j| - |R_p^j|\right] + |R_p|$$

Combining terms gives:

$$\sum_{j=1}^{K}\left[\sum_{i\neq p}|R_i| - \sum_{i}|R_i^j|\right] + |R_p| \quad (4.14)$$

If K=1, then only one site, S_j, is a processing site and there is no R_p. Equation 4.14 then becomes

$$\sum_i |R_i| - \sum_i |R_i^1|$$

This is minimized by choosing j to be the site S_j with

$$\max_j \left[\sum_i |R_i^j| \right]$$

We must now evaluate the case when K > 1. Equation 4.14 can be rewritten as

$$(K) \sum_{i \neq p} |R_i| + |R_p| - \sum_{j=1}^{K} \sum_i |R_i^j|$$

Any $|R_i|$ $i \neq p$ will be multiplied by K (K > 1) but $|R_p|$ is a singular term and is not multiplied by any constant. Thus the equation is minimized by chosing R_p to be the $\max_i |R_i|$. K is then chosen to minimize

$$\sum_{i \neq p} |R_i| - \sum_i |R_i^j|$$

by choosing only those values of j for which the term is zero or negative. Thus we want only those S_j for which

$$\sum_{i \neq p} |R_i| < \sum_i |R_i^j| \qquad (4.15)$$

K is then the count of those sites S_j.

QED

Theorem 3 proves that communication on a site-to-site network is minimized either by having one processing site which has the most data, or by choosing R_p to be the largest relation and choosing a site as a processing site only

if the amount of data the site would have to receive as a processing site is less than the <u>additional</u> data that it would have to send as a nonprocessing site (that is, the amount it receives is less than R_p^j). Theorem 4 will differentiate between the two cases identified by theorem 3.

THEOREM 4:

If for every site S_j, j = 1 to N

$$\sum_{i \neq p} |R_i| > \sum_i |R_i^j|$$

then choose K=1.

PROOF

IF K = 1 then all sites except one, S_1, must transmit their fragments to S_1. The number of bytes transmitted can be written as:

$$\sum_i \sum_{j=1}^{N} |R_i^j| - \sum_i |R_i^1|$$

or simply

$$\sum_i |R_i| - \sum_i |R_i^1|$$

It is sufficient to show that if we include any other site as a processing site, the number of bytes transmitted will increase. If another processing site, S_2 is added, the formula 4.14 becomes

$$\sum_i |R_i| - \sum_i |R_i^1| - |R_p^2| + \sum_{i \neq p} |R_i| - \sum_{i \neq p} |R_i^2|$$

This was derived as follows. The original cost includes moving all fragments from sites $S_j, j \neq 1$ to site S_1. R_p^2 is subtracted since it does not have to be moved if S_2 is a processing site. Additionally, all sites $S_j, j \neq 2$ would have to send their relation $R_i, i \neq p$ fragments to site S_2. Now combining terms 3 and 5 gives:

$$\sum_i |R_i| - \sum_i |R_i^1| + \sum_{i \neq p} |R_i| - \sum_i |R_i^2|$$

We were given that

$$\sum_{i \neq p} |R_i| - \sum_i |R_i^2| > 0$$

thus including another site increases the number of bytes transmitted.

QED

In summary, theorem 4 determines if K = 1. If K = 1, then theorem 3 states that the processing site should be the site with the most data. If K > 1, then by theorem 3, R_p should be chosen to be the largest relation and the processing sites should be those sites for which

$$\sum_{i \neq p} |R_i| - \sum_i |R_i^j| \leq 0$$

The above equation compares the amount of data the site would have to receive as a processing site with the amount of data it would have to send as a non-processing site.

4.2.3. Minimizing Response Time

If the optimization criterion is to minimize response time, then the choices for R_p and K must take local processing time into consideration. Response time depends on the time required to process the query at the local sites, as well as the time required to transmit relation fragments between sites. As K increases, more sites become processing sites and there is an increase in parallelism, potentially decreasing processing time; however, as theorems 1-4 have shown, increasing K can increase the communication traffic on either network model. Processing time can also be reduced if R_p has a useful physical structure. When a relation fragment is moved, it loses its physical structure. Only R_p will retain its original structure since it is the only relation that isn't moved. This issue is examined further in section 4.4

Before examining the trade-offs involved between local processing and communication traffic, we will describe the problems involved in estimating the time required to process a local query. The processing time depends on the access paths used (accessing through a clustered or non-clustered index, using a hash key, sequentially reading the relation, etc.), the processing strategies used (tuple substitution, sort-merge join, reformatting, etc.), and the physical location of the

relation on disk (whether relation pages are clustered together, etc.). Moreover, the situation will be different on every processing site. To more fully appreciate what is involved in predicting the processing time, the interested reader is referred to [GRIF79, HAWT79]. The paper by [GRIF79] describes the complexity of estimating access path and join techniques for System R on a single site data base. The problem is compounded by the fact that there are multiple sites and a strategy must be found that is optimal for all sites. Additionally, the necessary statistical information may or may not be present and up to date at the master site.

At least one case exists for which the best choices for R_p and K can be determined for optimizing response time. If no relation is structured usefully, then R_p should be chosen to minimize the amount of data at each processing site. Since each processing site must have a copy of all fragments of each $R_i, i \neq p$, the choice for R_p should be the largest relation. This will process the minimum amount of data at each site. As was demonstrated in section 4.2.2, this will also minimize communication traffic.

There is a heuristic which can be employed to improve response time by increasing parallelism. On a broadcast network, if there are several candidates for R_p of a com-

parable size (we will say within "T" percent), then R_p should be chosen to be the relation with the most fragments, eg. $\max_i(M_i)$. This will maximize the value of K, improving local processing time.

For a site-to-site network, equation 4.15 can be changed to be:

$$T \sum_{i \neq p} |R_i| < \sum_i |R_i^j|$$

where T is a heuristic value between 0 and 1. When T = 1, communication costs are minimized as was proven in section 4.2.2. When T = 0, all sites will be chosen as processing sites. When T is slightly less than 1 then communication costs may not be minimized but more sites may become processing sites, increasing parallelism. We are relaxing the requirements for being a processing site in the hopes that the increased parallelism will improve response time. This will allow more sites to qualify as processing sites, increasing parallelism and reducing local processing time.

This approach represents a compromise based on reasonable assumptions. Experimentation should determine good values of T for a particular application environment.

In summary, exact formulas developed in theorems 1-4 can be used to choose R_p and K based on minimizing the cost formula for communications. It may be desirable to examine alternate choices of R_p and K which, while not

optimal for communication costs, will increase parallelism and improve overall response time.

4.3. Analysis of Split Tactic

As was explained in section 3.3, a query involving three or more relations can always be split into at least two pieces. Splitting the query may be more cost effective than processing the query all at once. Splitting a query delays the transmission of one or more relations in the hopes that the size of the relation will be reduced by some intermediate processing. We will examine the effectiveness of the query splitting technique only in terms of minimizing the number of bytes transmitted since that is its primary usefulness.

4.3.1. Overview of Query Splitting Analysis

When splitting a query it is always possible to exhaustively examine every combination of variables, looking for the best strategy. The computational complexity is reduced and the implementation simplified if a limited search can be used to find the best way to split a query. It is of primary concern in this section to compare exhaustive and limited search techniques.

Estimating the number of tuples which will satisfy a query is another important issue. The ideal is "perfect information"; that is, the size of the result of any query

is accurately known in advance. On the other hand, knowing the cardinality of each relation and one bit of information about each domain in each relation is a reasonable minimum requirement. The performance of the query splitting algorithms will be analyzed based on these two extremes of information.

In this section we will also examine dynamic decision making. Since estimates are never perfect, the size of the result from a processing step may be different than its estimated size. If so, the remaining processing strategy might need to be reevaluated. Dynamic decision making evaluates the processing strategy after each processing step.

There is a fixed overhead associated with running a remote query. Splitting one query into two doubles the fixed overhead. Thus the amount of data movement saved must be sufficient to offset the additional message traffic. As will be discussed in section 4.7, the overhead is strongly dependent on whether dynamic or static decision making is being used. If synchronization is required after each processing step, the overhead is the transmission of the query to the K processing sites, and the subsequent completion response from each processing site. For a broadcast model the overhead is 1 + K messages and for a site-to-site model the overhead is 2K mes-

sages.

The analysis will proceed as follows. A graph structure [WONG77] is introduced for expressing the interrelationships between the variables in the query in a concise manner. Two search strategies are presented - limited and exhaustive. They differ in the number of cases which they consider. The algorithms were coded and the performance of both algorithms were measured on a variety of test cases using an analysis program. This program tested:

(1) limited versus exhaustive search given perfect information.

(2) limited versus exhaustive search given 1 bit of information per domain and assuming a worst case estimate. This will be explained in detail in section 4.3.9.

(3) limited versus exhaustive search using either static or dynamic decision making.

(4) limited versus exhaustive search given 1 bit of information per domain and assuming less than a worst case estimate.

Finally, general figures of merit are given for each case and some general conclusions are drawn.

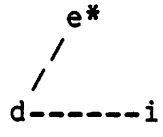
4.3.2. Graphic Query Representation

A graphic technique is useful for expressing the interrelationships of the variables in a query. Once all

single variable restrictions have been performed, the important structural information about a query is expressed by identifying the variables in the target list, and the variables in each conjunctive clause in the qualification. Let each variable in a query be represented by a single node in a graph. If the variable occurs in the target list then place an asterisk ("*") next to it. An edge connecting two nodes represents the fact that one or more conjunctive terms connect the two variables. For example, the query

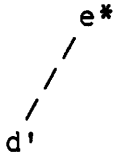
```
retrieve (e.name)
        where
                e.dept = d.dept
        and
                d.location = i.location
```

is represented as



For the sake of this discussion, any greater detail (such as the names of the joining domains, etc.) will not be necessary. The above query could be split into two pieces:

followed by



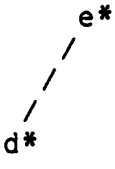
where d' is the result from the first query. In QUEL this split is expressed by the query:

```
retrieve into d'(d.dept)
        where
                d.location = i.location
```

followed by the query

```
retrieve (e.name)
        where
                e.dept = d'.dept
```

Another possible candidate for splitting the query is



followed by

In QUEL, this split is expressed as:

```
retrieve into d'(e.name,d.location)
        where
                e.dept = d.dept

retrieve (d'.name)
        where
                d'.location = i.location
```

A query can be split on any edge provided that the two subgraphs created both contain at least two variables. For example, since there is no edge between "e" and "i", there are no other choices for splitting the original query.

4.3.3. Split Algorithm Using Limited Search

Given an n variable query ($n \geq 1$), section 4.2 showed how to compute the communication cost required to process the query. Call this cost MAXP.

$$MAXP = PROC\lceil R_1, R_2, \ldots, R_n \rceil$$

Consider processing the query by doing a two variable query followed by the remaining n - 1 variables. This is always possible although not always cost effective. Splitting the query reduces the problem to two queries and the cost becomes

$$PROC\lceil R_1, R_2 \rceil + PROC\lceil R_2', R_3, \ldots, R_n \rceil$$

Examine all meaningful combinations of two variables followed by the remaining n - 1 variables. Meaningful combinations means those variables which are immediately connected in the query graph. Choose the minimum cost. If the minimum cost is less than MAXP, then assign MAXP to the new value and save the names of the two variables which should be processed first. Next consider processing the query by doing a three variable query followed by the

remaining n - 2 variables. The cost would become:

$$PROC\lceil R_1, R_2, R_3 \rceil + PROC\lceil R_3', R_4, \ldots, R_n \rceil$$

Once again, look at all meaningful combinations of 3 variables followed by n - 2. If the minimum cost is less than MAXP, then assign MAXP to it and remember the names of the variables involved. This procedure continues until the case of doing n - 1 variables followed by 2 variables is considered. This limited search requires considering at most

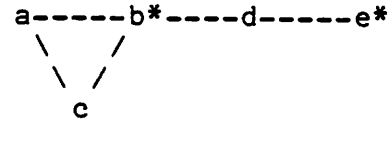
$$\sum_{i=2}^{n} \left\lceil \begin{matrix} n \\ i \end{matrix} \right\rceil$$

choices. The maximum number of choices occurs only if the user's query contained a join between every variable and every other variable.

Suppose we had a five variable query. Limited search would consider four cases:

Case	Notation
Perform all 5 variables at once	5
2 variable followed by 4 variable	2-4
3 variable followed by a 3 variable	3-3
4 variable followed by a 2 variable	4-2

Now suppose the query were



Theoretically, the maximum possible number of choices is

$$\sum_{i=2}^{n} \left\lceil \begin{matrix} n \\ i \end{matrix} \right\rceil = \sum_{i=2}^{5} \left\lceil \begin{matrix} 5 \\ i \end{matrix} \right\rceil = 26$$

but in fact the number of choices is limited by the interconnections of the variables in the graph. The two variable choices would be:

```
PROC(a, b) + PROC(b', c, d, e)
PROC(a, c) + PROC(c', b, d, e)
PROC(b, c) + PROC(c', a, d, e)
PROC(b, d) + PROC(d', a, c, e)
PROC(d, e) + PROC(e', a, b, c)
```

The three variable choices would be:

```
PROC(a, b, c) + PROC(c', d, e)
PROC(b, c, d) + PROC(d', a, e)
PROC(a, b, d) + PROC(d', c, e)
PROC(b, d, e) + PROC(e', a, b)
```

The four variable choices would be

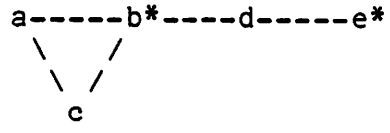
```
PROC(a, b, c, d) + PROC(d', e)
PROC(b, c, d, e) + PROC(e', a)
```

In this example, the number of possibilities was 11. Once the minimum cost choice has been selected, the first piece of the two can be immediately executed using the FP technique. The result relation from the first piece will be a single relation (although it may be encoded in any of ways discussed in section 3.5). Next the second piece of the query is adjusted to make any domain references to variables in the first piece refer to the result relation of the first piece. The bookkeeping process of maintaining domain references is straightforward and is similar to the one variable restriction algorithm found in [STON76].

Once the first piece of the query is complete, the second piece can be processed. This is done by applying the query splitting algorithm to it recursively. In this manner all possible combinations of query splitting can be generated. The search strategy is "limited" in how it makes its decision - not in what decisions are possible.

The limited search algorithm is shown in figure 1. The algorithm is written in a stylized version of the "C" language [KERN78]. Figure 1 shows the flow of control for the algorithm but little of the bookkeeping required. SPLIT(q) is the name of the query splitting routine. Trans_cost(q) is the cost to transmit the relations in order to process the query q. Variable_count(q) returns the number of variables in the query q. Estimate_result_size(q) computes an estimate of the result size of q using any desired estimation procedure. Process(q) actually performs the FP technique on the query q and updates all necessary information.

