THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

A QUERY OPTIMIZATION IN DISTRIBUTED DATABASE SYSTEMS

C-W. Chung

CRL-TR-4-83

Under the Direction of
Professor Keki B. Irani

FEBRUARY 1983

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

---

[1]This research was supported by the Department of the Army, Ballistic Missile Defense Advanced Technology Center, Rome Air Development Center, and the Defense Mapping Agency under contract F30602-80-C-0173, and by the Air Force Office of Scientific Research/AFSC, United States Air Force under AFOSR contract F49620-82-C-0089. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agency.

engn

UMR1194

ABSTRACT

A QUERY OPTIMIZATION
IN DISTRIBUTED DATABASE SYSTEMS

By
Chin-Wan Chung

Chairman: Professor Keki B. Irani

This research is concerned with a model and a method of minimizing the inter-site data traffic incurred by a query in distributed relational database systems. In order to process a query which references data from multiple sites in a computer network, portions of the database at other sites have to be transferred to the user's site. The usual methodology for distributed query processing consists of reducing the referenced relations using a sequence of semijoin operations after initial local processing.

The mathematical model has been developed to determine an optimal sequence of semijoins which minimizes the total inter-site data flow in processing a distributed query. The core of this model is a method which efficiently and accurately estimates the size of an intermediate result of a query. In particular, the assumption that joining attributes are independent during the processing of a query by a sequence of semijoins has been relaxed.

Since the distributed query optimization problem is known to be NP-hard, a heuristic algorithm has been developed to determine a low-cost sequence of semijoins. The efficiency of the algorithm is increased by partitioning the set of joining attributes into blocks and sequencing these blocks, as well as by a straightforward, yet effective sequencing among the semijoins between the joining attributes inside a block. The algorithm decreases the cost of a query by selecting the low-cost, highly reductive semijoins first. Cost comparisons with the existing algorithms have been provided. The time complexity of the main features of the algorithm has been analytically derived.

The algorithm has been implemented in PASCAL. The tests show that the scheduling time for a sequence of semijoins for a reasonable size query is less than 0.05 seconds when the program is executed by a main-frame computer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

Appendix

NOTATIONS

A(B): The set of active attributes in block B.

A_i: The initial set of values of a_i after initial local processing.

a_i: The ith attribute in the user query.

AJA = $\{AJA_i \mid A_i \in J\}$.

AJA_j: The set of all attributes associated with a_j.

A_{n+m}: The reduced set for the mth expanded lattice.

$A\beta(a_i)$: The set of associate blocks of a_i.

BC(k): The block cost of unvisited block B_k.

b_i: The benefit achieved by ϕ_i.

b_{ij}: The benefit achieved by f_{ij}.

B_k: The kth block in Π.

$\bar{c}$: Σ X Φ --> REAL: The cost reduction function.

$C_A = \{|K_i| \mid a_i \in J\}$.

$C_D = \{|D_i| \mid B_i \in \Pi\}$.

c_i: The cost incurred by ϕ_i.

c_{ij}: The cost incurred by f_{ij}.

C(M): The transmission cost to transfer the message of length M.

${}_nC_i (= C_i^n)$: The number of combinations of n objects taken i at a time.

$C_R = \{|R_i| \mid R_i \in R^+\}$.

C^X: The set of elements in L^I covered by X.

DB = <INFO, PAR>: The state of the database.

D_i: The domain of the attributes in block B_i.

d_i: The density of a_i.

$dim(L^I)$: The dimension of the initial lattice L^I.

DIST = $<ST_R, ST_U>$: The data distribution information.

d_N: The depth function of L^N.

d[x]: The depth of an element x in a lattice.

$d_Z[X]$: The relative depth of an element $X \in L^Z$.

EJA = $\{EJA_i \mid A_i \in J\}$.

EJA_i: The set of all attributes equivalent to a_i excluding a_i.

END_Aβ: The set of end associate blocks.

E[X]: The expected value of random variable X.

E^X: The expanded sublattice with the greatest element X.

f_{ij}: The semijoin from a_i to a_j, where a_i and a_j are the joining attributes in the same block.

G^Z: The set of generators of L^Z.

IC: The initial cost of a query.

INFO = $<N_R, N_B, JR, SJR, EJA, AJA, \Phi>$.

I_X: The index set of a reachable set X.

J: The set of joining attributes of the user query.

JR: The set of all joining relations.

K_i: The current set of values of a_i.

L: (i) The expanded lattice for block B.

(ii) A lattice.

L^I: The initial lattice for block B.

l(L): The length of a lattice L.

L^N: The new expanded lattice after the generation of E^X for $X \in L$.

L^Z: The L^I-type lattice with the greatest element Z.

M: The length of a message.

$N_B = \{|B_i| \mid B_i \in \Pi\}$.

n_i: The net benefit of ϕ_i.

n_{ij}: The net benefit of f_{ij}.

$N_R = \{|\alpha_i| \mid R_i \in R^+\}$.

O^Z: The least element of L^Z.

$PAR = <C_R, C_D, C_A, W_A>$.

$P_c(\pi, C)$: The cost function, of path π and candidate block B_C, used in selecting the next block to be visited.

$P_n(\pi, C)$: The cost function, of path π leading to candidate block B_C, used in determining a path.

$Q = <T, J, R^+, \Pi, \mu, \psi>$.

R^+: The set of relations referenced by the user query.

R_A: The answer relation of a user query.

REAL: The set of real numbers.

R_i: The ith relation in R^+.

RS_k: The current set of all reachable sets for B_k.

RS_k^I: (i) The initial set of all reachable sets for block B_k.

(ii) The initial reachable sets in level k of

the lattice for block B.

RS^X: The poset corresponding to E^X.

S = <INFO, PAR, DIST>: The state of the distributed database.

$\bar{s}$: Σ X Φ --> Σ: The state transition function.

S_i: The state of distributed database before ϕ_i.

SI(B): The sequence of inactive attributes in B.

S_{i+1}: The state of distributed database after ϕ_i.

SITE: The partition of R^+ such that any two relations in the same block are stored at the same site.

SJR: The set of all singleton joining relations.

S(R): The size of a relation R.

ST_i: The ith block in SITE.

$ST_R = \{ST_R^i \mid R_i \in R^+\}$.

ST_R^i: $ST_j \in$ SITE such that $R_i \in ST_j$.

ST_U: The set of relations stored at the user site.

SVβ: The sequence of visited blocks.

T: The target list of the user query.

T_{ij}: The restricting set of K_i and K_j.

Uβ: The set of unvisited blocks.

V_f: The fixed portion of the message overhead.

v_N: The level function of L^N.

V_p: The portion of the message overhead which is proportional to the length of the message.

v[x]: The level of an elements x in a lattice.

Vβ: The set of visited blocks.

$W_A = \{w_i \mid a_i \in T \cup J\}$.

w_i: The width (in bytes) of attribute $a_i \in T \cup J$.

$|X|$: The cardinality of set X.

α_i: The set of component attributes of relation R_i.

β: The set of all blocks.

μ: $(T \cup J) \dashrightarrow R^+$ is a membership function.

Π: The partition of J induced by the equivalence relation '='.

π: The path consisting of a sequence of semijoins.

π_{ci}: The path for candidate block B_{ci}.

Σ: The state space of the distributed database.

σ: The sequence of semijoins being scheduled.

$\upsilon = 1 + V_p/M$: The proportional coefficient.

Φ: The set of all possible semijoins to process a given user query.

$\emptyset$: An empty set.

$\emptyset_i$: The ith semijoin in a sequence of semijoins.

ψ: $J \dashrightarrow \Pi$ is a partition function.

Ω: The set intersection.

$X \wedge Y$: The greatest lower bound of X and Y.

$X \vee Y$: The least upper bound of X and Y.

CHAPTER 1

INTRODUCTION

1.1 Background

In this section, we will briefly review the concept and the development of the area in which our research problem is embedded.

The value and size of data in organizations have been continuously increasing in recent years. In order to provide an efficient and flexible use of data while maintaining its consistency and security, the technology of database management has been rapidly developed.

A database is a collection of stored interrelated data used by the numerous application programs of any organization. In traditional file systems, each application program has its own private data files [NOLA 73]. This results in redundant storage of data, generally in different formats. The uncontrolled separate update of redundant data by each application program leads to the severe problem of data inconsistency. Also because of the lack of unity of formats, it is difficult to develop new application programs to operate against many existing data files. The purpose of the database system is to overcome the drawbacks of the file system through the integration of the organization's data so

that the stored data can be shared. An integrated database provides the organization with centralized control of its operational data. There are many advantages of having centralized control of data [DATE 77, MART 77]. A database management system is a set of generalized system software which manipulates databases and provides interfaces with a broad range of languages to aid all users. An information processing system which uses a database management system is referred to as a database system.

Database systems are mainly classified into two categories: centralized database systems and distributed database systems. In a centralized database system, the whole database is stored at one computer site at which a database management system also resides. In a distributed database system, the database is scattered among the computer sites, each of which is equipped with a local database management system and supporting modules to interface with other local database management systems. A distributed database system is implemented on a computer network, that is, a set of computer sites which communicate with one another via a communication network consisting of the switching computers and the communication channels.

The main characteristics of the distributed database system is that it acts conceptually as a centralized system in terms of user view and system control, while physically permitting the geographic distribution of an organization's data and accesses to them. For this reason, the distributed

database system is a suitable solution to the information processing problems of geographically dispersed organizations such as the military, government affiliated organizations and large corporations.

The major advantages of the distributed database system over the centralized database system, in terms of applications requiring access to an integrated database from geographically dispersed locations, can be stated as follows:

1) A portion of the database is stored at or near the sites where it is frequently accessed. Consequently, the communication cost and delay are reduced.
2) Since the distributed database system is implemented on a computer network including multiple computer sites, the breakdown of some computers or a part of the communication network does not cause the total failure of the system. Therefore, the distributed database system is more reliable.
3) Evolution of one subsystem is possible without disturbing the rest of the system. Hence the expansion of the system is easier.
4) For transactions referencing data from multiple sites, intermediate processing can be performed at each site at the same time. This indicates increased parallel processing and load distribution among sites.

There have been a number of special purpose distributed database system implementations reported in the literature [CHAM 77]. However, there are only a few prototype systems [ROTH 80, STON 77] which can be classified as general purpose systems. No general purpose distributed database system is available because an assortment of difficult technical problems must be solved before a workable system can be produced. These problems include:

1. Database distribution,
2. Distributed query processing,
3. Distributed concurrency control,
4. Continuity of operation in the presence of failures,
5. Directory management.

Research in distributed database systems has been continuing since the late 1970's. Many new results have been found. Nevertheless, none of the above problems has been completely solved.

1.2 The Problem and the Approach

In this section, we will state the research problem and discuss the major issues of our approach. We will present them first in general terms and then in more specific terms.