An example of the algorithm is now presented . Given the original query:

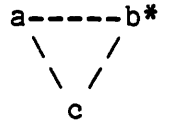


the split algorithm might decide to split the query into two, three variable pieces:

Algorithm for limited search

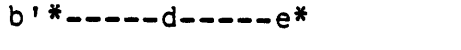
```
SPLIT(q)
{
        maxp = trans_cost(q)
        split = FALSE;
        for (i = 2; i < variable_count(q); i = i + 1)
        {
                for (each combination of i variables in q)
                {
                        form q' and q''
                        cost = trans_cost(q')
                        estimate_result_size(q')
                        cost = cost + trans_cost(q'')
                        if (cost < maxp)
                        {
                                maxp = cost
                                save q' and q''
                                split = TRUE
                        }
                }
        }
        if (split is TRUE)
        {
                process(q')
                adjust(q'')
                SPLIT(q'')
        }
        else
        {
                process(q)
        }
}
```

Figure 1.



piece 1

followed by the remaining piece



piece 2

Piece 1 will be processed using the FP technique. The splitting algorithm is then applied to piece 2. To continue the example, suppose the splitting technique decided to split piece 2 into:

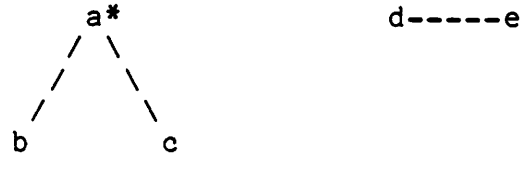
```
d*-----e*        piece 2
```

followed by

```
b'*-----d'       piece 3
```

At this point the query splitting tactic is complete. Note that two decisions were made. The first was to split the five variable query into a three variable query followed by another three variable query. The second decision was to split the three variable query into a two variable query followed by another two variable query.

In certain rarely expressed queries the first query may only return true or false. This happens only if the original user's query was disjoint. For example the query:



is in two disjoint pieces. The semantics of the query are if any "d" matches any "e" then process the remaining query on "a", "b", and "c". The splitting algorithm will find such a query and recognize that a result relation

from the "d--------e" piece is not necessary.

4.3.4. Optimality of Split Tactic with Limited Search

The splitting algorithm with limited search attempts to find the optimal sequence in which to process a query. It will optimize for minimum network communications but the algorithm is a "greedy" [WONG77] algorithm, performing a "local optimization" and not a "global optimization". This section explains the characteristics of a limited search. In the following section (4.3.5), the algorithm will be extended to perform an exhaustive search.

The split tactic using limited search bases the cost estimate of performing the second piece of a query on the assumption that the piece will be processed without further splitting. In a four variable query, for example, it does not explicitly consider processing the query as three two variable subqueries (2-2-2). Instead, it examines the cost of doing a two variable followed by a three variable query (2-3). If it decides to split the query as 2-3 then the next processing step will consider splitting the three variable into 2-2, thus achieving the end effect of performing 2-2-2. Limited search works well only if what is best for the current processing step will be best overall. If the optimal strategy is 2-2-2 then it is not unreasonable to assume that the split 2-3 is going to be better than either the 4 or 3-2 splits. Thus, by doing

only a limited search it should be possible to find the optimal processing strategy. The advantage of limited search is that it considers fewer total cases at each decision point and is therefore computationally more efficient. The next section describes query splitting with exhaustive search. Following that, the two methods are compared in detail.

4.3.5. Splitting Technique with Exhaustive Search

The splitting tactic with exhaustive search will examine all possible cases before deciding what the next piece will be. Define e(n) to be the number of cases which exhaustive search must consider for an "n" variable query. E(n) can be expressed recursively as:

$$\begin{aligned} e(1) &= 1 \\ e(2) &= 2 \\ e(n) &= \sum_{i=2}^{n} \left\lceil \begin{matrix} n \\ i \end{matrix} \right\rceil * e(n-i+1) \end{aligned} \qquad (4.16)$$

This formula is an extension of the limited search formula. For each case, 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. Once again the number of actual choices is limited by the user's query, i.e. by the interconnections of the variables in the graph.

The algorithm for exhaustive search is shown in figure 2. The major difference between figure 2 and figure 1 (limited search) is that instead of computing the cost of executing q'' assuming it will be done in one piece,

Algorithm for query splitting using exhaustive search

```
ESPLIT(q, execute)
{
        cost = maxp = trans_cost(q)
        split = FALSE;
        for (i = 2; i < variable_count(q); i = i + 1)
        {
                for (each combination of i variables in q)
                {
                        form q' and q''
                        cost = trans_cost(q')
                        estimate_result_size(q')
                        cost = cost + ESPLIT(q'', FALSE)
                        if (cost < maxp)
                        {
                                maxp = cost
                                save q' and q''
                                split = TRUE
                        }
                }
        }
        if (execute is TRUE)
        {
                if (split is TRUE)
                {
                        process(q')
                        adjust(q'')
                        ESPLIT(q'', TRUE)
                }
                else
                {
                        process(q)
                }
        }
        return (cost)
}
```

Figure 2.

exhaustive search calls itself recursively on q'' in order to determine the least expensive cost for processing q''. The additional parameter to ESPLIT(q, execute) is a flag which states whether to actually execute the query or just compute the cost of executing the query. This is necessary since the recursive call is for measurement reasons and not for executing the query.

As an example, consider again the case of a five variable query. Exhaustive search will examine the cases:

Case	Notation
Perform all 5 variables at once	5
2 var followed by 4 var	2-4
2 var followed by 2 var followed by 3 var	2-2-3
2 var followed by 2 var followed by 2 var followed by 2 var	2-2-2-2
2 var followed by 3 var followed by 2 var	2-3-2
3 var followed by a 3 var	3-3
3 var followed by 2 var followed by 2 var	3-2-2
4 var followed by a 2 var	4-2

4.3.6. Comparison Between Limited and Exhaustive Search

The differences between limited and exhaustive search are the number of cases explicitly considered. Given an n variable query, limited search examines

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

while exhaustive search examines

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

When n = 3, both searches examine the same cases. The first difference arrises when n = 4. In that case, exhaustive search examines the 2-2-2 case explicitly but limited search does not. As n grows, the number of additional cases which exhaustive search must consider grows exponentially compared to limited search. For example, the maximum number of cases for values of n from 2 to 9 are:

n	limited	exhaustive
2	1	1
3	4	4
4	11	29
5	26	336
6	57	5687
7	120	132294
8	247	4047969
9	502	157601068

These numbers assume that every variable is connected to every other variable. It is much more likely that each variable is connected to only one or two other variables. For example, a query such as

$$R_1\text{----}R_2\text{----}R_3\text{----} \cdots R_n$$

would have to examine:

n	limited	exhaustive
2	1	1
3	3	3
4	6	12
5	10	60
6	15	360
7	21	2520
8	28	20160

9 36 181440

In practice most queries involve relatively few variables.

Conventional wisdom on "greedy" algorithms states they are not optimal but are used because they come close to optimality. If this is true, they represent good trade-offs because they examine fewer cases. To verify the merits of the limited search, the algorithm was coded and compared to exhaustive search using a program which is described in the next section.

4.3.7. Simulation Program

A program was written to compare the performance of the limited and exhaustive search algorithms under a large variety of situations. This section describes the conditions the program models and provides an overview of how the program works. The processing algorithms were coded and the program was used to measure a large number of cases under a variety of initial conditions. The issues of interest are:

(1) Number of variables.

(2) The interconnection of the variables. This is the graph of the query.

(3) The number of sites in the network.

(4) The distribution of data among the sites.

(5) The network model (broadcast or site-to-site).

(6) How result sizes are estimated.

(7) The cardinality of each variable.

(8) The size of the target list of each variable. If the size is zero then the variable is not in the target list.

The number of possible combinations is, of course, enormous and therefore several simplifying assumptions must be made in order to make the problem manageable.

Joining domains. It is assumed that each pair of relations is joined on different, unique domains. For simplicity, all joining domains are assumed to be 4 bytes in length. For example, if there is a term such as $R_1.A = R_2.B$, it is assumed that domains A and B are used only for that join and nowhere else in the query. This assumption simplifies keeping track of the size of each relation.

Independence between variables. The input to each test case includes the number of tuples which will satisfy the join between each pair of relations. To reduce the amount of data supplied for each test case, it is assumed that the result size of each join is independent of the order in which the joins are performed. This assumption does not limit the capability of the analysis; rather, it simplifies the amount of data which must be supplied for each case. By supplying different result sizes for the joins, it is possible to examine any case.

FP technique. Since the primary purpose of query splitting is to reduce communications costs, the FP technique was coded using Theorems 1-4. This means R_p is always the largest relation.

The choices for cardinalities for each relation were 10, 100, and 1000 tuples. This represents a two order of magnitude range for the cardinalities and is sufficient to show the effects of the algorithms under a wide variety of cases. The cardinalities can be scaled to represent larger relations (say 1000, 10000, or 100000 tuples) with identical results. It's the relative differences between the cardinalities that is important -- not the absolute values.

The size of the target list for each relation was either 0 or 10 bytes. For each input case, the program ran all possible combinations of relation cardinalities and target list sizes. This means that each variable takes on six cases - three different cardinalities, and two different target list sizes. For example, in a four variable query, the number of combinations would be

$$(3)(2)(3)(2)(3)(2)(3)(2) = 6^4 = 1296$$

or in general 6^n combinations where n = number of variables. Since the case of having a completely empty target list (that is no variable in the target list) is not very common or interesting, it is omitted. This reduces the

number of cases to

$$6^4 - 3^4 = 1215$$

for the four variable case, and in general

$$6^n - 3^n$$

for the n variable case.

To further simplify the study, only the broadcast model was analyzed. The results then become independent of the number of sites in the network and relatively independent of the distribution of data among the sites. This is a direct result of theorems 1 and 2 which state that the number of processing sites for a broadcast network must be either 1 or M_p. By assuming that the number of processing sites is always M_p, the results are completely independent of the distribution of data.

The program accepts as input:

(1) The graph of the query

(2) The true result size of each join

The algorithms are then run on all 6^n-3^n different combinations of cardinalities and target lists. The output for each case is

(1) The cost to run the query without splitting

(2) The cost using exhaustive search

(3) The cost using limited search

(4) A list of the processing steps made.

With the above information, we can compute the true cost of exhaustive and limited search query splitting based on exact information. In other words, as each possible split is considered, the algorithms have available the exact result size which would occur if the query were actually run. We shall call this having "perfect information". By doing this we eliminate all other factors and examine only the difference caused by the algorithms themselves. Note that under these conditions, exhaustive search will always find the optimal strategy.

4.3.8. Exhaustive Versus Limited Search with Perfect Information

The results of running the query:

```
A------B------C------D          QUERY 1.
   10     20     5
```

are shown in figure 3. The numbers above represent the cardinality of the result of the corresponding join. For example, the join between variables A and B will have 10 tuples in the result regardless of the sizes of A and B. The graph in figure 3 represents the cost to solve query 1 in 100 of the 1215 combinations measured. The results are ordered in decreasing difference between limited search and exhaustive search, and the first 100 cases are shown. Each case is independent from every other case. The points are connected by lines only for visual clarity.

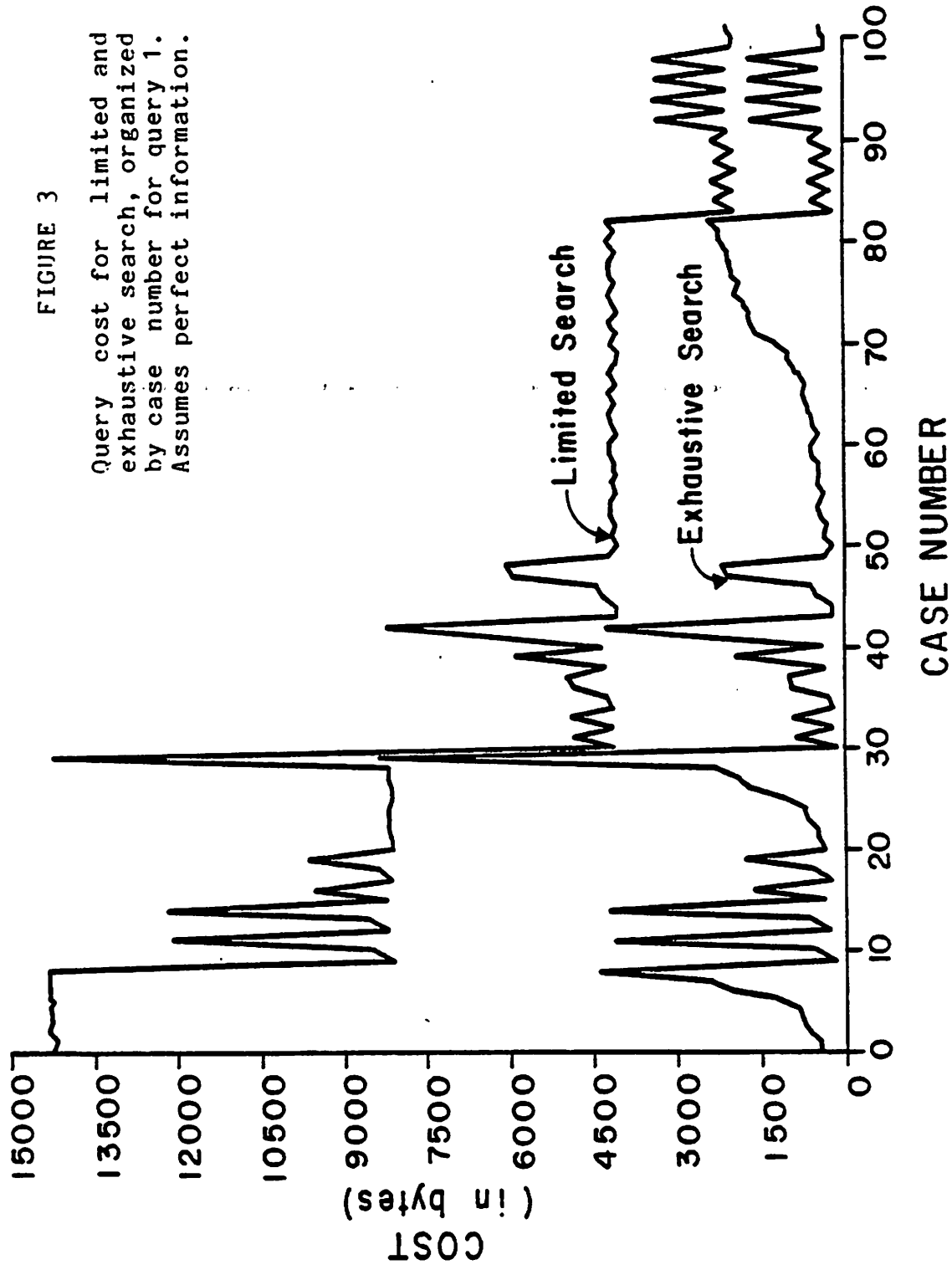


FIGURE 3

Query cost for limited and exhaustive search, organized by case number for query 1. Assumes perfect information.

The lower line is the cost of running the query using exhaustive search and the upper line is the cost using limited search.

Unexpectedly, exhaustive search does dramatically better than limited search. The peaks and valleys on the graph are not important. Since each data point is independent, and they have been arranged in decreasing difference, the lines are not smooth. The worst case difference is the case corresponding to variables A, C, and D having cardinalities of 1000 and target list sizes of 10; and variable B 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 - BC followed by BD followed by AB. Limited search broke the query in to two pieces, CD followed by ABC.

Another way to view the comparison between exhaustive and limited search is shown in figure 4. The X-axis is the percentage of the cost of limited search over exhaustive search. In other words, 50% means limited search moved 1.5 times the amount of data that exhaustive search moved. The Y-axis is the percentage of total cases. The graph shows that for 59% of the cases, limited search performed exactly the same as exhaustive search. For 70% of

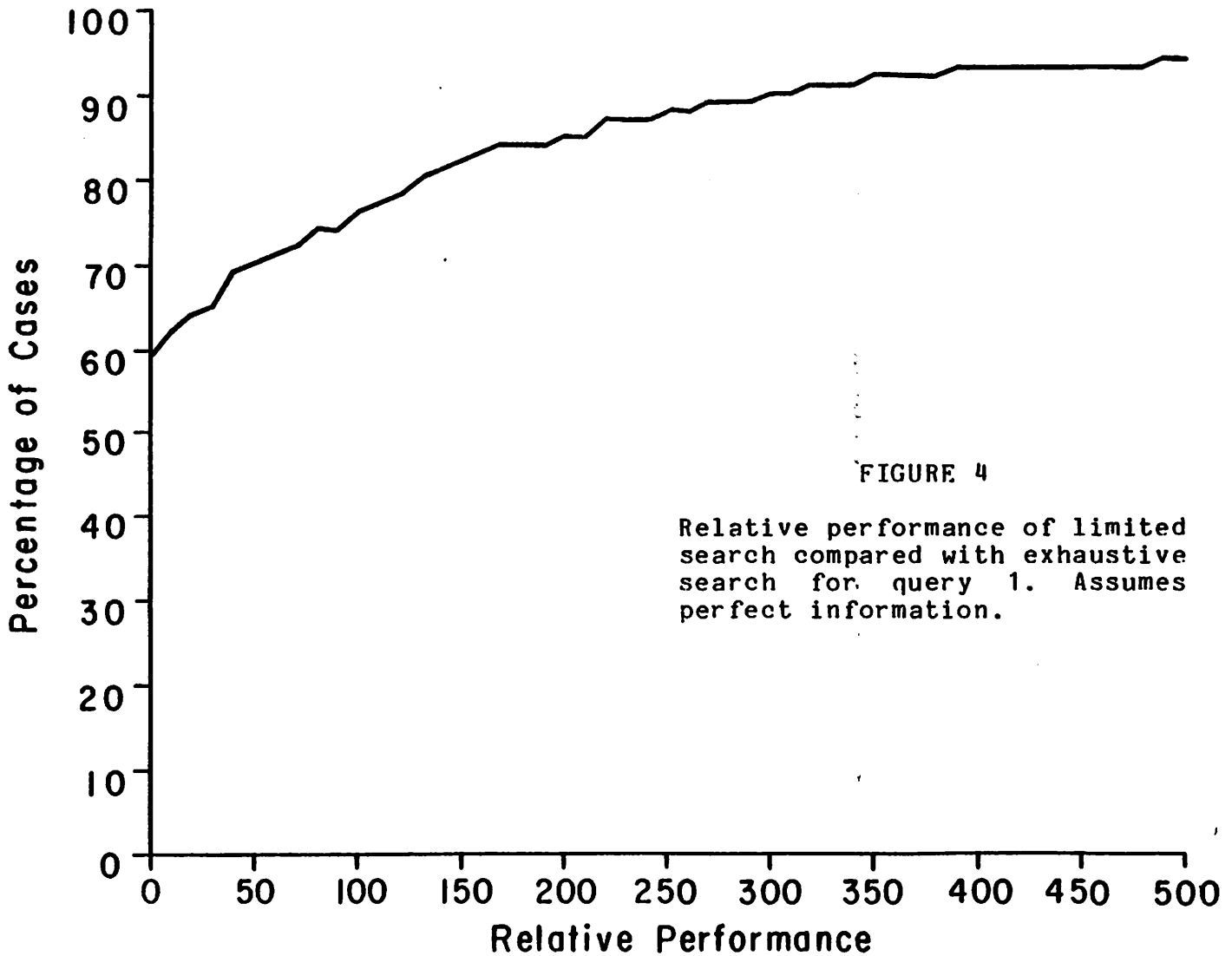


FIGURE 4

Relative performance of limited search compared with exhaustive search for query 1. Assumes perfect information.

the cases, limited search was within 160% of exhaustive search, etc. The line grows very slowly. One would expect limited search to do badly in a few cases but perform within 10% of exhaustive search in most cases. Figure 4 shows clearly that this is not the case.

Further examination of the worst case for limited search from figure 3 shows why limited search does not perform as well as exhaustive search. In that case the optimal strategy was

step	procedure	cost in bytes
BC	move B	80
B'D	move B'	180
AB'	move B'	120
		380

In order for limited search to find the optimal strategy, it would have to find that BC followed by AB'D was the best strategy. The cost for that split would be

step	procedure	cost in bytes
BC	move B	80
AB'D	move B' & A	180 + 14000
		14260

However, the strategy it chose was

step	procedure	cost in bytes
CD	move D	14000
ABC'	move C' & B	80 + 120
		14200

The lowest cost 2-3 split begins with the CD piece. Thus limited search chooses to first move an expensive piece (14000 bytes) and leave an inexpensive move (200 bytes) for the second piece. This shows why limited search does not do as well as one might expect.

To verify that the results are not due to the small result sizes of the join, another case was tried where the result sizes were:

```
A------B------C------D          QUERY 2.
  100    200     2
```

The results for query 2 are shown in figure 5. The results are similar to case 1. The limited search performed much worse than the exhaustive search for a large number of cases. Figure 5 shows 100 points, again sorted according to decreasing difference between exhaustive and limited search. Based on the cases shown here and other cases examined, limited search was dramatically worse than exhaustive search under a wide range of cases.

The results from this section assumed "perfect information". They do not necessarily predict where crude estimation procedures are used. In the next section, a

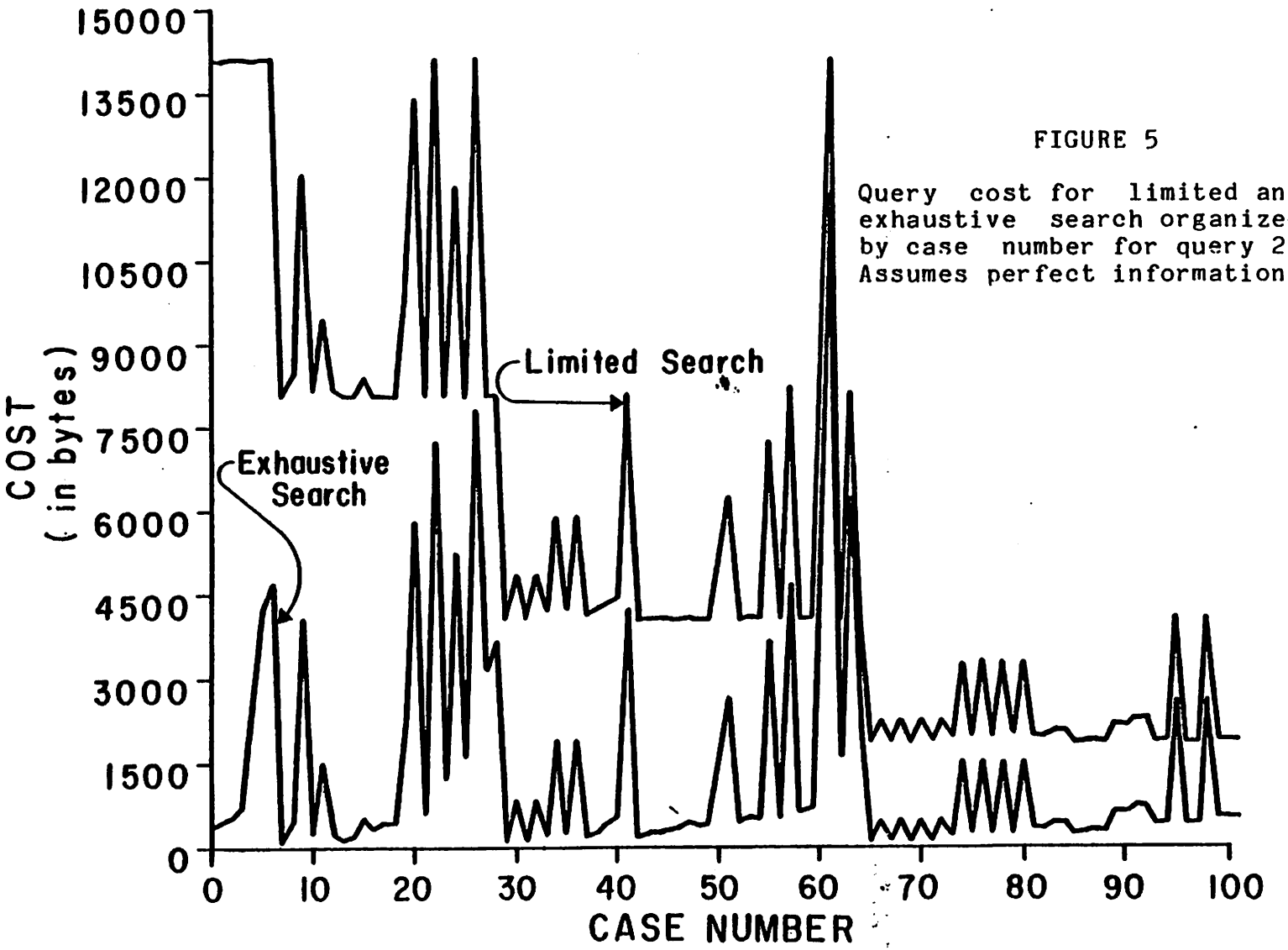


FIGURE 5

Query cost for limited and exhaustive search organized by case number for query 2. Assumes perfect information.

crude estimation procedure is described and the results are given.

4.3.9. Comparison Using Limited Statistics and Dynamic Decision Making

For making estimates with limited information we shall assume we know a relation's cardinality and one bit of imformation per domain. This is case 2 from section 3.9. We shall make "worst case" estimates, that is, what is the maximum number of tuples that can satisfy a given query?

In the worst case the size of the result from any query is limited by the product of the cardinalities of the relations in the target list. In addition the result size of each join is limited according to whether each joining domain has its bit set or not. We shall modify the graphic representations to include a '#' character if the domain has its bit set. the worst case cardinality is:

$$
\begin{array}{lcl}
R_1\#\text{------}R_2\# & = & \min(C_1, C_2) \\
R_1\text{------}R_2\# & = & \max(C_1, C_2) \\
R_1\#\text{------}R_2 & = & \max(C_1, C_2) \\
R_1\text{------}R_2 & = & C_1 * C_2
\end{array}
$$

If the joining domains for both R_1 and R_2 are unique, the maximum number of tuples in the result is limited to the number of tuples in the smaller relation. If only one

joining domain is unique, the maximum number of tuples in the result cannot exceed the number of tuples in the larger relation. Finally, if neither joining domain is unique, the worst case happens when all joining domains have the same value. In that case the cardinality of the result is the product of the cardinalities of the two relations.

The simulations were rerun using crude statistics to determine how limited search performed compared to exhaustive search. Query 1 was rerun with:

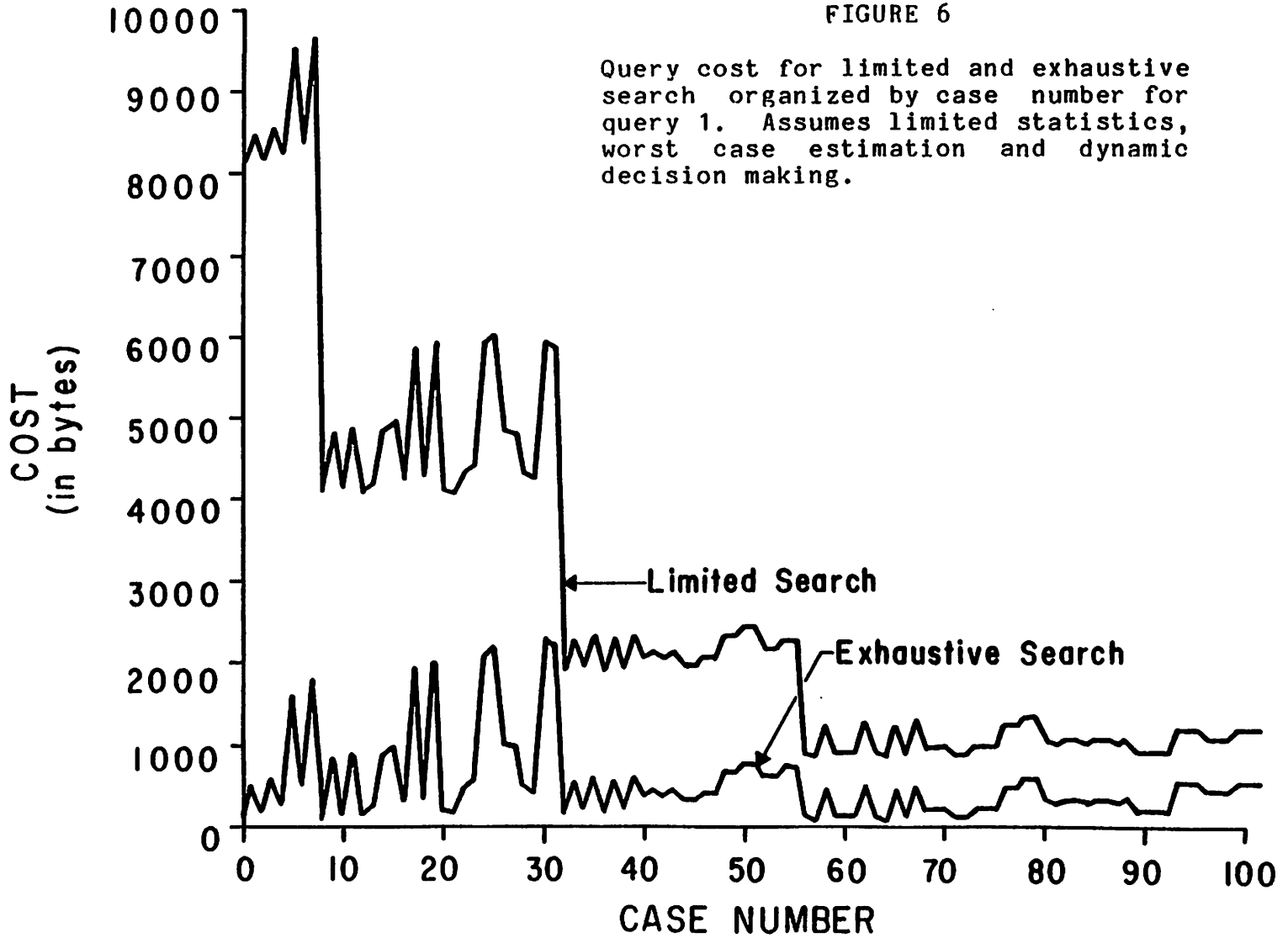
```
A#------#B#------#C#------#D              QUERY 1
    10       20       5
```

All joins are on unique domains. The "split" algorithm used worse case estimates to decide the best splitting strategy. Dynamic decision making was used. After each piece of the query was executed, the results from the query were collected and the remaining strategy was recomputed based on the updated cardinalities of the relations. The results are shown in figure 6.

The differences between limited and exhaustive search are not as dramatic as they were for the perfect information case but they are still conclusive. Limited search does not do well when compared to exhaustive search -- even with very crude statistics.

FIGURE 6

Query cost for limited and exhaustive search organized by case number for query 1. Assumes limited statistics, worst case estimation and dynamic decision making.



A somewhat surprising result is that without perfect information, limited search can do better than exhaustive search! For the case shown in figure 6, exhaustive search did better or the same as limited search in 1083 out of 1215 cases. The cases where limited search did better than exhaustive search are shown in figure 7. If all estimates are accurate, exhaustive search will always find the best strategy. If estimates are imprecise, however, the best strategy may have a higher estimate than some other strategy. Exhaustive search will choose the strategy with the lowest estimate and thus may miss the best strategy if estimates are imprecise. Limited search may still find the best strategy or a better strategy because it may not examine the particular case or cases which misled exhaustive search.