In this investigation we are concerned with developing a model and arriving at a methodology for deriving an optimal strategy for processing a distributed query. A query is a user transaction to retrieve information from the database. The user transactions in database systems are

queries and updates. An update is always preceded by a query to locate the necessary part to be updated. A query is called distributed if data from multiple sites is necessary to answer the query. The locations of necessary data are known to the distributed query processing routine beforehand. In order to process a distributed query, the portions of the database referenced by the query at each site have to be transferred to the user site where the final processing is performed. Distributed query processing is fundamentally different from centralized query processing in two respects:

1) The delay caused by the transfer of data among the sites involved in the query is substantial.
2) Local processing can be performed simultaneously by the computers at the sites involved in the query.

The type of communication network assumed in this research is the point-to-point packet switching network which uses ground lines as communication channels. In a packet switching network, the delay due to transmission of a fair amount of data between source and destination is proportional to the volume of the data [KLEI 76]. It was observed by [WONG 77] for the Arpanet that the data transfer rate between sites is some 100 times slower than the transfer rate between disk and main memory in typical large-scale computers. Consequently, the minimization of the inter-site data transfer is of primary importance in processing the distributed query. The efficiency of a

strategy for distributed query processing is mainly determined by the way the inter-site data transfer is handled. As illustrated in [ROTH 77], there is enormous variation in communication delay among a set of plausible distributed query processing schemes.

A relational model [CODD 70] is assumed to be the underlying data model. A relational model represents data logically and can be used as a conceptual framework for other data models. Some basic concepts of the relational model are presented in Appendix A. In order to describe the problem more precisely, the terminology of the relational model will be used. We assume that a relation is a unit of distribution. A relation may be further partitioned into smaller units by performing selection and/or projection on it to increase flexibility. The original relation can be recovered by joining the smaller relations. This extension to relations partitioned among many sites is straightforward [DAYA 79]. We will only consider the conjunctive queries. For a non-conjunctive query, the qualification clause can be transformed into a disjunctive normal form and the query can be decomposed into conjunctive queries. The answer to the complete query then is the union of the partial answers obtained for each conjunctive query.

Since selection and projection are the unary operations, they are always performed locally. Join can also be performed locally if the relations involved in the join are stored at the same site. Hence a query which does

not require a join of the relations stored at different sites can be decomposed into local queries. Such a query need not be considered a distributed query even if the query references relations from multiple sites.

Definition 1.1: A query is distributed if its qualification clause contains at least one join term which includes relations stored at different sites.

For any distributed query, initial local processing is mandatory to reduce the amount of data to be transferred locally. Initial local processings are performed in parallel at the sites containing the relations referenced by the query. Large portions of referenced relations are reduced by initial local processing for most of the distributed queries.

A crude way of processing a distributed query may be to transfer all the remaining portions of relations after initial local processing to the user site. Another possibility is to transfer relations involved in joins to other sites to enable local joins. Among many alternatives, the data reduction strategy we will attempt to model is based on the properties of the semijoin.

Suppose a distributed query contains a join term R.A = S.B where R and S are located at different sites. The semijoin (Appendix A) has the following properties [BERN 79]:

(1) $R[A=B]S = (R<A=B]S)[A=B]S$

(2) $R<A=B]S \subseteq R$

(3) R<A=B]S = R<A=B](S[B])

From (1), the join term can be processed with R<A=B]S instead of R. From (2), R<A=B]S is smaller in size than R. From (3), to perform R<A=B]S at the site of R, only S[B] is needed to be transferred instead of S itself from the site of S to the site of R. Hence if ||R - R<A=B]S|| > ||S[B]||, where ||R|| denotes the size of a relation R, then the transfer of S[B] contributes to the reduction of the total amount of data transfer without affecting the final query answer. After R is reduced to R<A=B]S, (R<A=B]S)[A] can also be sent to reduce S. Basically the distributed query processing strategy after initial local processing is a sequence of semijoins which consists of a set of inter-site data moves combined with local processings between the data moves. When the relations at each site cannot be further reduced, the remaining relations are transferred to the user site, where the final processing of the query is performed.

We assume that the delay caused by the local processing between inter-site data moves is constant compared with the great variation of the communication delay. This is especially valid when the data manipulation operations are performed by database machines because the processing time of operations is less dependent on the size of the operands and much shorter than when general purpose computers are used. Also the fact that the computing cost is reducing at a faster rate than the communication cost indicates that the communication delay will become the more dominant term in

the future.

In summary, our approach is to find a sequence of semijoins which minimizes the total amount of inter-site data communication necessary in transferring the data to the user site for the final processing.

In addition to the operational importance to the distributed database system, distributed query processing is crucial for the optimization of database distribution which is known as the problem of file allocation. The file allocation problem has to assume a certain distributed query processing method to formulate the communication cost in terms of file allocation variables. If the distributed query processing transfers all the remaining portions of relations, after initial local processing, to the user site, the formulation is straightforward. However, the resultant file allocation may be far from optimal. If the distributed query processing uses semijoins, the effect of semijoins on the communication cost has to be considered for an optimal allocation. Even if the data flow generated by semijoins for each pair of relations before the allocation are given as input, the contribution of semijoins to the communication cost cannot be formulated using only file allocation variables. One of the reasons is that when two relations are clustered at the same site, not only the semijoin between them vanishes but also the data flow generated by semijoins between each of these two relations and the rest of the relations change in a complex way. Hence the choice

of a good distributed query processing strategy and the successful modeling of its features are essential for realistic optimal database distribution.

1.3 Literature Survey

The purpose of this section is to review the published results of related work by others and to analyze their inadequacies.

The primitive concept of distributed query processing appeared in the context of file allocation problems. As mentioned before, every file allocation model has to use some type of distributed query processing method. In traditional file allocation models [CASE 72, CASE 73, CHU 69, IRAN 82, LEVI 75, MAHM 76], the derivation of the communication cost is based on the amount of data flow from each site to each file. This implies either that there is no distributed query, or that distributed queries are processed at the point where the query originates with all the necessary data moved to that point possibly after some degree of local processing. This type of distributed query processing incurs large volumes of unnecessary inter-site data traffic which can be screened out by using semijoins or local joins.

Among the file allocation models, [RAMA 79] introduced a somewhat different distributed query processing method to reduce inter-site data transfer. A technique was proposed to add redundant information onto the distributed database

so that distributed queries can be decomposed into local queries. The redundant information is an extra bit for each tuple indicating whether the tuple will participate in processing some specific join term. Hence if the relations in the join term are located at different sites, the necessary tuples in each relation can be selected by checking the extra bits without moving one relation to the site of the other. This approach often requires a great deal of system overhead to maintain redundant information in accordance with the update of the database. From a distributed query processing point of view, the impact of initial local processing in reducing the amount of inter-site data traffic is not properly reflected. The use of semijoins can further decrease the amount of data transfer.

The earliest work reported in the area of distributed query processing as an independent subject was in [WONG 77]. Since then, numerous strategies have been developed for distributed query processing. We will review some of the more important ones.

Among the distributed query processing algorithms reported in the literature, [WONG 77] was the first one to consider communication delay as a key element of query processing cost. Communication delay depends on the quantity of data transferred between sites. Thus the minimization of data transferred is the primary objective in optimization though not the only one. The concept of query decomposition in [WONG 76] is adapted to reduce a

distributed query to a series of local queries. A basic tactic proposed in [WONG 76], to decompose a query which references many relations into a set of queries referencing only one relation, is tuple substitution. Tuple substitution is a procedure by which one of the relations in a join term is successively replaced by the actual tuples. Tuple substitution for a distributed query implies the transfer of data, one tuple at a time, between sites. However, bulk transfer of data is more efficient than tuple-at-a-time transfer because of the overhead per message. Hence a distributed query is transformed into local queries by moving subrelations. The initial solution for this strategy is to transfer all the remaining relations referenced by the query to a single site after the initial local processing. An improved solution is a set of cost-effective subrelation moves among sites followed by local processings. The optimization procedure is applied recursively until no improvement is gained. Since the algorithm looks for an immediate improvement, it is classified as a greedy one and terminates at a local optimum. It is necessary to estimate the sizes of the relations resulting from local operations in order to determine the cost-effectiveness of subrelation moves. This problem was left unsolved. Moreover, the move of a subrelation is less effective than the semijoin in reducing the inter-site data transfer because the semijoin only requires transmission of values of the joining attributes.

The approach in [EPST 78] is basically the same as the one in [WONG 77]. The following modifications were made on the framework established in [WONG 77]: (1) each relation may be at a unique site or may be spread over several sites in a computer network; (2) the cost criteria considered are minimum response time and minimum communication traffic; (3) the algorithm treats point-to-point and broadcast networks separately. Some of the rules to determine the values of the variables are based on heuristics rather than well-grounded analysis.

The basis of the model developed in [CHU 79, CHU 82] is also the reduction of inter-site data transfer by using the transmission of subrelations. A query operation graph was defined, in which a node represents a subset of the sequence of operations that must be executed at the same site and arcs represent data transmissions between sites. Given a query, the set of query operation graphs which represent the sequence of operations is constructed. To determine a query processing policy, it is necessary to select a site for performing the operations represented by each node. Theorems were developed to find the best sites for performing the operations of a given graph. A linear operating cost function was derived to compute the cost of a processing policy represented by each query operation graph. The major considerations of the proposed operating cost model are communication cost, processing cost, and data reduction functions for processing a query for a given

environment. Data reduction functions describe the volume of output data in relation to the volume of input data for performing a specific operation. The operating cost function assumes the values of many parameters as given. Data reduction functions are key elements which make the distributed query optimization problem difficult. However, they are assumed to be estimated by simulation or measurement on the actual distributed database.

[HEVN 78] developed an optimal processing algorithm for a narrow class of distributed queries called simple queries. A simple query was defined as one by which, after initial local processing, each relation that is referenced contains only one attribute - a common joining attribute, which is also the only attribute in the target list. Hence the subrelation move in this case is the move of current values of the common joining attribute for each relation. Although the class of queries considered is so narrow that the algorithm is of little practical value, this strategy introduced the concept of the semijoin.

The result of [HEVN 78] was extended for general distributed queries by the same authors [HEVN 79b]. The semijoin is used as a major tactic to reduce inter-site data transmission. The general algorithm is heuristic and uses an improved exhaustive search. Two cost measures, response time and total time, were used. The data transmission pattern containing the transmission of a relation to the result node is called the schedule for the relation. Each

joining attribute in a relation is handled separately. The schedule for a joining attribute is a sequence of semijoins. The minimal cost schedule for each joining attribute is selected from a set of cost beneficial schedules. However, the algorithm maintains other cost beneficial schedules since they may lead to more beneficial cost reductions on other attribute schedules. When the minimal time schedules are found for each joining attribute, the algorithm integrates these schedules into the overall schedule for the relation. The query processing strategy is then constructed by synchronizing the schedules of all referenced relations in the query. Schedules to minimize response time include as much parallelism of data transmission as possible. This parallelism is not taken into consideration in the schedules for the minimization of the total time. It was emphasized that the consideration of parallel transmissions to minimize the response time increases the complexity of the algorithm by a significant amount while the reduction in schedule response time is limited in almost all cases. The time complexity analysis of the algorithm was carried out only for total time minimization. The major weakness of the strategy presented here comes from the assumption that joining attributes within each relation are independent. Thus a reduction of values of a joining attribute by the semijoin does not reduce the values of other joining attributes in the same relation under this assumption. This assumption simplifies the problem in many respects. For

example, each joining attribute in a relation can be handled separately, due to attribute independence. In reality, this assumption is clearly not true. Consequently, the validity of the query processing strategy is seriously affected by this assumption. Another shortcoming is that the schedule of each relation is separately constructed and these schedules are not integrated. As a result, if an optimal strategy contains non-cost-beneficial schedules for some relation, such an optimal schedule can not be found.