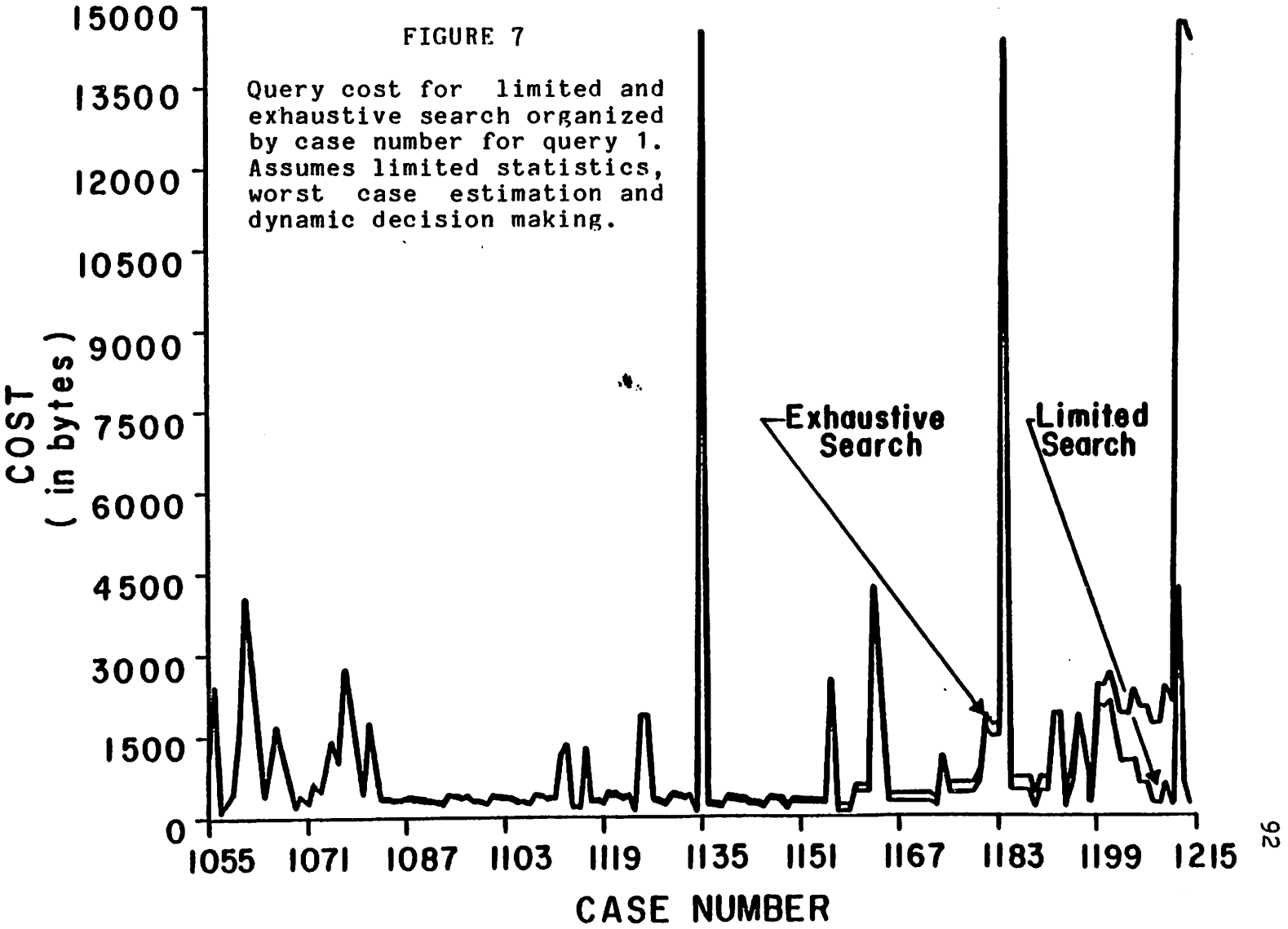
A number of other cases were tried to verify that the results using crude statistics are not sensitive to the sizes of the results of the joins. Figure 8 shows the results of:

```
A#------#B#------#C#------#D          QUERY 2
   100       200       2
```

Once again exhaustive search did much better overall than limited search. There were cases where limited search did better than exhaustive search but the differences were small.

FIGURE 7

Query cost for limited and exhaustive search organized by case number for query 1. Assumes limited statistics, worst case estimation and dynamic decision making.



For query 2, each joining domain was unique. Queries were also run where one or more joining domains were not unique. In each case, limited search performed significantly worse than exhaustive search.

The queries compared so far have been based on either perfect information, or one bit of information per domain with worst case estimation and dynamic decision making. The next section will compare limited and exhaustive search using static decision making and crude estimation.

4.3.10. Comparison Using Limited Statistics and Static Decision Making

As mentioned in section 3.8, static decision making requires less execution time overhead than dynamic decision making. For static decision making, processing algorithms had to be modified slightly. The processing was divided into one decision making phase and one execution phase. After each decision was made during the decision making phase, the relation cardinalities were updated with the estimated result sizes of the queries. Then the next decision was made. Once all decisions are made the execution phase began. During the execution phase, each piece of the query was processed and its actual cost for processing was accumulated. The simulation program kept both the estimated cost for running the query and the actual cost. All graphs of cases using static decision making

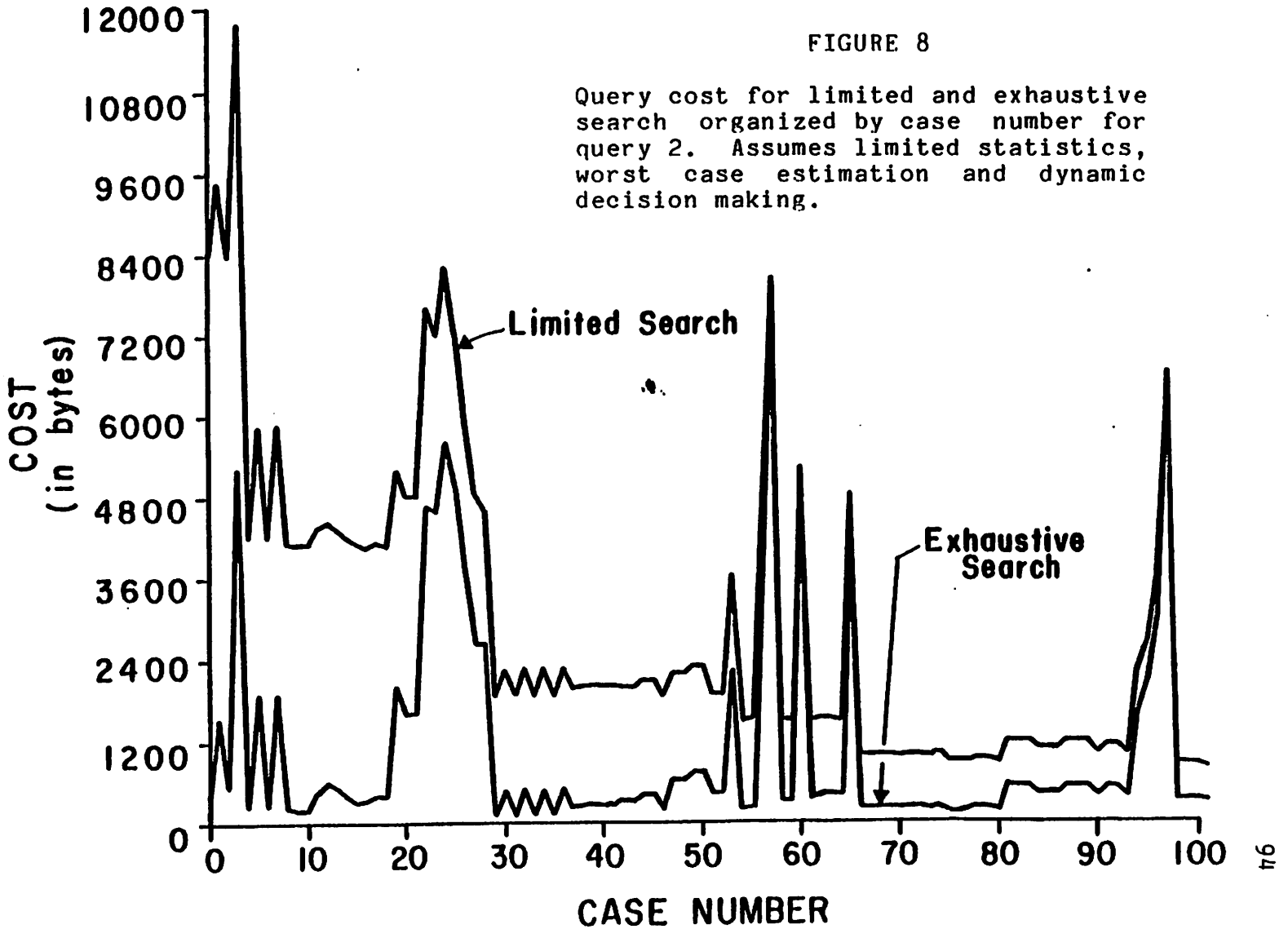


FIGURE 8

Query cost for limited and exhaustive search organized by case number for query 2. Assumes limited statistics, worst case estimation and dynamic decision making.

show the actual costs.

With perfect information static decision making performs the same as dynamic decision making. Thus the test cases for static decision making use the crude estimation procedure. If perfect information was used, the results would necessarily be identical to figures 3 through 5. The results for query 1 are shown in figure 9

```
A#------#B#------#C#------#D              QUERY 1
    10       20       5
```

Once again, exhaustive search does markedly better than limited search but the differences are not as dramatic as in other cases. Other test cases were run which varied the result sizes and the number of unique joining domains. The results were all similar to figure 9. For all the static decision cases, there were times when limited search did better than exhaustive search but once again they were small differences compared to the cases when exhaustive search did better than limited search.

The remainder of this section compares the relative performances of crude estimation versus perfect information and static versus dynamic decision making.

4.3.11. Comparision Between Perfect Information and Crude Estimation

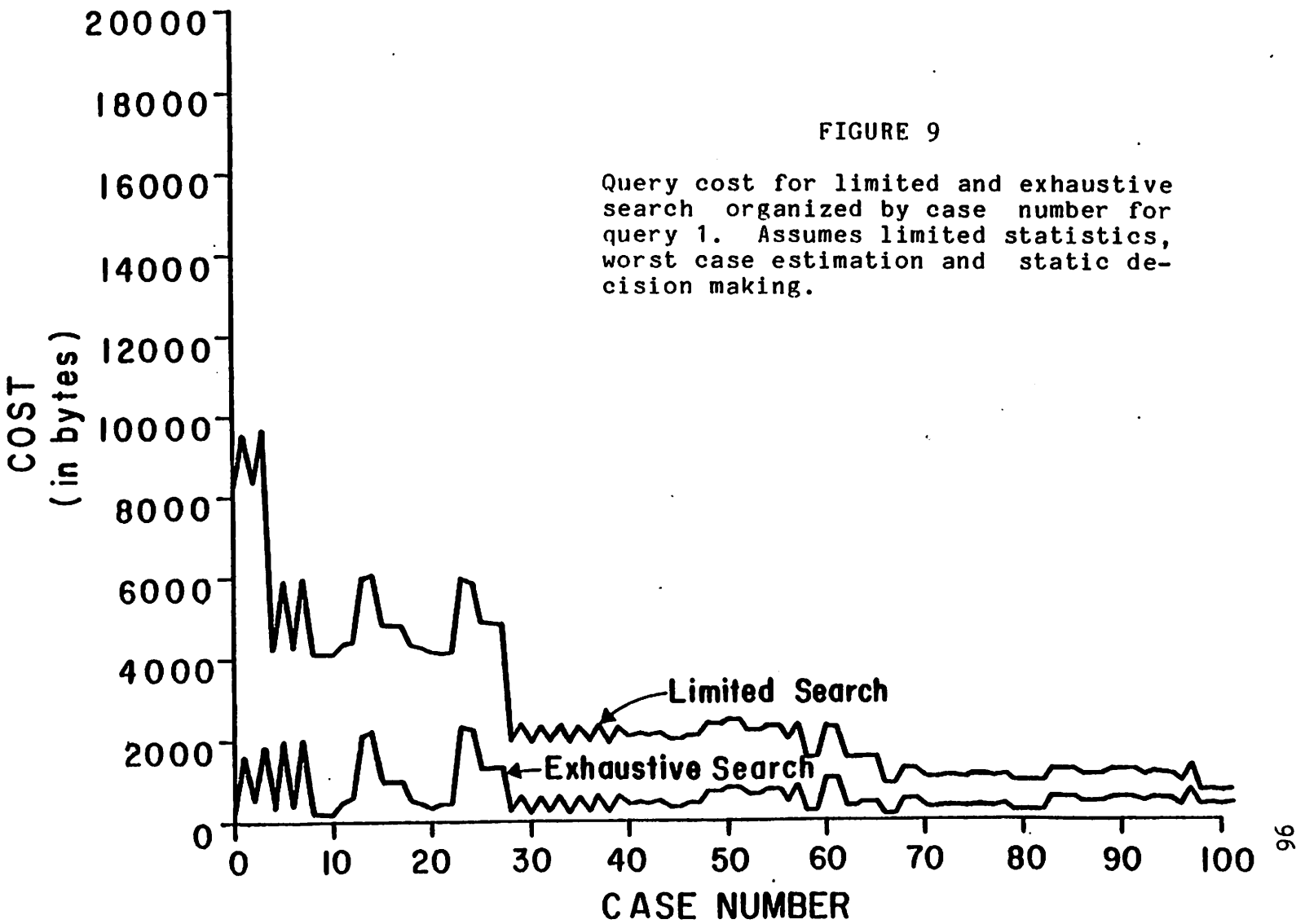


FIGURE 9

Query cost for limited and exhaustive search organized by case number for query 1. Assumes limited statistics, worst case estimation and static decision making.