An improvement over [HEVN 79b] was suggested in [CCA 80b]. The assumption of attribute independence is partially relaxed. Attributes in the same relation are considered to be dependent. However, attributes in different relations are assumed to be independent throughout the processing of a query. The objective is to process the distributed query with a minimum quantity of inter-site data transfer. That is, network bandwidth is regarded as the system bottleneck, and the optimization objective is to minimize the use of this resource. The semijoin is extensively used to reduce the inter-site data transfer. The proposed algorithm is a greedy optimization algorithm whose main function is to construct a profitable sequence of semijoins. Starting with a null sequence, the algorithm iteratively appends the cheapest profitable semijoins to the sequence until all such semijoins have been used. Then the algorithm determines the cheapest site at which to assemble the remaining relations, and appends commands to move the

remaining relations to that site. Techniques for improving the generated sequence were presented to help compensate for the short-sightedness of the greedy algorithm. While the consideration of attribute dependence is essential to the development of a realistic distributed query processing strategy, it substantially increases the complexity of the problem. The effects of the attribute dependence were not completely modeled in this work. The estimation of the reduction of relations by arbitrary semijoins is particularly important, but was not considered. Consequently, the reduction of relations due to an arbitrary sequence of semijoins can not be estimated accurately. Hence the class of sequence of semijoins allowed is very limited. The restriction of the solution space has a significant impact on the development of the algorithm. The reoccurrences of a semijoin are not allowed in the greedy algorithm. The assembly site is used because the relations are not completely reduced by all possible semijoins. The extra cost of sending the result from the assembly site to the query origin is generally substantial compared with the cost of sending the completely reduced relations directly to the query origin. The second pass enhancement is necessary to compensate for the deficiencies caused by the greedy nature of the algorithm and the use of an assembly site.

An improved version of the algorithm in [CCA 80b] was reported in [BERN 81]. Specifically, the following improvements were made: (1) the assumption that the

attributes in different relations are independent is relaxed; (2) the reoccurrences of a semijoin are allowed in the modified greedy algorithm which appends the most profitable semijoin to the sequence. The shortcomings of the algorithm in [BERN 81] are as follows: (1) the estimation method of the intermediate result size involves a complex graph search; (2) when the assembly site and the user site are different, the cost of moving the assembled answer to the user site was not considered; (3) both reported enhancements sometimes require significant computation.

Another improvement of the method presented in [HEVN 79b] was made in [CHEU 82]. A general query is decomposed into simple queries, the number of which is equal to the number of domains associated with the general query. To minimize the total time, a sequence of semijoins is scheduled by applying STRATEGY SERIAL [HEVN 79b] for each simple query. The remaining relations, except those which do not contain any target list and are involved in only one of the equijoin clauses of the given query, are transferred to the user site. This method also assumes that the joining attributes within each relation are independent.

All the distributed query processing strategies we have discussed assumed knowledge of the necessary data locations to be accessed.

Query processing experiments on a distributed database were reported in [EPST 80]. The strategies were compared by

simulation on the basis of number of bytes moved. The conclusions were: (1) limited search performs very poorly compared with exhaustive search; (2) good intermediate size estimates are crucial; (3) dynamic decision making consistently performs better than static decision making; however, because dynamic decision making has a greater run-time cost, it may not be a big winner.

CHAPTER 2

DATABASE STATE TRANSITION MODEL

In this chapter, we present a mathematical model for database state transition which allows us to estimate the change in database parameter values for any possible sequence of semijoins to process a given distributed query. This model will be used as a basis for developing a query optimization model in the next chapter.

The database state transition model includes two components described in set-theoretic terms.
These components are:

(1) Information from the user query,

(2) The effect of the semijoin on the database.

The database state transition model can be considered a function which determines the next state of the database given the current database state and a semijoin. The initial set of all possible semijoins is derived from the user query.

2.1 Query Information

In this section, the query is formally defined. From the query definition, the necessary information is derived and the parameters to describe the database are defined.

The original user query is reduced after initial local processing has been performed. During the initial local processing, all the selection terms and the join terms whose relations are stored at the same site are processed. Also the columns of the attributes which are neither in the target list nor in the remaining join terms are eliminated by projections. As a result, the reduced query contains relations and attributes which are necessary to further process the query. Some of the relations referenced by the reduced query may have been created by local joins.

Hereafter, the term "query" will represent the reduced query which necessitates distributed query processing. The attributes are differenciated whenever they are used in different relations. In other words, if an attribute A is used both in relations R and S, R.A and S.A are considered different attributes, since R.A and S.A represent different sets of values.

Definition 2.1: A query Q is an ordered six-tuple

$$Q = < T, J, R^+, \Pi, \mu, \psi >$$

where T is the target list of the user query,

J is the set of joining attributes of the user query,

R^+ is the set of relations referenced by the user query,

Π is the partition of J induced by the equivalence relation '=',

μ: (T U J) --> R^+ is a membership function,

and ψ: J --> Π is a partition function.

Since an attribute in the target list can also appear in a joining term, $T \Omega J$, where Ω denotes the set intersection, may not be empty. $T \cup J$ is a set of all attributes referenced by the query Q. The membership function μ specifies the relation to which each attribute belongs.

The partition Π is a set of blocks.

$$\Pi = \{B_1, B_2, \ldots, B_n\}$$

Here $B_k \in \Pi$ is an equivalence class under equality. Therefore for any attributes $a_i, a_j \in B_k$, $a_i = a_j$. Suppose $\mu(a_i)=R_i$ and $\mu(a_j)=R_j$ for $R_i, R_j \in R^+$. Since all the local joins were performed during initial local processing, R_i and R_j are necessarily stored at different sites. Consequently a semijoin is possible between any pair of joining attributes in $B_k \in \Pi$ for k=1, 2, ..., n. The partition function ψ indicates the block to which a joining attribute belongs. It is clear that the attributes in each block have a common domain.

Example 2.1 illustrates how to formulate a query information model.

Example 2.1

The following relations are stored in a hypothetical distributed database.

DEPARTMENT (D#, DNAME, COLLEGE)

COURSE (C#, CNAME, SUBJECT)

STUDENT (S#, SNAME, YEAR)

```
OFFER       (D#, C#, TERM)
ELECTION    (C#, S#, GRADE)
```

Consider the following user query.

"Find the names of the computer courses offered by the departments in the college of engineering during the fall term of 1980 in which a senior student received an A grade with the corresponding names of the senior students"

The user query in relational form is:

```
FIND  (COURSE.CNAME, STUDENT.SNAME)
WHERE (DEPARTMENT.D# = OFFER.D#)
 AND  (DEPARTMENT.COLLEGE = 'Engineering')
 AND  (OFFER.TERM = 'Fall 1980')
 AND  (OFFER.C# = COURSE.C#)
 AND  (COURSE.SUBJECT = 'Computer')
 AND  (COURSE.C# = ELECTION.C#)
 AND  (ELECTION.S# = STUDENT.S#)
 AND  (ELECTION.GRADE = 'A')
 AND  (STUDENT.YEAR = 'Senior')
```

Suppose the relations selected to process the above user query are all stored at different sites. The relations and the user query are reduced by the initial local processing as follows.

Reduced relations:

```
DEPARTMENT (D#)
```

COURSE (C#, CNAME)

STUDENT (S#, SNAME)

OFFER (D#, C#)

ELECTION (C#, S#)

Reduced query:

```
FIND  (COURSE.CNAME, STUDENT.SNAME)
WHERE (DEPARTMENT.D# = OFFER.D#)
 AND  (OFFER.C# = COURSE.C#)
 AND  (COURSE.C# = ELECTION.C#)
 AND  (ELECTION.S# = STUDENT.S#)
```

Now we formulate the query information model of the reduced query.

Let a_1 = DEPARTMENT.D#

a_2 = COURSE.C#

a_3 = COURSE.CNAME

a_4 = STUDENT.S#

a_5 = STUDENT.SNAME

a_6 = OFFER.D#

a_7 = OFFER.C#

a_8 = ELECTION.C#

a_9 = ELECTION.S#

R_1 = DEPARTMENT

R_2 = COURSE

R_3 = STUDENT

R_4 = OFFER

R_5 = ELECTION

Then $Q = < T, J, R^+, \Pi, \mu, \psi >$

$T = \{a_3, a_5\}$

$J = \{a_1, a_2, a_4, a_6, a_7, a_8, a_9\}$

$R^+ = \{R_1, R_2, R_3, R_4, R_5\}$

$\Pi = \{B_1, B_2, B_3\}$

$\mu = \{<a_1, R_1>,$
$<a_2, R_2>, <a_3, R_2>,$
$<a_4, R_3>, <a_5, R_3>,$
$<a_6, R_4>, <a_7, R_4>,$
$<a_8, R_5>, <a_9, R_5>\}$

$\psi = \{<a_1, B_1>, <a_6, B_1>,$
$<a_2, B_2>, <a_7, B_2>, <a_8, B_2>,$
$<a_4, B_3>, <a_9, B_3>\}$ ■

Definition 2.2: Attributes a_i and a_j are <u>equivalent</u> if $a_i, a_j \in B_k$ for some $B_k \in \Pi$. EJA_i is the set of all attributes equivalent to a_i excluding a_i.

Since $\psi(a_i)$ is the block in which a_i is included, $EJA_i = \psi(a_i) - \{a_i\}$ for all $a_i \in J$.

We define the following parameters:

(1) α_i = the set of component attributes of relation R_i

(2) $|X|$ = the cardinality of a set X

(3) D_i = the domain of the attributes in block B_i

(4) K_i = the current set of values of joining attribute a_i

(5) A_i = the initial set of values of a_i after initial local processing

(6) w_i = the width (in bytes) of attribute $a_i \in T \cup J$

(7)f_{ij} = the semijoin from a_i to a_j, where a_i and a_j are the joining attributes in the same block

Consider equivalent joining attributes a_i and a_j. Suppose $\mu(a_j)=R_j$. The semijoin f_{ij} reduces $|K_j|$. As a result, $|R_j|$ is decreased. From the attribute dependence, $|K_k|$ is reduced for all k such that $a_k \epsilon J$ and $\mu(a_k)=R_j$.

Definition 2.3: Two attributes a_j and a_k are <u>associated</u> with each other if a_j, $a_k \epsilon J$ and $\mu(a_j)=\mu(a_k)$. AJA_j is the set of all attributes associated with a_j.