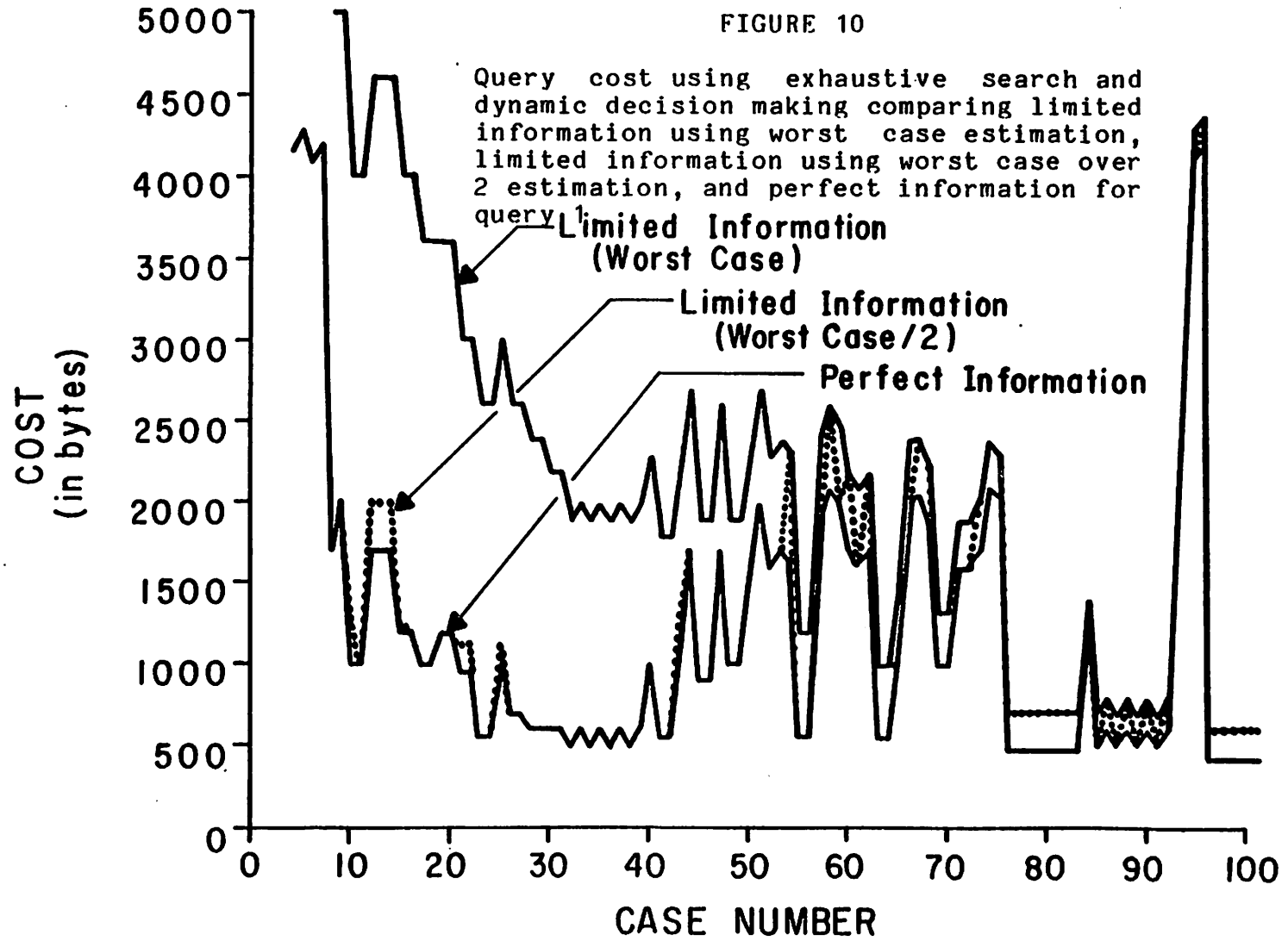
We have now compared limited versus exhaustive search under a variety of conditions. Those results can be used to see how crude estimation compares to perfect information. The results of such a comparison for query 1 are shown in figure 10. The two cases shown are exhaustive search with perfect information, and exhaustive search with worst case estimation and dynamic decision making. Figure 10 shows that even with a small amount of information, worst case estimation performs poorly in some cases but well overall.

So far estimates have been done assuming worst case. That is, it has been assumed that the maximum number of tuples possible would satisfy a query. What would be the effect of assuming only half the possible tuples would satisfy? Such an assumption is closer to what one might expect in a real environment. Notationally we shall call worst case estimation "WC" and worst case estimation divided by 2 "WC/2". The results of rerunning query 1 with estimates based on WC/2 are shown in figure 10. The two solid lines represent perfect information and WC estimation. The dashed line shows the results using WC/2 estimation. Overall WC/2 performs much better than WC.

It was noted that WC estimates did better than WC/2 in 62 out of 1215 cases. This is not surprising since the estimates are so imprecise. It seems reasonable, however,

FIGURE 10

Query cost using exhaustive search and dynamic decision making comparing limited information using worst case estimation, limited information using worst case over 2 estimation, and perfect information for query 1.



that since all of the result sizes for case 1 are small, the smaller the estimate, the better the results. Using WC/10 produced slightly better results than WC/2 estimates for case 1. There are situations where WC/2 is better than WC/10. One can conclude that the better the estimation procedure, the better the final results. However, knowing the relation cardinalities, having one bit of information per domain and performing a WC/2 estimate tends to produce good results in many situations. As final evidence, figure 11 plots the relative performance of perfect information versus WC/2 estimation with dynamic decision making. Note that 75% of the cases are the same and 100% are within a factor of 1.5.

4.3.12. Comparision Between Static and Dynamic Decision Making

The relative performance of static and dynamic decision making is shown in figure 12. There are cases when dynamic decision making does significantly better than static decision making. The overall difference, however, is not very large. It is interesting to note that there are cases when static decision making performs better than dynamic. This is true because the decisions are based on imprecise information. Both static and dynamic decision making always make the same first decision, but dynamic decision making will reevaluate its decisions. It

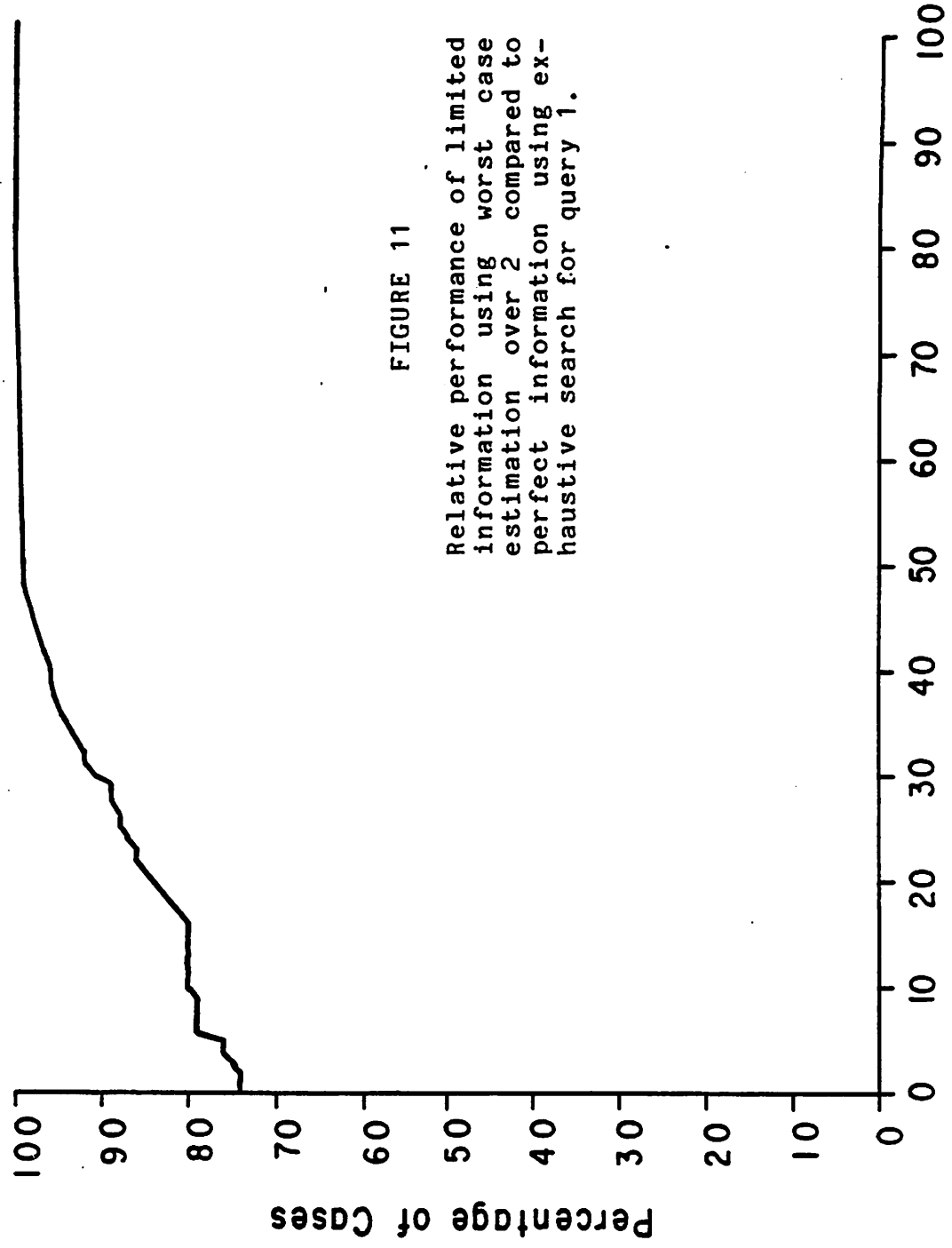


FIGURE 11

Relative performance of limited information using worst case estimation over 2 compared to perfect information using exhaustive search for query 1.

happened on several occasions that static performed better by accident, and dynamic decision making reevaluated itself out of the good decision into one which wasn't as good.

4.3.13. Summary of results

To assign an overall rating to each of the combinations of strategies presented in this section, a single number was derived for each case. This was done by averaging the cost for all test cases for each query. This is a very crude performance measure since it assumes that each test case is equally likely to occur. Figure 13 shows the average performance for queries 1 and 2. The results indicate that exhaustive search performs consistently better than limited search. They also show that the performance of exhaustive search is 30% better when WC/2 estimation is used instead of WC estimation. Using WC/2 and WC/10 estimation made limited search perform worse. For most cases the best strategy was to split the query into three two variable pieces (2-2-2). WC/2 and WC/10 tended to make too small an estimate for three variable pieces. As a result, limited search had a greater tendency to perform 2-3 splitting. The results from query 2 are similar to query 1. In general WC/2 estimation is better than WC estimation.

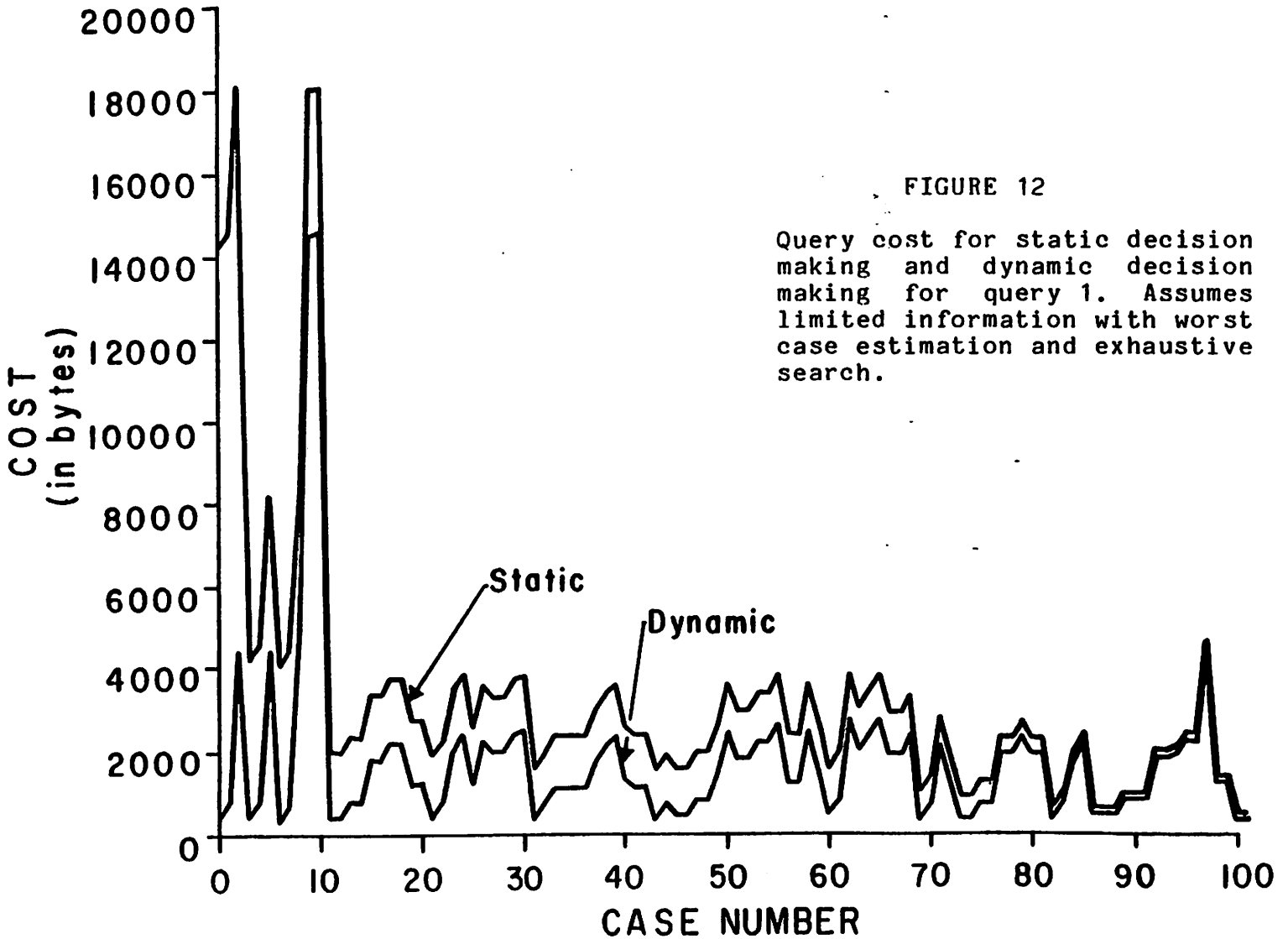


FIGURE 12

Query cost for static decision making and dynamic decision making for query 1. Assumes limited information with worst case estimation and exhaustive search.

Average performance (in bytes).

```
                10       20         5
QUERY 1    A#-----#B#------#C#-----#D
```

	exhaustive	limited
perfect information	508	1086
worst case, dynamic	755	927
worst case/2, dynamic	537	1252
worst case/10, dynamic	532	1053
worst case, static	860	1070
worst case/2, static	545	1257
worst case/10, static	536	1058

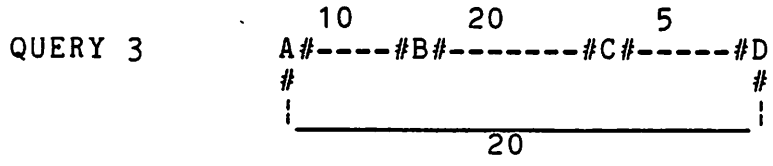
```
                 100      200      2
QUERY 2       A#-----#B#-----#C#----#D
```

	exhaustive	limited
perfect information	551	1036
worst case, dynamic	995	1128
worst case/2, dynamic	746	1403
worst case/10, dynamic	740	1207
worst case, static	1017	1210
worst case/2, static	781	1420
worst case/10, static	774	1229

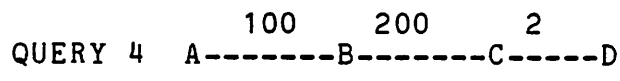
Figure 13

The average results for two other queries are shown in figure 14. Query 3 differs from queries 1 and 2 because every variable is joined to two other variables. Query 4 is similar to query 2 except that no domains have their "unique bits" set. This is equivalent to knowing only the relation cardinality (case 1 of section 3.9). Therefore the estimated number of tuples for WC estimation is always the product of the two cardinalities.

Average Performance (in bytes)



	exhaustive	limited
perfect information	631	1340
worst case, dynamic	1185	1183
worst case/2, dynamic	721	2128
worst case/10, dynamic	721	1471
worst case, static	1259	1289
worst case/2, static	724	2128
worst case/10, static	724	1475



	exhaustive	limited
perfect information	508	1086
worst case, dynamic	3581	4008
worst case/2, dynamic	3372	3541
worst case/10, dynamic	3307	3179
worst case/1000, dynamic	852	1240
worst case, static	3466	4008
worst case/2, static	3300	3549
worst case/10, static	3206	3319
worst case/1000, static	876	1290

Figure 14

Dynamic decision making performs somewhat better than static decision making but the differences tended to be in the 1% to 30% range. If the overhead for performing dynamic decision making is low, then it is a good tactic to use. If it is expensive to perform dynamic decision making, then static decision making appears to perform

well. The costs of dynamic and static decision making are examined in more detail in section 4.7.

4.4. Transmission of Physical Structure

Once the FP technique has chosen a fragment to be transmitted, the choice remains as to what to transmit. It may be possible to improve local processing by transmitting additional information such as a relation's index. Three choices will be analysed:

(1) moving a projected, sorted, duplicate free copy of the fragment,

(2) moving the complete fragment together with its primary structure (clustered index [GRIF79], isam directory [IBM66], hash division for modulo hashing [KNUT73]),

(3) moving the complete fragment together with a secondary structure (non-clustered index [GRIF79], secondary index [HELD75b]).