It is assumed that the query being considered is not decomposable further into simpler queries. That is, for each block, there is an attribute which is associated with an attribute in another block.

The set of values of a joining attribute a_i necessary to process a query is completely included in the relation containing a_j after a semijoin f_{ij}. As a result, if the relation containing a_i does not contain any attribute in the target list, the set of values of a_i can be ignored after f_{ij} in some cases which will be explained in detail in Section 3.1. This is always true when a relation consists of only one joining attribute. We want to distinguish these relations.

Definition 2.4: A relation R_i is a <u>joining relation</u> if $a \epsilon J$ for all $a \epsilon \alpha_i$. JR is the set of all joining relations. R_i is a <u>singleton joining relation</u> if $R_i \epsilon$ JR and

$|\alpha_i|=1$. SJR is the set of all singleton joining relations.

Let Φ be the set of all possible semijoins to process a given user query. Φ is initially obtained from Q by identifying equivalent joining attributes.

$$\Phi = \{f_{ij} \mid a_i, a_j \in B_k \text{ and } B_k \in \Pi\}$$

The query information model Q determines most of the parameters which describe the portion of database necessary to process the user query. Furthermore, the initial values of some of the parameters are provided by Q. Such a set of parameters is defined separately and called INFO.

$$\text{INFO} = < N_R, N_B, \text{JR}, \text{SJR}, \text{EJA}, \text{AJA}, \Phi >$$

WHERE $N_R = \{|\alpha_i| \mid R_i \in R^+\}$,

$N_B = \{|B_i| \mid B_i \in \Pi\}$,

$\text{EJA} = \{\text{EJA}_i \mid A_i \in J\}$,

$\text{AJA} = \{\text{AJA}_i \mid A_i \in J\}$,

and JR, SJR, Φ are defined as before.

Example 2.2 shows the derivation of the initial value of INFO from Q.

Example 2.2

Consider Q formulated in Example 2.1.

The initial value of

$$\text{INFO} = < N_R, N_B, \text{JR}, \text{SJR}, \text{EJA}, \text{AJA}, \Phi >$$

is derived from Q.

(1) $N_R = \{|\alpha_i| \mid i = 1, 2, 3, 4, 5\}$

$|\alpha_1| = 1$, $|\alpha_2| = 2$, $|\alpha_3| = 2$, $|\alpha_4| = 2$, $|\alpha_5| = 2$

(2) $N_B = \{|B_i| \mid i = 1, 2, 3\}$

$|B_1| = 2$, $|B_2| = 3$, $|B_3| = 2$

(3) $JR = \{R_1, R_4, R_5\}$

(4) $SJR = \{R_1\}$

(5) $EJA = \{EJA_i \mid i = 1, 2, 4, 6, 7, 8, 9\}$

$EJA_1 = \{a_6\}$, $EJA_2 = \{a_7, a_8\}$, $EJA_4 = \{a_9\}$,

$EJA_6 = \{a_1\}$, $EJA_7 = \{a_2, a_8\}$, $EJA_8 = \{a_2, a_7\}$,

$EJA_9 = \{a_4\}$

(6) $AJA = \{AJA_i \mid i = 1, 2, 4, 6, 7, 8, 9\}$

$AJA_1 = \emptyset$, where $\emptyset$ denotes an empty set,

$AJA_2 = \emptyset$, $AJA_4 = \emptyset$,

$AJA_6 = \{a_7\}$, $AJA_7 = \{a_6\}$, $AJA_8 = \{a_9\}$,

$AJA_9 = \{a_8\}$

(7) $\Phi = \{f_{16}, f_{61}, f_{27}, f_{72}, f_{28}, f_{82}, f_{78}, f_{87}, f_{49}, f_{94}\}$ ■

The initial values of the rest of the parameters can not be derived from Q. This set of parameters is called PAR.

$$PAR = < C_R, C_D, C_A, W_A >$$

where $C_R = \{|R_i| \mid R_i \in R^+\}$,

$C_D = \{|D_i| \mid B_i \in \Pi\}$,

$C_A = \{|K_i| \mid a_i \in J\}$,

and $W_A = \{w_i \mid a_i \in T \cup J\}$.

There are basically two approaches to determine the

initial value of PAR. One way is to use estimation. In a database system, the data directory maintains necessary system informations and periodically updates them. By estimating the effect of initial local processing on the value of PAR obtained from the data directory, the initial value is determined. The other way is to get the actual value of PAR after initial local processing from the sites involved in handling the query.

Although estimation is a simpler way, there are many advantages of using the actual value. Since large portions of referenced relations are reduced by initial local processing, the accuracy of the value of PAR after initial local processing is important for accurate estimation of the effects of subsequent semijoins. Also it sometimes happens that users issue queries without being certain of the existence of tuples satisfying the qualification clause. In this case, if any of the results of terms locally processed is null then the final answer to the query must be null. Consequently no further processing of the query is necessary.

These observations are consistent with the conclusion derived in [EPST 80] in favor of dynamic decision making. While dynamic decision making for the entire processing of a query is not practical because of great run-time delay, the delay of the initial dynamic decision is preferable to the inaccuracy of the initial value of PAR. Since initial local processing is performed simultaneously at multiple sites and

the counting on the partial results can be done at the same time, the delay caused by initial dynamic decision is not significant either.

The values of C_D and W_A are static, so they can be obtained from the data directory without the need of further manipulation. The initial values of C_R and C_A are obtained using either approach discussed above. We define the state DB of the database as follows:

$$DB = < INFO, PAR >$$

2.2 Lattice Model of the Effects of Semijoin

In this section, we discuss a method of estimating the effect of semijoins on the database which involves probabilistic analysis. Based on this analysis, a very general model is developed using a lattice which represents the reduction of the set of values of a joining attribute by an arbitrary sequence of semijoins.

2.2.1 Estimation of Effects

In this subsection, an estimation method is presented to determine the change of values of parameters describing database. The basic assumptions are stated and formulas are derived. We have completely relaxed the assumption of attribute independence in the process of a sequence of semijoins. The use of conditional probability to handle attribute dependence has not been considered in past research.

Our strategy for distributed query processing is to construct a sequence of semijoins which minimizes the total amount of data communication. In order to determine the contribution of a member semijoin to the total data flow of the sequence, we must know the volume of data flow involved in the semijoin and the amount of data reduced by the semijoin. The values of the database state before and after the semijoin are sufficient to determine these. Since we start from a given initial database state, our problem is to estimate the next database state induced by a semijoin given the current database state.

Note that the actual database is not affected by any semijoin. Temporary copies of referenced relations are retained after initial local processing and then the query is processed on the temporary copies.

The change of the INFO value is deterministic. It can be simply determined by inspecting the current value of INFO and the semijoin to be used. Since it does not require estimation and is more closely related to the optimization model, we will discuss it in the next chapter.

Among the elements in PAR, the values of C_D and W_A are static during a certain period. So they are not affected by a semijoin. The most important and difficult problem is the estimation of values of C_R and C_A, which are indispensable for the determination of the size of the intermediate result. The estimation of the size of the intermediate result is crucial for the optimization of query processing

in centralized as well as distributed database systems.

We discuss a method of estimating the cardinality of a relation reduced by a semijoin.

Consider a relation R_g and its attribute a_j such that $|K_j| = m$. Define a counting random variable X_i for each $v_i \in K_j$ which counts the number of tuples in R_g in which the value of a_j is v_i. Then for each X_i, possible values are 1, 2, ..., $|R_g|-m+1$ and $\sum_{i=1}^{m} X_i = |R_g|$. Assuming that X_i's are identically distributed,

$$E[\sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i] = mE[X_i] = |R_g|$$

where E[X] denotes the expected value of X. Hence

$$E[X_i] = |R_g|/|K_j| \qquad \text{for all } v_i \in K_j \tag{2.1}$$

If a semijoin changes the value of a variable, we will append 'N' to the name of the variable to designate the new value. After applying a semijoin f_{ij}, R_g and K_j are reduced to R_gN and K_jN, respectively. Since $|R_gN| = \sum_{v_i \in K_jN} X_i$, from (2.1),

$$\begin{aligned} |R_gN| &= E[\textstyle\sum_{v_i \in K_jN} X_i] \\ &= \textstyle\sum_{v_i \in K_jN} E[X_i] \\ &= |K_jN| \times E[X_i] \\ &= |K_jN| \times |R_g| \,/\, |K_j| \end{aligned} \tag{2.2}$$

Therefore, if we know $|K_jN|$ for some a_j which is an attribute of R_g, then we can compute $|R_gN|$. Hence we

investigate the reductions in K_j's caused by semijoins.

There are two different ways by which a K_j can be reduced:

(1) By a semijoin from the equivalent joining attribute of a_j:
If attributes a_i and a_j are in a block, a semijoin f_{ij} reduces K_j to a new set K_iK_j, where K_iK_j denotes $K_i \Omega K_j$, the intersection of K_i and K_j.

(2) By a semijoin to an attribute associated with a_j:
If a_j and a_s are the attributes in the same relation, they are in different blocks. The reduction of K_s by the semijoin f_{rs}, where a_r is in the same block as a_s, also reduces K_j.

First, we discuss the estimation of $|K_jN|$ due to a semijoin f_{ij}. Suppose $B_k = \{a_1, \ldots, a_i, a_j, \ldots, a_n\}$. Consider f_{ij} as the first element in a sequence of semijoins to process a query. If D_k is perceived to be the sample space, any $X \subset D_k$ is the probabilistic event that $v \in X$ for $v \in D_k$. Initially $K_h = A_h$ for all $a_h \in B_k$, and the only restriction on A_h's is that they be subsets of D_k. Therefore the events A_h's are mutually independent events. After f_{ij}, $K_jN = A_iA_j$. From $P(A_iA_j) = P(A_i)P(A_j)$,

$$|K_jN|/|D_k| = (|K_i|/|D_k|) \times (|K_j|/|D_k|)$$

which reduces to

$$|K_jN| = |K_i| \times |K_j| / |D_k| \qquad (2.3)$$

This result, obtained differently, was used in [CCA 80b]. A new procedure must be established to derive the correct estimation of $|K_jN|$ or $|K_iN|$ for a subsequent f_{ij} or f_{ji}, respectively. The reason is that K_j is a subset of K_i as a result of the first f_{ij}. This implies that a dependence between the events K_i and K_j is created as a consequence of the initial f_{ij}. Initially K_i and K_j are independent and the only restriction is that they are both contained in D_k. As a query is processed, more restrictions are added on K_i and K_j by themselves or through some other subset of D_k. We want to characterize the set which imposes a restriction on K_i and K_j in estimating the effect of f_{ij} or f_{ji}.

We generalize (2.3) by using conditional probability. The dependence formed by a semijoin comes from the containment relation among the value sets of joining attributes in the same block. However, not all the subsets of D_k represent the set of values of some $a_i \in B_k$ during query processing. Unless a set can be generated by applying a sequence of semijoins to A_i for some $a_i \in B_k$, the identity of the set is not known. Let ϕ_i be the ith semijoin in a sequence of semijoins.