Our goal is to determine when any of these options will minimize either the response time or communications traffic.

The analysis will proceed as follows: First, cost functions will be derived for performing equi-joins on local, two variable queries. The joins will be computed using tuple substitution [STON76], tuple substitution with reformatting [STON76], and sort-merge substitution

[BLAS76]. The two variable equi-join case was chosen for analysis because it is the simplest case which illustrates the effects of moving a relation's structure. Later in this section the solution for more general joins will be discussed. The cost functions for equi-joins will then be extended to include fragmented relations. The degree to which the primary and secondary structures aid local processing will then be determined. The cost functions will be measured in disk pages accessed, as this reasonably models the relative performance of the three choices.

4.4.1. Equi-join Cost Functions for Local Relations

Given two complete relations, R_1 and R_2, the equi-join can be computed by iteratively substituting each tuple of either relation.[1] Suppose R_1 is chosen. A tuple is read from R_1 and all corresponding references to R_1 are replaced by their values. The remaining query references only R_2. In the most optimistic case, R_2 has a primary structure on the joining domains, and only one page has to be read from R_2 for each tuple in R_1. The cost formula measured in pages in this case would be

$$COST_{pages} = P_1 + 1\ C_1$$

where

[1] See chapter 1 for a discussion of tuple substitution, sort-merge join and reformatting.

P_i = number of pages occupied by R_i

C_i = cardinality of R_i.

If the structure used to access R_2 is a secondary structure, then the most optimistic cost estimate would be

$$COST_{pages} = P_1 + 2\ C_1$$

based on one access to the secondary structure and one to the corresponding primary data page. The formula is optimistic. Both these formulas ignore the cost of searching a directory when necessary.

If R_2 has no useful structure, then the cost would be

$$COST_{pages} = P_1 + C_1P_2$$

since each page of R_2 would have to be read for every tuple in R_1. Alternatively, R_2 could be reformatted to a hash structure on the joining domain. In that case the total cost (again measured in pages) would be

$$COST = P_1 + 1\ C_1 + F_2$$

where

F_i = cost to reformat R_i to hash.

Assuming uniform distribution of keys, the cost to reformat is approximately

$$P_i LOG_{10} P_i$$

To reformat a relation, the tuples are first sorted according to their hash value and then inserted into the

relation. Since the tuples are in hash order, insertion into the relation is a linear cost. The cost for sorting depends on the amount of main memory available for sorting and the number of files which can be merged simultaneously. If there is enough memory to sort B pages, then in one pass through the relation, P_i pages will be sorted into $2P_i$ / B pieces at a cost of

$$2\frac{P_i}{B}$$

The constant 2 represents the fact that each page must be read and then written back. The P_i / B pieces can be merged back together in an "n" way merge at a cost of:

$$2\frac{P_i}{B} \text{ LOG}_n \lceil P_i \rceil$$

The INGRES system is capable of using n = 10. Since this is a reasonable number, LOG_{10} will be used throughout. The reformat cost is:

$$2\left\lceil\frac{P_i}{B}\right\rceil \text{ LOG}_{10} \lceil P_i \rceil$$

For the purposes of simplifying the analysis, the reformat cost will be simplified to

$$P_i \text{LOG}_{10} P_i$$

Since the above formula is greater than the actual cost, it will favor moving a relation's structure. The total cost for tuple substitution with reformatting is

$$COST = P_1 + 1\ C_1 + P_2 LOG_{10} P_2$$

The sort-merge join strategy [BLAS76] requires that each relation be sorted on the joining domains. The join can typically be computed by reading each relation once. The cost for such a technique (assuming both relations are already sorted) is

$$COST = P_1 + P_2$$

If only one relation (say R_1) is already sorted on the joining domain(s) then the cost is

$$\begin{aligned} COST &= P_1 + P_2 + P_2 LOG_{10} P_2 \\ &= P_1 + P_2 \lceil LOG_{10} P_2 + 1 \rceil \end{aligned}$$

If both relations need to be sorted then the cost is

$$\begin{aligned} COST &= P_1 + P_2 + P_1 LOG_{10} P_1 + P_2 LOG_{10} P_2 \\ &= P_1 \lceil LOG_{10} P_1 + 1 \rceil + P_2 \lceil LOG_{10} P_2 + 1 \rceil \end{aligned}$$

4.4.2. Equi-join Cost Function for Distributed Relations

The cost estimates will now be extended for distributed relations. If the FP technique is applied to a two variable query on R_1 and R_2, each site S_j will have:

R_1^j

JOINED WITH

$R_2^1 \quad R_2^2 \quad \cdots \quad R_2^j \quad \cdots \quad R_2^n$

At each processing site S_j, there is a logical, complete relation referred to above as R_2, and a fragment of the relation R_1^j. The physical relation R_2 might be stored as 1, 2, or n separate relations. If the fragment R_2^j is the original user relation, then the other fragments, $R_2^i, i \neq j$, could be assembled into one relation or kept as separate relations. If R_2^j is already a copy of the original R_2^j (as would happen if a one variable restriction had previously been performed on R_2^j), then the other fragments of R_2 could be appended directly to R_2^j or left as separate relations. If the physical structure of the relation fragments is to be preserved, they must be left as n separate relations. Access methods, by necessity, must place physical restrictions on the locations of tuples based on their key values. The author knows of no way to combine two relation fragments, maintained separately, and still preserve the physical structure of the individual fragments. For this reason, if the structure is to be preserved, the $R_i, i \neq p$, relations must be left in their

composite pieces.

The cost for tuple substitution assuming that the composite relation is not the relation being substituted is

$$COST = P_1 + (C_1)(n)$$

where

$n = M_2$ = the number of fragments of R_2.

This again assumes hashing on the joining domain. For each tuple in R_1, one page of each of the n relation fragments must be accessed. (Throughout this section whenever assumptions are made, they will be made in favor of moving a relation's structure.)

If the relation fragments $R_2^i, i=1,\ldots,n$ were put into one unstructured relation, then tuple substitution could either sequentially scan R_2 at a cost of

$$P_1 + C_1P_2$$

or R_2 could first be reformatted making the total cost

$$P_1 + C_1 + P_2LOG_{10}P_2$$

Therefore the cost for tuple substitution is

$$COST=MIN\left[P_1 + C_1P_2,\ P_1 + C_1 + P_2LOG_{10}P_2\right]$$

The sort-merge join requires that both relations be sorted on the joining domains. Thus, in general, the fragments of R_2^i will have to be one sorted relation. The

cost for performing the sort-merge on a relation which is composed of a collection of fragments is the same as for a local relation.

The formulas developed in this section are summarized below:

Summary of query processing formulas

$P_1+(n)C_1$

Tuple substitution for R_1, R_2 has useful primary structure

$P_1+2(n)C_1$

Tuple substitution for R_1, R_2 has useful secondary structure

$P_1+C_1P_2$

Tuple substitution for R_1, R_2 has no useful structure

$P_1+C_1+P_2LOG_{10}P_2$

Tuple substitution for R_1, reformat R_2

P_1+P_2

Sort merge with both relations already sorted

$P_1+P_2(LOG_{10}P_2+1)$

Sort-merge with R_1 already sorted

$P_1(LOG_{10}P_1+1)+P_2(LOG_{10}P_2+1)$

Sort-merge with neither relation sorted

4.4.3. Effectiveness of Transmitting a Relation's Structure

The formulas for estimating the costs of processing two variable queries on distributed data bases have now been identified. Note carefully that any assumptions have been made in favor of moving a relation's structure. We will now analyze the effectiveness of moving a relation's structure; first assuming tuple substitution will be used, and second assuming the sort-merge join will be used. Formulas which show the cost effectiveness of moving a relation's physical structure will be developed. Finally, all combinations of the processing tactics will be compared.

Observe that transmitting a projected, duplicate free fragment minimizes communication traffic. Any domains unnecessary for processing the query will have been eliminated. The removal of duplicates at an early stage helps subsequent processing. This is clearly the best tactic for minimizing communication traffic.

Any tactic which sends more than the minimum amount of data must improve local processing time by at least the time required to transmit the extra data.

If tuple substitution is used as the join tactic, the transmission of the fragment's structure is useful only if the relation is not the one chosen for substitution. The

relation being substituted is read sequentially and its structure, if any, is ignored. If the relation is the "inner relation", i.e., the one not chosen for substitution, and the primary structure is transmitted, then the cost formula is

$$COST = P_1 + (C_1)(n)$$

If the relation fragments R_2^i are transmitted without structure and reformatting is done then the cost formula is

$$COST = P_1 + C_1 + P_2 LOG_{10} P_2$$

In practice, reformatting is nearly always effective except in cases of small relations [YOUS78b]; thus it is reasonable to assume reformatting takes place. Comparing the two costs we have:

$$P_1 + C_1 n < P_1 + C_1 + P_2 LOG_{10} P_2$$

$$n < 1 + \frac{P_2 LOG_{10} P_2}{C_1}$$

Ignoring the constant 1, our final result is

$$n < \frac{P_2 LOG_{10} P_2}{C_1} \qquad (4.16)$$

If the inequality holds, it <u>may be</u> cost effective to transmit a primary structure (it must improve performance by at least the added overhead of transmitting the structure). If the choice is to transmit a secondary structure, then the most optimistic assumptions would be that

$$P_1 + 2C_1n < P_1 + C_1 + P_2LOG_{10}P_2$$

$$n < \frac{P_2LOG_{10}P_2}{2C_1}$$

The case for the sort-merge join is entirely different. For such a tactic both relations must be sorted. The relation fragments should be transmitted already sorted so that they need only be merged at the processing site. Thus for the sort-merge join tactic, the minimum communication cost and the minimum local processing cost will both occur by transmitting the projected, duplicate free, sorted fragments.

We have shown that moving a relation's structure is not an effective tactic for sort-merge. Theorem 5 will prove that assuming a reasonable number of sites, and a reasonable number of tuples per page, equation 4.16 is never satisfied. This means that moving a relation's structure is not an effective tactic. To prove this we will make the assumption that moving the structure must be cost effective on a majority of sites. This allows us to reasonably assume that on each site,

$$nP_1 \geq P_2$$

where

R_1 = the relation left fragmented

R_2 is the composite relation at each site.

This is true because the FP technique will only choose R_1

to remain fragmented if it is larger than R_2. Once the fragments of R_2 have been moved, then each site that has the complete copy of R_2 has on the average 1/n of R_1.

THEOREM 5:

Suppose R_1 is left fragmented and R_2 is the relation which is duplicated on all processing sites. Given that R_1 has at least 10 tuples per page and that there are at least two fragments of R_2, then moving the structure of R_2 is better than reformatting R_2 only if $P_1 > 10^5$.

PROOF:

We must show that

$$P_1 + C_1 n > P_1 + C_1 + P_2 LOG_{10} P_2$$

Subtracting P_1 from both sides gives:

$$C_1 n > C_1 + P_2 LOG_{10} P_2$$

The number of tuples in R_1 is equal to the number of pages P_1, times the number of tuples per page, which we shall refer to as 'x':

$$C_1 = xP_1$$

Subtracting C_1 from both sides and substituting for C_1 gives

$$xP_1 n - xP_1 > P_2 LOG_{10} P_2$$

It is known that

$$nP_1 \leq P_2$$

otherwise the FP technique would not have chosen R_1 as the relation to remain fragmented. Substitute nP_1 for P_2. This substitution can only increase the right hand side. Thus this substitution favors moving the structure.

$$xP_1n - xP_1 > nP_1LOG_{10}\lceil nP_1 \rceil$$

Dividing by P_1 gives

$$xn - x > nLOG_{10}\lceil nP_1 \rceil$$

Dividing all terms by n gives:

$$x - \frac{x}{n} > LOG_{10}\lceil nP_1 \rceil$$

Factoring out x from the left side and dividing yields:

$$x > \frac{LOG_{10}\lceil nP_1 \rceil}{1 - \frac{1}{n}}$$

We were given that x, the number of tuples per page, is 10. Since there are at least two fragments of R_2, the number of sites, n, must be ≥ 2. With two sites we have

$$5 > LOG_{10}\lceil 2P_1 \rceil$$
$$10^5 > P_1$$

QED

It should be noted that n = 2 and 10 tuples per page are both very low numbers and they can reasonably be expected to be larger. If they are larger it makes moving a relation's structure even less effective. Even so, R_1 (and correspondingly R_2) must be enormous, e.g. over 100,000 blocks or with 4096 bytes/block, 400 megabytes, before moving the structure could be effective! At n = 2, P_1 would have to be over 10^8 pages! Even then, the additional data which must be transmitted to move the structure has not been considered. Doing so would only make it less favorable to transmit a relation's structure.

It has now been demonstrated that moving the structure of a relation is very rarely (if ever) a cost-effective tactic. To illustrate the meaning of theorem 5, the various processing strategies are plotted in figure 15. The lines represent:

(1) Tuple substitute for R_1, R_2 is unstructured

(2) Tuple substitute for R_2, R_1 is unstructured

(3) Tuple substitute for R_2, R_1 is well structured

(4) Tuple substitute for R_2, R_1 is reformatted to be well structured

(5) Tuple substitute for R_1, R_2 is reformatted to be well structured

(6) Sort-merge join with R_1 already sorted

(7) Sort-merge join with neither relation initially

sorted

(A) Tuple substitute for R_1, transfer primary structure of R_2, assume 5 sites in the network

(B) Tuple substitute for R_1, move the primary structure along with R_2, assume 10 sites in the network

(C) Tuple substitute for R_1, move the primary structure along with R_2, assume 50 sites in the network

Figure 15 plots the various strategies for one sample case. For figure 15, R_1 is the relation left fragmented and R_2 is a composite of the n fragments of R_2^j. The number of pages of R_1 is fixed at 1000. The number of tuples per page is fixed at 100 for both R_1 and R_2. Note carefully that R_2 must be smaller in total bytes than n times R_1, since the relation left fragmented must always be the largest relation (section 4.2). The limits where

$$|R_1| = |R_2|$$

for cases A, B, and C are indicated by vertical lines.

From the graph it can be seen that moving the structure (choices A,B,C) is never better than reformatting R_2 (choice 5). The result of Theorem 5 proved this.