Definition 2.5: For a block $B_k = \{a_1, \ldots, a_n\}$, let H be a proper subset of D_k. Consider a sequence of semijoins $S_m = \phi_1, \phi_2, \ldots, \phi_m$. H is a <u>reachable set</u> for B_k after ϕ_m if there exists a concatenation V, W of two sequences of semijoins, V and W, where: (i) V is a subsequence of S_m;

(ii) any semijoin in W is between the attributes in B_k; and (iii) the sequence V, W reduces A_i to H for some $a_i \in B_k$.

When a query is partially processed after a sequence S_m, the current set of all reachable sets for B_k, denoted by RS_k, is defined to be the set containing D_k and all reachable sets for B_k after S_m. Since S_m is a special case of the sequence V, W, every K_i is an element of RS_k. S_m is a null sequence after initial local processing. In this case, V, W = W, and the current set of all reachable sets for B_k is called the initial set of all reachable sets for B_k, and is denoted by RS_k^I.

Consider a set $K \in RS_k$ which is the smallest set containing both K_i and K_j. The only information available for K_i, K_j and K is that the values of K_i and K_j are distributed in the set of values K. Hence the knowledge that the event K_j has occurred does not affect the probability of occurrence of the event K_i when the effective sample space is reduced to the event K. That is

$$P(K_i|K_jK) = P(K_i|K)$$

Multiplying both sides by $P(K_j|K)$ gives

$$P(K_iK_j|K) = P(K_i|K)\ P(K_j|K) \qquad (2.4)$$

Hence the events K_i and K_j are conditionally independent given the event K. Note that conditional independence does not imply independence. For any $K' \in RS_k$ such that $K \subset K'$

$$P(K_i|K_jK') = P(K_i|K_jK) = P(K_i|K) > P(K_i|K')$$

Hence the events K_i and K_j are conditionally dependent given the event K'. By using Definition 2.5, we formally define the restricting set.

Definition 2.6: Let $a_i, a_j \in B_k$ for some $B_k \in \Pi$. The <u>restricting set</u> of K_i and K_j is the smallest set in RS_k that is a superset of both K_i and K_j.

For example, initially $K_i = A_i$ and $K_j = A_j$. The restricting set of K_i and K_j is D_k. After f_{ij}, $K_j \subset K_i$. Hence K_i is the restricting set of K_i and K_j.

The estimation of $|K_jN|$ after a semijoin f_{ij} as an element in an arbitrary position in a sequence of semijoins is as follows:

$$P(K_iK_j) = P(K_iK_j|K)P(K) + P(K_iK_j|\bar{K})P(\bar{K})$$

Since $K_iK_j\bar{K} = \emptyset$ because $K_iK_j \subset K$, we have

$$|K_iK_j|/|D_k| = P(K_iK_j|K)(|K|/|D_k|)$$

Multiplying both sides by $|D_k|$ and using (2.4) gives

$$|K_iK_j| = |K|P(K_i|K)P(K_j|K)$$

$$= |K_i| \times |K_j| / |K|$$

Hence $$|K_jN| = |K_i| \times |K_j| / |K| \qquad (2.5)$$

Initially the restricting set is D_k for any K_i and K_j. In this case, (2.5) is reduced to (2.3). In order to evaluate

(2.5), it is necessary to develop a method to calculate the cardinality of the restricting set of K_i and K_j at any point in the sequence of semijoins. This will be discussed in the subsequent subsections.

Next we consider the change in the cardinality of the set of values of a joining attribute a_j by a semijoin f_{rs} which goes to an attribute a_s associated with a_j.

This change is caused by the dependence among associated joining attributes. We consider the dependence among associated joining attributes in set level. In other words, associated joining attributes a_j and a_s are dependent in the sense that the change of $|K_j|$ depends on the change of $|K_s|$, and vice versa. For cardinality estimation purposes, it is proper to handle the attribute dependence in set level.

To estimate $|K_jN|$ due to a semijoin f_{rs}, where a_r is in the same block as a_s, and a_j and a_s are the attributes in the same relation R_g, a solution to the problem considered by Yao [YAO 77] is used. This reduction was ignored in most of the previous semijoin strategies. It was first observed in [CCA 80b] that Yao's solution is applicable for the estimation of $|K_jN|$. $|R_gN|$ due to a semijoin f_{rs} can be computed using (2.2) and (2.5). Suppose $|R_g| = n$, $|K_j| = m$ and $|R_gN| = k$. Then $|K_jN|$ after a semijoin f_{rs} is given by:

$$m \times [1 - \Pi_{i=1}^{k}\{(n\times(1-1/m) - i + 1)/(n - i + 1)\}] \qquad (2.6)$$

2.2.2 Initial Lattice

In this subsection, we show that the set RS_k^I forms a lattice. This, we show by generating the elements of RS_k^I in a step by step manner. The lattice $(RS_k^I, \underline{c})$ is called an initial lattice and it will be used as a building block to generate a lattice for more general cases.

Consider $B_k = \{a_1, a_2, \ldots, a_n\}$. As the dependence among attributes in B_k is formed by a sequence of semijoins, the pairwise relationship of the sets in $\{K_i \mid a_i \in B_k\}$ plays an important role in parameter estimation. Because of the probabilistic nature of estimation, at any instance during the query processing, $K_iK_j \neq \emptyset$ for any $a_i, a_j \in B_k$. At some point in the process of a sequence of semijoins, there are four difference cases:

(1) $K_i = K_j$

(2) $K_i \subset K_j$

(3) $K_j \subset K_i$

(4) $K_i \not\subset K_j$ and $K_j \not\subset K_i$

Let T_{ij} be the restricting set of K_i and K_j. We have the following equivalences:

(a) $K_i = K_j$ iff $T_{ij} = K_i = K_j$

(b) $K_i \subset K_j$ iff $T_{ij} = K_j$

(c) $K_j \subset K_j$ iff $T_{ij} = K_i$

(d) $K_i \not\subset K_j$ and $K_j \not\subset K_i$ iff $T_{ij} \neq K_i$ and $T_{ij} \neq K_j$

Hence the relationship between K_i and K_j can be determined from the knowledge of T_{ij}. Consequently, it is sufficient to develop a model from which we can find the restricting set of K_i and K_j at any point in the sequence of semijoins.

Since the characteristic of the model is common to each block $B_k \in \Pi$, we will drop the block index for the rest of this chapter for simplicity unless it is necessary. For example, B denotes a block, RS denotes the set of all reachable sets for B, and so on.

Without loss of generality, let $B = \{a_1, a_2, \ldots, a_n\}$. A_i is the set of values of a_i after initial local processing. f_{ij} reduces K_j and generates a new set K_jN. Since $K_jN = \{v \in K_j \mid v \in K_i\}$, $K_jN = K_iK_j$. From the remark after Definition 2.5 concerning RS^I, we observe that for any $X \in RS^I$, X can be reached from A_i for $a_i \in B$ after a sequence of semijoins each of which is between the attributes of B. Hence for any $X \in RS^I$, $X = \Omega_{i \in I} A_i$ where $I \subseteq \{1, 2, \ldots, n\}$. We will discuss later the dynamic change of RS. This is caused by the generation of new sets which cannot be represented by the intersection of sets in $\{A_i \mid a_i \in B\}$. Let $RS_1^I = \{A_i \mid a_i \in B\}$, i.e.

$$RS_1^I = \{A_1, A_2, \ldots, A_n\}.$$

The elements in RS_1^I are pairwise incomparable sets with respect to the set-inclusion relation. The set RS_i^I is generated by intersecting i sets in RS_1^I at a time for i = 1, 2, ..., n.

Let ${}_nC_i$ denote the number of combinations of n objects taken i at a time. There are ${}_nC_i$ elements in RS_i^I.

$$RS_2^I = \{A_1A_2, A_1A_3, \ldots, A_{n-1}A_n\}$$

.

.

$$RS_n^I = \{A_1A_2 \ldots A_n\}$$

Let $RS_0^I = \{D\}$. Then

$$RS^I = U_{i=0}^{n} RS_i^I$$

In developing a lattice model, we try to use the standard terminologies and notations from the existing lattice theory. Some definitions used in lattice theory are presented in Appendix B. If new concepts are introduced or it is necessary to modify the existing definition because of the particular structure of our model, separate definitions will be given. In a lattice, g.l.b.{X,Y} is denoted by $X \wedge Y$, and l.u.b.{X,Y} is denoted by $X \vee Y$.

We will show that $(RS^I, \underline{c})$ is a lattice. Let X, Y and Z be the elements in RS^I. Define

g.l.b.{X,Y} = XY,

l.u.b.{X,Y} = minimum{$Z \in RS^I \mid X \underline{c} Z$ and $Y \underline{c} Z$}

It was proved as a theorem in [BIRK 67] that any family of subsets of a set which is closed under intersection forms a lattice under set-inclusion by taking the g.l.b. of two sets as the intersection of the two sets and the l.u.b. of two sets as the smallest set in the family which contains the union of the two sets. It follows that $(RS^I, \underline{c})$ is a lattice.

From the above discussion, we state the theorem which relates the lattice theory and query processing in database systems.

Theorem 2.1: Given a block $B \in \Pi$, the initial set of all reachable sets RS^I forms a lattice under set-inclusion with

g.l.b.{X,Y} = XY and

l.u.b.{X,Y} = minimum{$Z \in RS^I$ | $X \subseteq Z$ and $Y \subseteq Z$}

for any $X,Y \in RS^I$. In estimating the effect of f_{ij} or f_{ji} for $a_i, a_j \in B$, the restricting set is l.u.b.{K_i, K_j} and the reduced set is g.l.b.{K_i, K_j}. ∎

The initial lattice $(RS^I, \subseteq)$ is denoted by L^I. The greatest element of L^I is D, the domain of the attributes in the block. Since a block is finite and nonempty, the lattice which models our problem is also finite and nonempty. The structure of the initial lattice and the expanded sublattice, which will be discussed in the next subsection, are exactly the same. The structure of L^I is important in generalizing the model. Especially the concept of level, which has already been indicated by the subscript i in RS_i^I, is essential in expanding the lattice and performing the search in the lattice when searching is necessary during the query optimization procedure. We will devote the rest of the subsection to identify the structure of L^I.

First, the relationships among {RS_i^I | $RS_i^I \subset RS^I$} are investigated. These relationships will be used to establish the level structure in L^I.

Lemma 2.1: The following is true for RS^I:

(i) For any $X_j \in RS_j^I$ and $0 \leq i < j \leq n$, there exists $X_i \in RS_i^I$ such that $X_j \subset X_i$,

(ii) For any $X_i \in RS_i^I$ and $0 \leq i < j \leq n$, there is no $X_j \in RS_j^I$ such that $X_i \subset X_j$,

(iii) If $X_1, X_2 \in RS_i^I$ and $X_1 \neq X_2$ then X_1 and X_2 are incomparable,

(iv) If $X_j \subset X_i$ for $X_i \in RS_i^I$ and $X_j \in RS_j^I$ then $i < j$.