In conclusion, moving the structure of a relation can only be an effective strategy if there are very few tuples per page in R_1. Figure 15 shows that for a variety of parameters, the best processing strategy is either 3, 4,

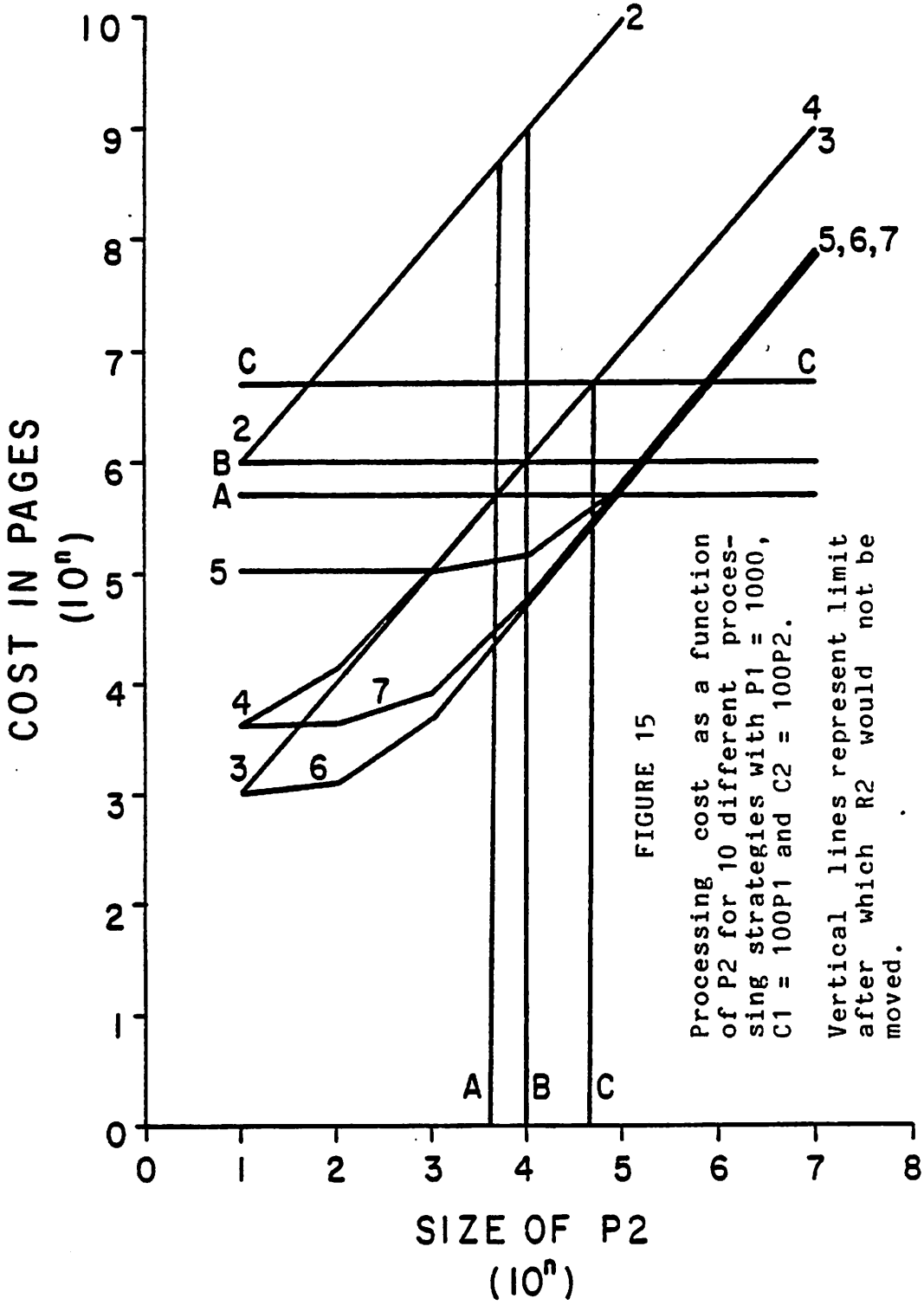


FIGURE 15

Processing cost as a function of P2 for 10 different processing strategies with P1 = 1000, C1 = 100P1 and C2 = 100P2.

Vertical lines represent limit after which R2 would not be moved.

5, 6, or 7.

4.5. Analysis of Join Encoding Techniques

When two relations are joined and the result transmitted to other sites, network communication costs may be lower if the join is transmitted in some encoded form. The three transmission techniques which will be evaluated are

(1) transmitting the actual computed join,

(2) encoding the join as a tuple from relation R_i followed by the tuples from R_j which are to be joined to the tuples from R_1,

(3) storing the tuples which belong to the join as two separate relations, transmitting the relations separately, and recombining the relations at the destination.

The three techniques model, respectively, a 1:1 relationship, a 1:n relationship, and an n:m relationship.

This analysis will be concerned only with minimizing the amount of data which must be transmitted. Once tuples reach their destination the three cases have different impacts on local processing time. Case 1 requires no extra local processing time since the join has already been completely formed. Case 2 requires only a minimum of extra processing since the tuples are transmitted in the order that makes it trivial to reconstruct the join. Case

3 can have a major impact on local processing time since the join must be recomputed from scratch.

The analysis will consider only equi-joins. Extensions to other joins will be considered later in this section. For simplicity, if the join is on two or more domains, then the term "joining domain" should be understood as the concatenation of the domains. The three encoding techniques will be evaluated assuming:

(1) the joining domains from both relations contain only unique values,

(2) the joining domain from only one relation contains only unique values,

(3) the joining domains from both relations contain duplicate values.

If the values in each of the joining domains are unique, then a tuple can match at most one other tuple. In such a situation, actually forming the join is always the best tactic. This is because the other encodings require additional information to be transmitted in order to reconstruct the join. The joining domain itself does not have to be saved unless it is referenced in a subsequent query.

If only one of the relations has unique values in its joining domain, then each tuple from that relation can match multiple tuples in the other relation. This is a

one-to-many relationship. Forming the join will generate redundant information since domains in the first tuple will be repeated for each of the tuples in the second relation which satisfy the equality. Case 2 of the encoding techniques will be the optimum since the tuple would actually be stored in the non-redundant, one-to-many fashion. Case 3 would not be as efficient as case 2 because the joining domains from both relations would have to be saved in order to reconstruct the join.

If neither joining domain contains unique values, then an n-to-m relationship can exist. In such a situation, both cases 1 and 2 could have tuples stored redundantly but case 3 would not. Case 3, however, must store the joining domain value twice, once in each output relation. Cases 1 and 2 store the joining domain once but only if it is needed in subsequent queries. As an example of this case, consider the query from section 3.5:

```
retrieve into temp (e.name, e.number, e.dname)
        where
                e.manager = d.manager
```

If encoding technique 1 were used, the join would be computed and stored in one relation:

```
temp (e.name, e.number, e.dname)
```

If encoding technique 3 were used, the join would be computed using two separate relations:

e' (e.name, e.number, e.manager)
d' (d.dname, d.manager)

Case 1 would be better than case 3 if:

$$|temp| < |e'| + |d'|$$

In this example, the correct choice between cases 1 and 3 is not obvious since e' and d' must include the "manager" domain but temp does not.

So far we have considered only equi-joins. If relations are not being joined with an equi-join it is the same situation as the case where neither joining domain was unique.

In summary, there are at least three encoding techniques for transmitting joins, case 1 which models the case of a 1:1 relationship, case 2 which models a 1:n relationship, and case 3 which models an n:m relationship. If the relationship between the two joining domains is known in advance then the appropriate encoding technique can be chosen with the exception of the n:m case. In the n:m case, the decision depends on whether the joining domains need to be included in the transmission.

4.6. Analysis of Shuffling Strategies

As was discussed in section 3.7, when performing update queries several sites may have tuples which must be moved to one or more different sites. Each processing

site has a relation holding result tuples from the query. The tuples either have a domain that states which site each tuple belongs to, or the distribution criterion has to be used to determine which site the tuple belongs to. Shuffling strategies determine how to move all tuples to their destination site. We will examine which of two "shuffle" strategies, centralized or decentralized, is better regarding minimization of communication traffic, of local processing time, and of the number of messages to be transmitted.

The analysis will proceed as follows. There are K sites which processed the very last piece of the query. M_r is the number of sites where the original result relation resides. For simplicity, we will assume that all messages are to remote sites. This means that even if the master sends a message to itself, it is counted as a network message.

Case (a): Centralized shuffling

The master begins by telling each of the K sites to examine their solution relation for tuples which are already at the correct site, process those tuples and then send the remaining tuples to the master site. This requires one message on a broadcast network and K messages on a site-to-site network. To return the results requires

K messages on either network. Next the master site sorts the tuples, determining the destination for each tuple. The tuples are then sent to their correct sites and the sites told to process the updates. This requires M_r messages and another complete transmission of the result tuples. Finally each site that received tuples acknowledges completion of processing requiring M_r messages. Therefore, a site-to-site model requires $2k + 2M_r$ messages and a broadcast model requires $K + 2M_r + 1$ messages. The size of the result is reduced by the number of tuples already at their correct site. The remaining tuples are transmitted twice. Each of the K sites must scan the solution relation once to determine the tuples that belong to it.

Case (b): Decentralized shuffling

Each site is told to begin processing, requiring one message on a broadcast model and K messages on a site-to-site model. Each site examines which tuples belong to it. Next, every processing site K transmits to every site in M_r, the tuples that belong to them. This requires KM_r messages. Each site in M_r acknowledges completion of processing requiring an additional M_r messages.

Both techniques take the shuffle process to the same stage of completion. For recovery and consistency rea-

sons, a special synchronization (a two-phase commit [ESWA76, STON78]) will occur after the shuffle to actually commit the updates. The shuffle's function is only to distribute result tuples to their correct destinations. Table 4.2 summarizes the two cases.

	Centralized	Decentralized
Messages (broadcast)	$1 + K + 2M_r$	$1 + M_r + K\ M_r$
Messages (site-to-site)	$2K + 2M_r$	$K + kM_r + M_r$
Data movement	local data processed, remaining data moved twice.	local data processed, remaining data moved once.
Processing	one query in parallel at K sites, followed by processing at master site.	M_r queries at K sites

Table 4.2: Summary of Shuffle Strategies

Table 4.2 shows that the minimum data movement will occur with decentralized control. The minimum number of messages occurs with centralized control. The least amount of parallel processing occurs with centralized control.

The choice between centralized and decentralized shuffling depends primarily on the communication network. Decentralized control requires that each site be able to concurrently open a channel to all other sites. Further-

more, decentralized control requires the communication bandwidth to be larger than the sum of the local processors' ability to supply data; otherwise, the processing sites will contend with each other for access to the network. The trade-off between the two choices depends on which will better utilize the network. If messages are insignificant in cost and the network is fast enough to minimize the possibility of contention, then decentralized shuffling is the best choice. Otherwise, centralized shuffling should be chosen.

4.7. Decision Making and Estimation

The aspects of decision making and strategy selection which are particular to distributed data bases are discussed in this section. Emphasis will be placed on estimating result sizes, dynamic decision making, and the movement of the result relation. We will begin with some general comments concerning the relationships between these three concepts. Then we will discuss those aspects which require particular attention on distributed data bases.

The advantage of dynamic decision making, as demonstrated in section 4.3, is the ability to use results from the current processing step for choosing the next processing step. This is particularly important if the estimation procedure is crude. As the accuracy of estimation

improves, dynamic decision making is no longer important since with accurate estimates, both static and dynamic decision making will generate the same strategy.

A similar situation exists for determining whether to move the result relation. With limited estimation capability, there is no way to determine whether to move the result relation. With accurate estimation the cost functions for the FP technique can take into account the additional costs associated with moving the result relation. As mentioned in section 3.6, if the result relation is moved, additional network communication will be necessary to transmit the result tuples (or deletions) back to their correct sites. If the result relation is not moved, then there is no additional network communication. Without a good estimate of how many tuples in the result relation will be changed, there is no way to include the additional cost resulting from moving the result relation. Accurately determining the cost-effectiveness of moving the result relation is beyond the scope of this thesis. It is, however, an important problem and well worth further investigation.

On a centralized data base, the "cost" of doing dynamic verses static decision making is the extra processing required at runtime to make the decisions. This cost is, of course, highly dependent on the specific

implementation. There have been no studies made which examine the cost of dynamic decision making during "decomposition" or query strategy selection. Based on coding the query processing strategies for INGRES, it is the opinion of this author that the cost on a centralized data base for dynamic decision making can be made small. On a distributed data base, the cost for dynamic decision making may be quite large due to the additional message traffic. As an example, we repeat the query from section 3.8. Suppose every site has a fragment of the two relations, parts and supply, and the query is issued:

```
retrieve (p.pname)
        where
                p.pnum = s.pnum
        and
                s.snum = 475
        and
                s.shipdate = "79-07-01"
```

The processing strategy is to detach and execute the one variable restriction on supply and then, using the FP technique, process the join. The FP technique will move either p or s to all sites. Using both dynamic and static decision making, we will count the number of messages needed to control the processing of the query.

Beginning with static decision making, suppose s is chosen as the relation to be moved, leaving p as the relation left fragmented. In one message the master can issue the set of commands to all N processing sites: restrict

supply, move the restriction of supply to all other sites, when all fragments of supply have been received from the other sites perform the join and return a "done" signal. This execution strategy requires one message on a broadcast network and N messages on a site-to-site network.

Now consider the case of dynamic decision making. The decision whether it is better to move p or s is not made until after the restriction on s is performed. The master will issue the command to do the one variable restriction. This will require one message on a broadcast network and N messages on a site-to-site network. The processing sites then send back the sizes of the restricted supply relation, requiring N messages on either network. Now the master chooses p or s to be moved and issues the command to move either p or s, perform the join, and send a "done" when complete. This requires one message on a broadcast network and N messages on a site-to-site network.

For this example, there is a major difference in the number of messages between the two cases:

	broadcast	site-to-site
static	1	N
dynamic	2 + N	3N

Conceivably the best choice was to move p and not s. In that case the estimate of the result size of the restriction on s was incorrect (possibly the data in s did not fit the estimation model.) Potentially, the savings resulting from moving p instead of s were greater than the overhead of using dynamic decision making. This strongly depended on whether messages on the actual communication network were a significant cost. The choice between dynamic and static decision making is strongly influenced by the tremendous additional network overhead that results from using dynamic decision making. Ultimately one must consider the cost to send messages and the expected accuracy of the estimation procedure.

4.8. Comparison to Other Proposed Algorithms

This section compares the tactics proposed in this thesis to tactics proposed by others. Wong has suggested an algorithm which is used by SDD-1 [ROTH77a]. The algorithm is described in [WONG77]. Hevner and Yao have proposed an algorithm which is based on Wong's work [HEVN78b].

The comparison will proceed as follows. Both Wong's and Hevner's algorithms will be described. Since Hevner's algorithm encompasses the tactics suggested by Wong, only Hevner's algorithm will be compared to the tactics discussed in this thesis. Finally the salient differences

between the two approaches will be discussed.

Wong's algorithm [WONG77] is based on the site-to-site network model and does not consider a broadcast model at all. The sole optimization criterion is minimizing the number of bytes transmitted. Only retrieval is considered. Moreover, it is assumed that relations do not span more than one site. It is not clear how the algorithm will be extended to allow multiple fragments of a relation on multiple sites.

Wong's algorithm begins by performing all possible local processing. This includes all one variable restrictions, and all other local processing including joins between two relations which are completely contained at the same site.

Next an initial sequence of moves is determined that will move all remaining fragments to one site for processing. The site chosen is the one with the most data. After processing, the results are moved to the final destination site. Wong represents the sequence of moves by nodes in a tree. If the query involves less than three relations, then the optimization is complete. If there are three or more relations then a "node splitting" tactic is used to break the data movement into pieces. The node splitting tactic is applied recursively until no further splitting is possible.

If splitting a node into two pieces results in an immediate reduction in network communication costs then the split is always included in the processing strategy. Once a node is split, it is never recombined.

When moving relations, fragments are not combined. If a join is performed the join-restrict technique discussed in sections 3.5 and 4.5 is employed. When a join is executed, all relations are restricted by the join and later transmitted as separate relations. When all relations reach the final processing site, they are finally joined into the one result relation.

Hevner's algorithm is based on a network model similar to Wong's. A site-to-site model is the only one considered. Furthermore, it is assumed that parallel transmissions are possible, that is, any set of sites can concurrently communicate with each other. This includes one site sending to two different sites and one site receiving from two sending sites. Two different optimizations are considered: (1) minimum transmission cost and (2) minimum network delay. Transmission cost is assumed to be a linear function of the number of bytes transmitted. The two criteria differ if two or more transmissions occur in parallel. The minimum delay will always be less than or equal to the minimum transmission cost. The algorithm ignores any local processing costs and it assumes

that relations cannot span more than one site.