Proof: Obvious.

Lemma 2.2: In RS^I, if $X_j \subset X_i$ and there is no X_k such that $X_j \subset X_k \subset X_i$ for $X_i \in RS_i^I$, $X_j \in RS_j^I$ and $X_k \in RS^I$ then $j = i+1$.

Proof: Obvious.

The next lemma further restricts the structure of the initial lattice.

Lemma 2.3: L^I is a boolean lattice.

Proof: Let $P^+(X)$ be the power set of a set X. For any set X, $(P^+(X), \subseteq)$ is a boolean lattice under set-inclusion. Intersection and union are used for meet and join, respectively. Hence $L = (P^+(RS_1^I), \subseteq)$ is a boolean lattice. Define a function θ_1: L --> L such that $\theta_1(Y) = Y'$, a set complement of $Y \in P^+(RS_1^I)$. It can be shown that θ_1 is a dual isomorphism. Hence $L' = (\{\theta_1(Y) | Y \in P^+(RS_1^I)\}, \subseteq)$ is also a boolean lattice with the ordering relation reversed. Define a function θ_2: L' --> L^I such that $\theta_2(X) = D\ \Omega\ (\Omega_{A_i \in X}$

Theorem 2.1: Given a block $B \in \Pi$, the initial set of all reachable sets RS^I forms a lattice under set-inclusion with

g.l.b.{X,Y} = XY and

l.u.b.{X,Y} = minimum{$Z \in RS^I$ | $X \subseteq Z$ and $Y \subseteq Z$}

for any $X,Y \in RS^I$. In estimating the effect of f_{ij} or f_{ji} for $a_i, a_j \in B$, the restricting set is l.u.b.{K_i, K_j} and the reduced set is g.l.b.{K_i, K_j}. ∎

The initial lattice $(RS^I, \subseteq)$ is denoted by L^I. The greatest element of L^I is D, the domain of the attributes in the block. Since a block is finite and nonempty, the lattice which models our problem is also finite and nonempty. The structure of the initial lattice and the expanded sublattice, which will be discussed in the next subsection, are exactly the same. The structure of L^I is important in generalizing the model. Especially the concept of level, which has already been indicated by the subscript i in RS_i^I, is essential in expanding the lattice and performing the search in the lattice when searching is necessary during the query optimization procedure. We will devote the rest of the subsection to identify the structure of L^I.

First, the relationships among {RS_i^I | $RS_i^I \subset RS^I$} are investigated. These relationships will be used to establish the level structure in L^I.

Lemma 2.1: The following is true for RS^I:

A_i). Clearly θ_2 is an isomorphism. It follows that L^I is a boolean lattice. ∎

In a poset P, 'a covers b' means that a > b and there is no x such that a > x > b for any x ∈ P. We introduce the concept of level in the lattice.

Definition 2.7: A <u>leveled lattice</u> is a lattice L with a strict antitone function v: L --> Z from L to the chain of all integers such that if x covers y, then

$$v[x] = v[y] - 1 \text{ for all } x,y \in L.$$

Definition 2.8: The <u>length</u> l(L) of a lattice L is the l.u.b. of the lengths of the chains in L, where a chain in L is a subset L' of L such that $x \leq y$ or $y \leq x$ for all $x,y \in L'$.

When l(L) is finite, L is said to be of finite length.

Definition 2.9: In a lattice L of finite length, the <u>depth</u> d[x] of an element x ∈ L is the l.u.b. of the lengths of the chains $I = x_0 > x_1 > \ldots > x_k = x$ between the greatest element I and x.

It is obvious that d[O] = l(L) where O is the least element. The following theorem completely characterizes the structure of L^I.

Theorem 2.2: L^I is a leveled boolean lattice with v[X] = d[X] for all $X \in RS^I$.

Proof: In a boolean lattice $L^I = (RS^I, \underline{c})$, $RS^I = U_{i=0}^n RS_i^I$ and D is the greatest element. The mathematical induction is used. The length of maximal chains from D to X is 1 if and only if $X \in RS_1^I$. Hence $d[X_1] = 1$ if and only if $X_1 \in RS_1^I$. From Lemma 2.1.(i), for any $X_{i+1} \in RS_{i+1}^I$, there exists at least one $X_i \in RS_i^I$ such that $X_{i+1} \underline{c} X_i$. Also from lemma 2.1.(ii), there is no $X_j \in RS_j^I$ such that $X_{i+1} \underline{c} X_j$ for $j > i+1$. Hence $d[X_{i+1}] = i+1$ if and only if $X_{i+1} \in RS_{i+1}^I$. Consequently, $d[X_i] = i$ if and only if $X_i \in RS_i^I$ for all i. Now let $v[X] = d[X]$ and consider $X_i \in RS_i^I$ and $X_j \in RS_j^I$. $X_i > X_j$ implies $< D, X_i, X_j >$ is a chain. clearly $v[X_i] < v[X_j]$. If X_i covers X_j, then from Lemma 2.2, $i = j-1$. Hence $v[X_i] = v[X_j] - 1$. ∎

Note that $v[X] = i$ if and only if $X \in RS_i^I$ for all $X \in L^I$. Since L^I is completely characterized by the elements in RS_1^I, we can consider that L^I is generated by the elements in RS_1^I.

Definition 2.10: The <u>dimension</u> $\dim(L^I)$ of the initial lattice L^I is the cardinality of the elements in RS_1^I. The elements in RS_1^I are the <u>generators</u> of L^I.

For the initial lattice L^I, $l(L^I) = \dim(L^I)$. If $\dim(L^I) = n$, the number of elements in level i is ${}_nC_i$ and the total number of elements in L^I is

$$\sum_{i=0}^{n} {}_nC_i = 2^n$$

Since L^I is of finite length and leveled by d[X], it

satisfies the condition that all maximal chains between the same endpoints have the same finite length which is known as the Jordan-Dedekind chain condition.

We give an example of an initial lattice model to illustrate the effect of a sequence of semijoins.

Example 2.3

In this example, we construct L^I for $B = \{a_1, a_2, a_3\}$. Then L^I is used to determine the restricting set and the reduced set for each semijoin in a sequence of semijoins between the attributes in B.

From the initial value sets $\{A_1, A_2, A_3\}$:

$$RS_1^I = \{A_1, A_2, A_3\}$$
$$RS_2^I = \{A_1A_2, A_1A_3, A_2A_3\}$$
$$RS_3^I = \{A_1A_2A_3\}$$

With $RS_0^I = \{D\}$, RS^I is given by:

$$RS^I = \{D, A_1, A_2, A_3, A_1A_2, A_1A_3, A_2A_3, A_1A_2A_3\}$$

It is obvious that $v[X] = i$ iff $X \in RS_i^I$. The Hasse diagram of L^I is depicted in Figure 2.1.

Consider an arbitrary sequence of semijoins

$$f_{21}, f_{32}, f_{13}, f_{21}, f_{12}$$

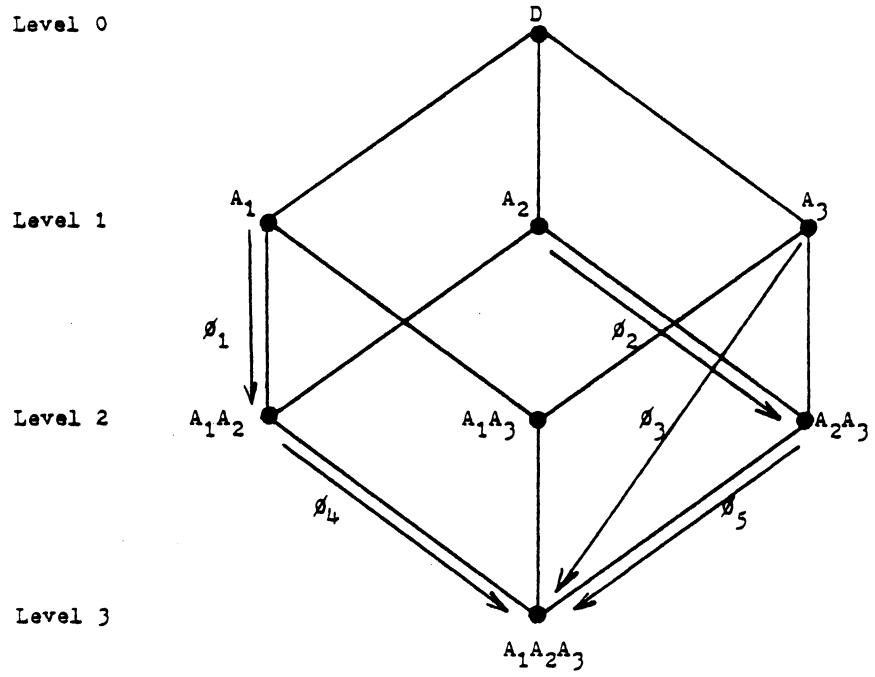
Let ϕ_i: the ith semijoin in the sequence

T_i: the restricting set for ϕ_i

E_i: the reduced set by ϕ_i

Initially $K_i = A_i$ for all $a_i \in B$. If $\phi_i = f_{jk}$ then K_k is reduced to E_i. The effect of each semijoin is:

(1) $\phi_1 = f_{21}$



The Hasse Diagram of the Initial Lattice
and the Effect of Each Semijoin for Example 2.3

Figure 2.1

$T_1 = K_1 \vee K_2 = A_1 \vee A_2 = D$

$E_1 = K_1 \wedge K_2 = A_1 \wedge A_2 = A_1A_2$

(2) $\phi_2 = f_{32}$

$T_2 = K_2 \vee K_3 = A_2 \vee A_3 = D$

$E_2 = K_2 \wedge K_3 = A_2 \wedge A_3 = A_2A_3$

(3) $\phi_3 = f_{13}$

$T_3 = K_1 \vee K_3 = E_1 \vee A_3 = D$

$E_3 = K_1 \wedge K_3 = E_1 \wedge A_3 = A_1A_2A_3$

(4) $\phi_4 = f_{21}$

$T_4 = K_1 \vee K_2 = E_1 \vee E_2 = A_2$

$E_4 = K_1 \wedge K_2 = E_1 \wedge E_2 = A_1A_2A_3$

(5) $\phi_5 = f_{12}$

$T_5 = K_1 \vee K_2 = E_4 \vee E_2 = A_2A_3$

$E_5 = K_1 \wedge K_2 = E_4 \wedge E_2 = A_1A_2A_3$

Since $K_1 = K_2 = K_3 = A_1A_2A_3$ after ϕ_5, K_i will not be reduced by any semijoin for all $a_i \in B$. The effect of each ϕ_i is shown in Figure 2.1. ■

2.2.3 Expanded Sublattice

In this subsection, the preliminary step in generalizing the initial lattice model is presented. This subsection and the next systematically generate the elements of RS at any instance during the query processing in order to identify its structure and useful properties.

For $B \in \Pi$, K_i for any $a_i \in B$ is in L^I before any semijoin to an attribute associated with a_i is made. The effect of a semijoin to $a_k \in AJA_i$ on a_i cannot be modeled by

L^I. In other words, L^I cannot model the effect of semijoin between the attributes in one block has on the attribute in another block.

Definition 2.11: An <u>expanded sublattice</u> is a lattice generated to model the effect of a semijoin f_{jk}, between the equivalent joining attributes a_j and a_k in one block, on the attribute a_i, which is associated with a_k, in another block.

The expanded sublattice with the greatest element X will be denoted by E^X and the corresponding poset will be denoted by RS^X.

We shall illustrate the generation of the expanded sublattice by generating the first expanded sublattice. Suppose $K_i = X \in L^I$ for $a_i \in B$ before f_{jk} is performed for $a_k \in AJA_i$ and $a_j \in EJA_k$. The reduction of K_k results in the reduction of K_i. The reduced K_i, namely K_iN, cannot be expressed by the intersection of sets in RS_1^I. The only restriction on K_iN is that it has to be a subset of X. Hence it is incomparable with the sets in L^I covered by X. The generation of an expanded sublattice that reflects f_{jk} can be described as follows:

(ES1) Suppose $RS_1^I = \{A_1, A_2, \ldots, A_n\}$. Let A_{n+1} represent the K_iN formed by semijoin f_{jk}. Then $A_{n+1} \subset X$ and $A_{n+1} \not\subset A_i$ if $X \not\subset A_i$.

(ES2) Let C^X be the set of elements in L^I covered by X. Let $RS_1^X = \{X \cap A_{n+1}\} \cup C^X$.

(ES3) Generate E^X with X as the greatest element and RS_1^X as

the set of generators.

A_{n+1} is the reduced K_i by f_{jk}. From Lemma 2.1, any two elements in C^X are incomparable. From (ES1), $A_{n+1}X$ is incomparable with any element in C^X. Consequently the elements in RS_1^X are pairwise incomparable. From (ES3), if $|RS_1^X| = |RS_1^I|$ then E^X and L^I are isomorphic. Since $C^X \subset E^X$, for any $Y \in L^I$, $Y \subset X$ implies $Y \in E^X$.

Since $A_{n+1} \subseteq X$, $A_{n+1} = XA_{n+1}$. We prefer to denote the new element by XA_{n+1}. This will enable the elements in the same level of E^X to be represented by the intersection of the same number of elements in the set $\{A_1, A_2, \ldots, A_n, A_{n+1}\}$. Also all the elements smaller than X are represented by the intersection of X and the elements chosen from the set $\{A_1, \ldots, A_n, A_{n+1}\}$. Therefore, A_{n+1} is used when the set is used as a basic element while XA_{n+1} is used to designate a specific element in a lattice. For the mth expanded sublattice, the reduced set will be represented by A_{n+m}.

A lattice which is generated as L^I by the set of incomparable elements covered by the greatest element is called an <u>L^I-type lattice</u>. The L^I-type lattice with the greatest element Z is denoted by L^Z and its least element is denoted by O^Z. The set of generators of L^Z is denoted by G^Z. The expanded sublattice is an L^I-type lattice.

Definition 2.12: In an L^I-type lattice L^Z of finite length, the <u>relative depth</u> $d_Z[X]$ of an element $X \in L^Z$ is the

l.u.b. of the lengths of the chains between Z and X.

The following lemma provides an useful property for proving subsequent lemmas.

Lemma 2.4: Let $W \in L^Z$ and L^W be an L^I-type sublattice of L^Z generated by the elements in L^Z covered by W. Then, $O^Z = O^W$.

Proof: Let $RS_1^Z = \{Z_1, Z_2, \ldots, Z_n\}$ and $I = \{1, 2, \ldots, n\}$. If $d_Z[W] = k$ then $W = \Omega_{j \in J} Z_j$ for $J \subset I$ and $|J| = k$. There are n-k elements in L^Z covered by W. Denote these n-k elements by $W_1, W_2, \ldots, W_{n-k}$. Without loss of generality, let $J = \{1, 2, \ldots, k\}$. It follows that $W = \Omega_{j=1}^{k} Z_j$ and $W_h = W \Omega Z_{k+h}$ for $h = 1, 2, \ldots, n-k$. Hence

$$O^W = \Omega_{h=1}^{n-k} W_h = W \Omega (\Omega_{h=1}^{n-k} Z_{k+h})$$
$$= (\Omega_{h=1}^{k} Z_h) \Omega (\Omega_{h=1}^{n-k} Z_{k+h}) = \Omega_{h=1}^{n} Z_h = O^Z. \quad \blacksquare$$

The following example shows when and how to generate an expanded sublattice.

Example 2.4

Assume that we already have L^I as illustrated in Example 2.3. Let $B' = \{a_i, a_j\}$ be another block and $a_j \in AJA_3$. Consider the following sequence of semijoins:

$$f_{21}, f_{32}, f_{ij}$$

The effect of $\phi_1 = f_{21}$ and $\phi_2 = f_{32}$ are the same as in Example 2.3. As a result of $\phi_3 = f_{ij}$, $K_3 = A_3$ is reduced. The generation of L^{A_3} is as follows:

(1) Since $RS_1^I = \{A_1, A_2, A_3\}$, introduce A_4 such that $A_4 \subset A_3$, $A_4 \not\subset A_1$ and $A_4 \not\subset A_2$

(2) Since $C^{A_3} = \{A_1A_3, A_2A_3\}$, $RS_1^{A_3} = \{A_1A_3, A_2A_3, A_3A_4\}$

(3) Generate E^{A_3} with A_3 as the greatest element and $RS_1^{A_3}$ as the set of generators. $RS^{A_3} =$ $\{A_3, A_1A_3, A_2A_3, A_3A_4, A_1A_2A_3, A_1A_3A_4, A_2A_3A_4, A_1A_2A_3A_4\}$

The relative depth $d_{A_3}[X] = i$ iff $X \in RS_i^{A_3}$. The reduced set of K_3 by ϕ_3 is A_3A_4. The Hasse diagram of L^{A_3} and the effect of ϕ_3 are shown in Figure 2.2. ■

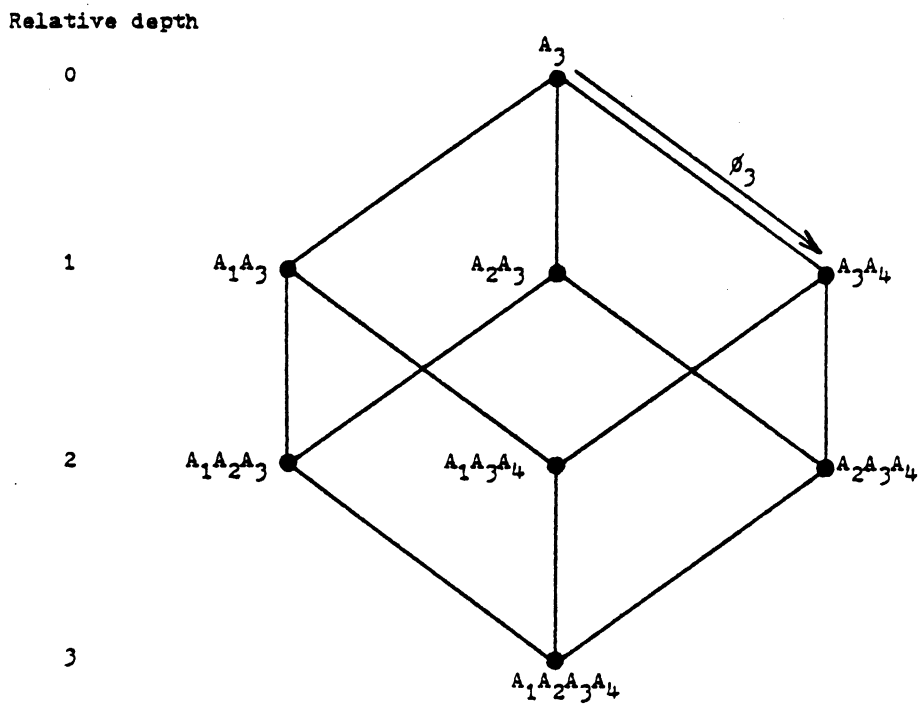
2.2.4 Expanded Lattice

In this subsection, we show that RS at any instance during the query processing also forms a lattice. An algorithm to generate such a lattice is presented and its correctness is proved.

Definition 2.13: An <u>expanded lattice</u> is the smallest lattice that contains the initial lattice and all the expanded sublattices subsequently generated.

The expanded lattice models the state transitions which are caused by the dependence among equivalent joining attributes and associated joining attributes mixed together.

The expanded lattice for a block B models RS, and is denoted by L. Initially $L = L^I$. Subsequently, L dynamically grows as a query is processed by a sequence of semijoins which contains a semijoin between the attributes



The Hasse Diagram of the Expanded Sublattice and the Effect of a Semijoin for Example 2.4

Figure 2.2

of a block other than B. Different expanded lattices will be generated for different sequences. Consider the new expanded lattice after the generation of E^X, for $X \in L$. The new expanded lattice is denoted by L^N. We define the lattice union as follows:

Definition 2.14: Let $L_1 = (P_1, \leq)$ and $L_2 = (P_2, \leq)$ be lattices such that:

(1) P_1 and P_2 are the collections of subsets of a set D,

(2) In both L_1 and L_2, $\leq$ denotes set-inclusion,

(3) In both L_1 and L_2, g.l.b. and l.u.b. are defined to be the same as in L^I,

The <u>lattice union</u> $L_1 \cup L_2$ is $(P_1 \cup P_2, \leq)$ with g.l.b., l.u.b. and $\leq$ as defined in L_1 and L_2.

L_1L_2, L_1-L_2 and L_2-L_1 are similarly defined. It is clear that $L_1 \cup L_2$, L_1L_2, L_1-L_2 and L_2-L_1 are all posets.

It can be shown that $L \Omega E^X$ is a lattice. Since the elements of L and E^X are subsets of the domain D, $L \cup E^X$ is well-defined. However, $L \cup E^X$ is not always a lattice. The following lemma determines when $L \cup E^X$ is a lattice.

Lemma 2.5: Let O be the least element of L. $L \cup E^X$ is a lattice if and only if $O \in E^X$.

Proof: We have to show that $X_i \vee X_j \in LUE^X$ and $X_i \wedge X_j \in LUE^X$ for any $X_i, X_j \in LUE^X$. It is obvious when $X_i, X_j \in L$ or $X_i, X_j \in E^X$. Assume $X_i \in L-E^X$ and $X_j \in E^X-L$. We will first show that $X_i \vee X_j \in LUE^X$. This does not require $O \in E^X$. Let

UB be the set of upper bounds of X_i and X_j, then UB c L. Since the greatest element of E^X is X, $X_j \leq X \leq D$. Also $X_i \leq D$. Hence D $\in$ UB and UB $\neq \emptyset$. Since $U_i \wedge U_j \in$ UB for any $U_i, U_j \in$ UB, $X_i \vee X_j$ = g.l.b.UB $\in$ L. Now consider $X_i \wedge X_j$.