The algorithm begins by doing all the local processing that can be done - exactly like Wong's algorithm. Next an initial strategy (called a schedule by Hevner & Yao) is set up to move all relations to the site where the result is required, again exactly as in Wong's algorithm. An "improved exhaustive search" looks for a strategy better than the initial strategy. It differs from Wong's node splitting tactic in that more combinations of moves are considered. This is a fundamental difference between Wong's and Hevner's algorithms. For computational efficiency, the search space is limited by first ordering the relations from smallest to largest (measured by their size in bytes). Then the search begins by considering moving the smallest relation to one or more of the other relations. The authors claim that by ordering the relations according to size and then examining the possible moves,

> "data transmissions are quickly eliminated from consideration if they cannot possibly be part of a minimum time solution." [HEVN78b]

When minimizing total network traffic, Hevner examines more alternatives than Wong. As a result, Hevner will find an equal or better solution than Wong. This is because Wong's algorithm looks only at splitting a sequence of moves in two and once the decision to split is made, it is never rescinded. Wong's node splitting

algorithm is a "greedy" algorithm that optimizes for the current processing step without regard to "global" optimization. Hevner claims [HEVN78b] that he will find solutions that Wong will not find when optimizing for minimum network delay. This claim is reasonable since his model of a site-to-site network allows parallel transmissions and Wong chooses to minimize only total transmission cost.

In summary, it is sufficient to compare our algorithm to Hevner's since it encompasses Wong's algorithm.

4.8.1. Comparison to Algorithm G

We will now compare the tactics presented in this thesis with those of Hevner & Yao's "Algorithm G" [HEVN78b]. First, the processing and network models the two proposal use are compared. It is shown that Algorithm G considers only a subset of the issues that this thesis considers. Next portions of the two sets of algorithms are compared assuming the site-to-site model and assuming one relation per site.

The network models of algorithm G and this thesis differ greatly. Algorithm G considers only a site-to-site model. Thus we cannot compare special tactics and analyses which are used on broadcast networks. The site-to-site models use the same cost formula but Hevner assumes that sites can communicate in parallel. The cost to com-

municate, measured in bytes, will be identical in both algorithms but the time required to communicate will be different. To the extent that parallel communication exists in the actual network and can be taken advantage of, this is an important distinction.

Algorithm G assumes that local processing costs are insignificant compared to network communication costs. It makes no attempt to reduce the demands on local processing. This is most apparent in the use of the join-restrict technique. Explicitly integrated into algorithm G is the fact that the join-restrict technique will always be employed and that two different relations will never be joined for transmission. Recomputing the join at the destination requires a substantial amount of local processing. As shown in section 4.5, the join-restrict technique is not always optimal for minimizing communication costs. This thesis treats the join-restrict technique independently from the other processing tactics and uses it only when it is optimal.

Algorithm G does not consider relations spanning more than one site. It is not at all clear how to extend the cost formula to include multiple site relations. It was shown in section 4.2 that for relations spanning multiple sites, the choice of processing sites, K, is critical for minimizing the communication cost and for maximizing

parallelism. Those trade-offs cannot be compared since Algorithm G does not consider fragmented relations.

Since Algorithm G uses exhaustive search for determining its strategies, it can find better solutions than the query splitting technique using limited search of this thesis but it will find the same solutions as the query splitting technique with exhaustive search presented in section 4.3.

An example will be used to illustrate the important differences between the two algorithms. For comparison we must restrict ourselves to one relation per site. For convenience, we will name the sites with the same name as the variable which resides on that site. Consider the query:

```
retrieve (p.pname, s.sname, j.jname)
        where
                p.pnum = s.pnum
        and
                s.snum = j.snum
```

Assume that minimum network delay is the sole objective. Depending on the transmission cost estimates, Hevner's algorithm might do the following:

(1) Simultaneously move S to J and S to P

(2) Run two separate queries on sites P and J. On site P:

```
retrieve into p' and s'(p.pname, p.pnum,
                s.snum,s.sname,s.pnum)
        where p.pnum = s.pnum
```

and site J:

```
retrieve into j'(j.snum, j.jname)
        where j.snum = s.snum
```

(3) Move J' to P' and run the final query

```
retrieve (p'.pname, s'.sname, j'.jname)
        where
                p'.pnum = s'.pnum
        and
                s'.snum = j'.snum
```

The query splitting tactic would:

(1) Move S to J

(2) Execute the query:

```
retrieve into J'(j.jname, s.sname, s.pnum)
        where j.snum = s.snum
```

(3) Move J' to P and run the final query:

```
retrieve (p.pname, j'.sname, j'.jname)
        where
                p'.pnum = j'.pnum
```

The difference between the two strategies is that in step 1, Algorithm G moves S to two different sites in parallel. Since this is done in parallel, the delay time is the same as moving S to only one site. Then in step 3, J' is moved to the site where P is located. Using our technique, J' would be necessarily larger since it must include part of the relation S. For algorithm G, S was moved in Step 1. Algorithm G would have a shortened

network delay and would be faster by the time required to transmit the difference between the sizes of the two J's in step 3.

If minimizing total communications was the optimization criterion, then our technique would remain the same but Algorithm G would:

(1) Move S to J.

(2) Execute the query:

```
retrieve into J' and S'(j.jname,j.snum,
                        s.sname,s.snum,s.pnum)
        where s.snum = j.snum
```

(3) Move J' and S' to P

Both algorithms would perform the same steps except for deciding if J' is a composite of J and S or J' and S' are transmitted separately. Our algorithm would determine the most beneficial way to encode and transmit the join. Hevner's algorithm would always transmit J' and S' as two separate fragments and recompute the join. Thus depending on the specific cost estimates of the query, both algorithms would perform the same or the algorithms from this thesis would do better if chooses a more efficient encoding of J'.

This example illustrates that the primary difference between the two techniques is the decision to trade extra network communications traffic in exchange for less

network delay. This would always be cost effective if local processing time were insignificant compared with communications time, and if parallel transmissions were possible on the communication's network.

Distributed processing algorithms are also strongly influenced by one's model of a typical data base design. Does one views relations as residing completely at one site or spanning many sites? Hevner's view leans strongly towards one relation residing completely at one site while our view is one of relations fragmented across most sites. If one assumes complete relations on a site, it becomes very important to look for subqueries that can be processed in parallel; otherwise, computer sites would sit idle. If one assumes relations are fragmented across most sites, then most sites will participate as processing sites in all queries.

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH

5.1. Conclusions

This thesis has presented a general framework for processing queries on distributed, relational data bases. The distributed environment was modeled on two different communication networks, broadcast (modeled after the ETHERNET), and site-to-site (modeled after the ARPANET). Each of the tactics presented in this thesis were analyzed for both network models.

A relational model was presented which allowed relations to be fragmented across any number of processing sites and a distribution criterion was used to distinguish which site a tuple belongs to.

Both communication network delay and local processing delay were considered for optimization criterion. If minimization of response time is the primary goal, then it was shown that both network delay and local processing delay must be taken into account.

The Fragmented Processing (FP) technique can be used to process any query on a distributed data base. It can be used to minimize network communications and to maximize parallelism in processing. The increased parallelism is

important in reducing the time delay for local processing. It was shown that a high degree of parallelism is possible in particular on a broadcast network.

The query splitting tactic can be used to reduce network communications on queries involving three or more relations. It can use either a limited or an exhaustive search for deciding how to break a query apart. A simulation program was used to compare limited and exhaustive search. It was shown that exhaustive search does significantly better than limited search in nearly all cases. The simulation program was also used to compare static and dynamic decision making. It was shown that dynamic decision making performs noticeably better than static decision making. The simulation was further used to compare perfect information with a simple estimation procedure. The results indicated that there are many cases where a simple estimation can perform very well.

Several join encoding tactics were presented. The join encodings provide a way to trade decreased communication costs for increased local processing costs. The method used to encode a join for transmission is divorced from any of the other tactical decisions.

Two shuffling strategies were analyzed. The primary trade-offs were shown to be the bandwidth of the network and the cost to send a message.

It was shown that the transmission of a relation fragment's physical structure is rarely cost effective. This greatly simplifies the bookkeeping required in distributed systems. Furthermore, it simplifies estimating the cost to move a relation fragment since no decision has to be made concerning whether to move the relation's physical structure(s).

Finally, the relationship of query size estimation, dynamic decision making, and movement of the result relation were analyzed with respect to the distributed environment. On a network where the cost to send a message is very small, dynamic decision making should be used. On a network where the message cost is high, static decision making should be used and the accuracy of size estimation is much more important.

5.2. Future Research

There are numerous future research areas in the field of distributed data bases. We will outline a few extensions of this thesis which merit further research.

To begin with, there are cases for distribution criteria which specify multiple relations. As an example consider two relations with identical distribution criteria. If the joining domains occured in the distribution criterion, then a very intelligent theorem prover might be

able to discover that the join can be completely solved locally on each site. The job of the theorem prover could be simplified by allowing multiple relations to be included in one distribution criterion. The qualification of such a distribution criterion would apply to all the identified relations.

There is a close relationship between the function of the distribution criterion and access method structures. The distribution criterion serves as a coarse, high level index for the relation. Further research might develop a unified concept of physical structure and distribution criterion. Such a generalization could allow multiple fragments of a relation on one site. In the same way distribution criteria restrict the number of sites involved in a query, they can choose among several fragments of a relation on one site. This could help limit the search time and also improve concurrency control if primary fragment locking [RIES78] is being used.

The network model could be expanded to include a heterogeneous mixture of broadcast and site-to-site models. For example, such a situation makes sense in an environment where all the sites in one building are connected with a broadcast network and all the buildings are connected in a site-to-site network.

Finally, a great deal could be learned by implementing a distributed system. A preliminary system has been implemented at U.C. Berkeley and is being extended. It would be invaluable to measure the true performance benefits realized by parallel query processing.

REFERENCES

[BAYE70] Bayer, R. & McCreight, E., "Organization and Maintenance of Large Ordered Indices", Proc. 1970 ACM-SIGFIDET Workshop on Data Description, Access, and Control, Houston, Texas, Nov. 1970.

[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.

[CHAM76] Chamberlin, D.D.; "Relational Data Base Management: A Survey," Computing Survey, Vol. 8, no. 1, March 1976.

[CHU69] Chu, W.W.; "Optimal File Allocation in a Multiple Computer System", IEEE Transactions on Computers, vol. C-18, no. 10 October 1969.

[CHU76] Chu, Wesley W.; "Performance of File Directory Systems for Data Bases in Star and Distributed Networks," AFIPS Conference Proceedings, vol.

45, 1976.

[CODD70] Codd, E.F.; "A Relational Model of Data for Large Shared Data Banks," CACM vol. 13, no. 6, June 1970.

[DALA78] Dalal, Y.K. & Metcalfe, R.M., "Reverse Path Forwarding of Broadcast Packets", Communications of the ACM, Volume 21, Number 12, December, 1978.

[DATE77] Date, C.J., "An Introduction to Database Systems", Second Edition, Addison-Wesley, 1977.

[DUHN78] Duhne, R.A. & Severance, D.G., "Selection of an Efficient Combination of Data Files for a Multiuser Database", Proceedings AFIPS National Computer Conference, 1978.

[ELLI77] Ellis, C.A.; "A Robust Algorithm for Updating Duplicate Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[EPST78] Epstein, R.; Stonebraker, M.; Wong, E; "Distributed Query Processing in a Relational Data Base System", SIGMOD Conference Proceedings, May, 1978.

[ESWA76] Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, L.I.; "On the Notions of Consistency and Predicate Locks in a Database System", CACM vol. 19, no. 11, November, 1976.

[GRIF79] Griffiths Selinger, P. et. all., "Access Path Selection in a Relational Data Base Management System", IBM Research Laboratory, San Jose, California, RJ2429(33240), January, 1979.

[HAMM76] Hammer, M. & Chan, A., "Index Selection in a Self-Adaptive Data Base Management System", Proc. ACM SIGMOD, 1976.

[HAWT79] Hawthorn, P. B., "Evaluation and Enhancement of the Performance of Relational Database Management Systems", Ph.D. Disertation, University of California, Berkeley, California, 1979.

[HELD75a] Held, G.D., M.R. Stonebraker, and E. Wong; "INGRES - A Relational Data Base System," Proc. NCC vol. 44, 1975.

[HELD75b] Held, G.D., "Storage Structures for Relational Data Base Management Systems", Ph.D. Dissertation, University of California, Berkeley, ERL

Memo No. ERL-M533, 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.

[IBM66] IBM Corporation, "OS ISAM Logic", IBM, White Plains, N.Y., 1966, GY28-6618.

[KERN78] Kernighan, B.W. & Ritchie, D.M., "The C Programming Language", Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1978.

[KNUT73] Knuth, D., "The Art Of Computer Programming", Vol. 3, Addison-Wesley, Reading, Mass., 1973.

[LAMP76] Lamport, L.; "Time, Clocks and Ordering of Events in a Distributed System," Mass. Computer Associates Report CA-7603-2911, March 1976.

[LBL76] Proceedings of the First Berkeley Workshop on Distributed Data Management and Computer Networks, May 1976, Berkeley, California.

[LBL77] Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, Berkeley, California.

[LBL78] Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, August, 1978, San Francisco, California.

[LEVI75] Levin, K.D. & Morgan, H.L., "Optimizing Distributed Databases - A Framework for Research", Proc. NCC vol. 44, 1975.

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

[RIES78] Ries, D.; Epstein, R,. "Evaluation of Distribution Criteria for Distributed Data Base Systems", University of California, Electronics Research Laboratory, UCB/ERL M78/22, May 12, 1978.

[RIES79] Ries, D.R., "The Effects of Concurrency Control on Data Base Management System Performance", Ph.D. Disertation, University of California, Electronics Research Laboratory, UCB/ERL M79/20, April, 1979.

[ROBE70] Roberts, L. and Wessler, B., "Computer Network Development to Achieve Resource Sharing," Proc. SJCC, 1970, AFIPS Press.

[ROSE77] Rosenkrantz, D.J., Sterns, R.E., Lewis, P.M.; "A system Level Concurrency Control for Distributed Database Systems", 1977 Berkeley Workshop on Distributed Data Management and Commputer Networks, Lawrence Berkeley Laboratory, May 1977.

[ROTH77a] Rothnie, J.B. and N. Goodman; "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[ROTH77b] Rothnie, J.B. and N. Goodman; "A Survey of Research and Development in Distributed Database Management", Proceedings Very Large Data Bases, October, 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. (To appear).

[STON75] Stonebraker, M.R.; "Implementation of Integrity Constraints and Views by Query Modification", University of California, Electronics Research Laboratory, Memorandum ERL-M514, March 1975.

[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.

[STON77] Stonebraker, M.R. and E. Neuhold; "A Distributed Database Version of INGRES," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[STON78] Stonebraker, M.R.; "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", University of California, Electronics Research Laboratory,

[STON79] Stonebraker, M.R.; "MUFFIN: A Distributed Database Machine", University of California, Electronics Research Laboratory, UCB/ERL M79/28, May, 1979.

[THOM75] Thomas, R.H.; "A Solution to the Update Problem for Multiple Copy Databases Which Use Distributed Control," BBN Report 3340, Bolt Beranek and Newman Inc., Cambridge, Mass., July 1975.

[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.

[YAO77] Yao, S.B., "An Attribute Based Model for Database Access Cost Analysis", ACM Transactions on Database Systems, Vol 2, No. 1, March 1977.

[YOUS78a] 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.

[YOUS78b] Youssefi, K.; "Query Processing for a Relational Database System," Ph.D Dissertation, University of California, Berkeley, 1978, Electronics Research Laboratory, Memorandum UCB/ERL M78/3, January 6, 1978.