(->) Suppose $X_i \wedge X_j \in LUE^X$ and $O \notin E^X$. Since $O \notin E^X$, O is a minimal element in L U E^X. E^X is an expanded sublattice such that $X' \notin L$ for some $X' \in RS_1^X$. Hence $O^X \notin L$, which implies O^X is also a minimal element in L U E^X. Since O and O^X are both minimal elements in L U E^X, $O \wedge O^X \notin LUE^X$. This contradicts that $X_i \wedge X_j \in LUE^X$.

(<-) For any $X_i \in L\text{-}E^X$ and $X_j \in E^X\text{-}L$, $X_i \wedge X_j = X_i \wedge (X \wedge X_j)$. From the associativity of $\wedge$, $X_i \wedge (X \wedge X_j) = (X_i \wedge X) \wedge X_j$. Since $X_i, X \in L$, $X_i \wedge X \in L$. On the other hand, for any $X_k \in L$, $O \in E^X$ implies $X_k \in E^X$ whenever $X_k \leq X$. Since $X_i \wedge X \leq X$, $X_i \wedge X \in E^X$. $X_i \wedge X \in E^X$ and $X_j \in E^X$ implies $(X_i \wedge X) \wedge X_j = X_i \wedge X_j \in E^X$.■

Therefore sometimes it is sufficient to generate an expanded sublattice to form L^N and sometimes not. If $O \notin E^X$, we have to construct the smallest lattice containing L U E^X. It is necessary to construct a lattice in order to generalize the properties of RS^I to RS so that the g.l.b. and l.u.b. of any two sets in $\{K_i | a_i \in B\}$ can be determined at any time during the query processing. The algorithm to generate an expanded lattice is given below.

```
PROCEDURE  GEN_LAT (L, E^X)
  // GEN_SUBLAT (X, G) is a procedure to generate   //
  // an L^I-type lattice with the greatest element X  //
  // and the set of generators G //
```

```
IF    O ∈ E^X
THEN  L^N <- L U E^X
ELSE
  BEGIN
    Y_1 <- max {Y | Y covers elements in L - E^X
                    and elements in E^X - L}
    C^Y1 <- {Y | Y ∈ L U E^X and Y_1 covers Y}
    L^Y1 <- GEN_SUBLAT (Y_1, C^Y1)
    IF       O ∈ L^Y1
    THEN     L^N <- L U E^X U L^Y1
    ELSE
      BEGIN
        m <- 1
        WHILE   O ∉ L^Ym
        DO    BEGIN
                Y_m+1 <- max {Y | Y covers elements in
                        L - L^Ym and elements in L^Ym - L}
                C^Ym+1 <- {Y | Y ∈ L U E^X U (U_k=1^m L^Yk)
                                and Y_m+1 covers Y}
                m <- m+1
                L^Ym <- GEN_SUBLAT (Y_m, C^Ym)
              END
        L^N <- L U E^X U (U_k=1^m L^Yk)
      END
  END
END GEN_LAT
```

We show that GEN_LAT (L, E^X) generates the smallest lattice containing L U E^X.

Lemma 2.6: Suppose $O \notin E^X$. L U E^X U ($U_{k=1}^m L^{Y_k}$) is the smallest lattice containing L U E^X if and only if $O \in L^{Y_m}$.

Proof: (<-) Consider E^X and L^{Y_1}. Since $Y_1 \in L \cap E^X$, Y_1

$\in E^X$. Let L^{Y_1X} be the sublattice of E^X generated by the elements covered by Y_1. From Lemma 2.4, $O^X = O^{Y_1X}$. Since $L^{Y_1X} \subset L^{Y_1}$, $O^X \in L^{Y_1}$. From Lemma 2.5, $E^X \cup L^{Y_1}$ is a lattice. Following the same procedure, $E^X \cup (\cup_{k=1}^{m} L^{Y_k})$ is a lattice. Since $O \in L^{Y_m}$, $O \in E^X \cup (\cup_{k=1}^{m} L^{Y_k})$. Consider $X_i \in L - \{E^X \cup (\cup_{k=1}^{m} L^{Y_k})\}$ and $X_j \in \{E^X \cup (\cup_{k=1}^{m} L^{Y_k})\} - L$. Using the same arguments of the proof of Lemma 2.5, $X_i \vee X_j \in LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$. Suppose $X_j \in L^{Y_h}$ for some h, where $0 \le h \le m$, and $L^{Y_0} = E^X$. $X_i \wedge X_j = X_i \wedge (Y_h \wedge X_j) = (X_i \wedge Y_h) \wedge X_j$. Since $Y_h \ge X_i \wedge Y_h \ge O$ and $O \in E^X \cup (\cup_{k=1}^{m} L^{Y_k})$, $X_i \wedge Y_h \in L^{Y_g}$ for some $g \ge h$. Hence $X_i \wedge X_j \in E^X \cup (\cup_{k=1}^{m} L^{Y_k})$. Consider $LUE^X \cup (\cup_{k=1}^{m} L^{Y_k}) - \{Z\}$ for any $Z \in \cup_{k=1}^{m} L^{Y_k} - (LUE^X)$ and let C_Z be the set of elements in $LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$ covering Z. For any $Z_1, Z_2 \in C_Z$, $Z_1 \wedge Z_2 \notin LUE^X \cup (\cup_{k=1}^{m} L^{Y_k}) - \{Z\}$. Hence $LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$ is the smallest lattice.

(->) If $O \notin L^{Y_m}$ then O and O^{Y_m} are minimal elements in $LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$. So $O \wedge O^{Y_m} \notin LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$. ∎

Consequently $LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$ is the new expanded lattice after the generation of E^X. Therefore $L^N = LUE^X \cup (\cup_{k=1}^{m} L^{Y_k})$. Denote the level function and the depth function of L^N by v_N and d_N, respectively. Summing up the previous discussions, we have the following theorem.

Theorem 2.3: $L^N = L \cup E^X \cup (\cup_{k=1}^{m} L^{Y_k})$ is the smallest

leveled lattice containing $L \cup E^X$ with $v_N[Z] = d_N[Z]$ for all $Z \in L^N$ where $d_N[Z]$ is given such that:

(i) $d_N[Z] = d[Z]$ for $Z \in L$

(ii) $d_N[Z] = d_X[Z] + d[X]$ for $Z \in E^X$.

(iii) $d_N[Z] = d_{Y_k}[Z] + d[Y_k]$ for $Z \in L^{Y_k}$ and $1 \leq k \leq m$.

Proof: We have only to show that L^N is a leveled lattice. This follows immediately from Theorem 2.2 and the fact that L^N is a union of a finite number of L^I-type lattices. Consider $Z \in L^H L^M$, where L^H and L^M are either E^X or L^I or one of the L^{Y_k}'s. We will show that $v_N[Z]$ is consistent whether we use $d_H[Z]$ or $d_M[Z]$. Suppose $Z_1 \in L^H$ covers Z and there is $Z_2 \in L^M$ such that $Z_1 > Z_2 > Z$. Since L^H is an L^I-type lattice $Z_1, Z \in L^H$ implies $Z_2 \in L^H$. This is a contradiction. ∎

The fact that an expanded lattice is a leveled lattice can be used to reduce the search space when some elements of the lattice are stored and they are searched to retrieve the data associated with those elements. Suppose an expanded lattice L contains m expanded sublattices. Since every element in level k, $1 \leq k \leq n+m$, of L is represented by the intersection of k elements in the set $\{A_1, A_2, \ldots, A_n, A_{n+1}, \ldots, A_{n+m}\}$, we can easily identify the level of an element. Therefore, if the elements in the same level of L are stored together, we have only to search the elements in the level in which the element being searched is. We give an example which shows how to generate an expanded lattice.

Example 2.5

In this example, we generate an expanded lattice which models the interaction among equivalent joining attributes as well as among associated joining attributes by a sequence of semijoins.

Let $B_1 = \{a_1, a_2, a_3, a_4\}$,

$B_2 = \{a_g, a_h\}$,

$B_3 = \{a_i, a_j\}$,

$B_4 = \{a_k, a_m, a_n\}$,

and $a_g \in AJA_1$, $a_n \in AJA_2$, $a_i \in AJA_4$

Consider a sequence of semijoins

$f_{12}, f_{43}, f_{21}, f_{hg}, f_{34}, f_{kn}, f_{ji}$

L^I is shown in Figure 2.3. Initially $L = L^I$.

The effect of each semijoin is given as follows:

(1) $\phi_1 = f_{12}$

- K_2 is reduced from A_2 to A_1A_2.

(2) $\phi_2 = f_{43}$

- K_3 is reduced from A_3 to A_3A_4.

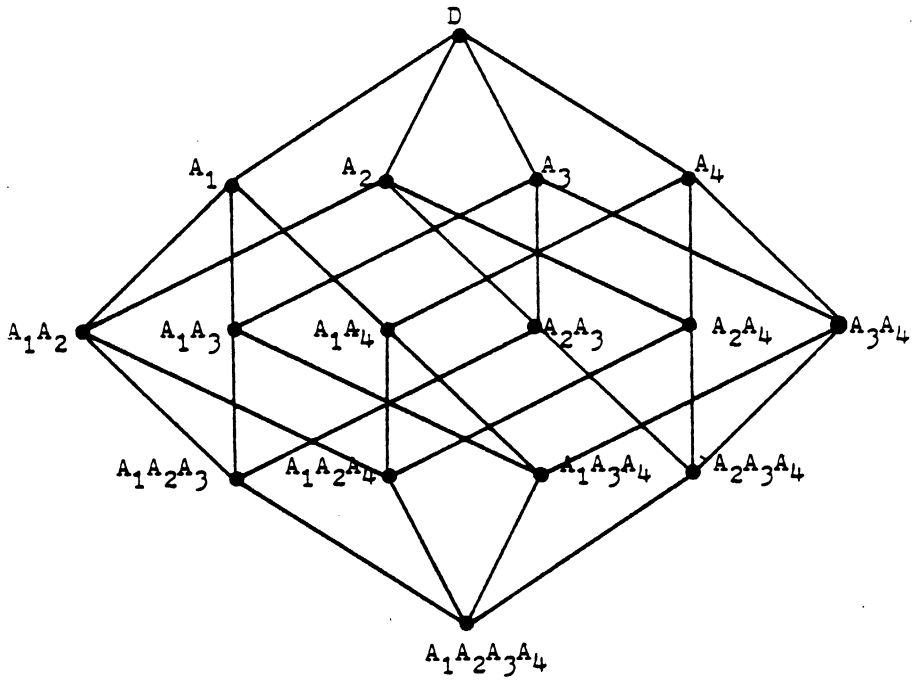
(3) $\phi_3 = f_{21}$

- K_1 is reduced from A_1 to A_1A_2.

(4) $\phi_4 = f_{hg}$

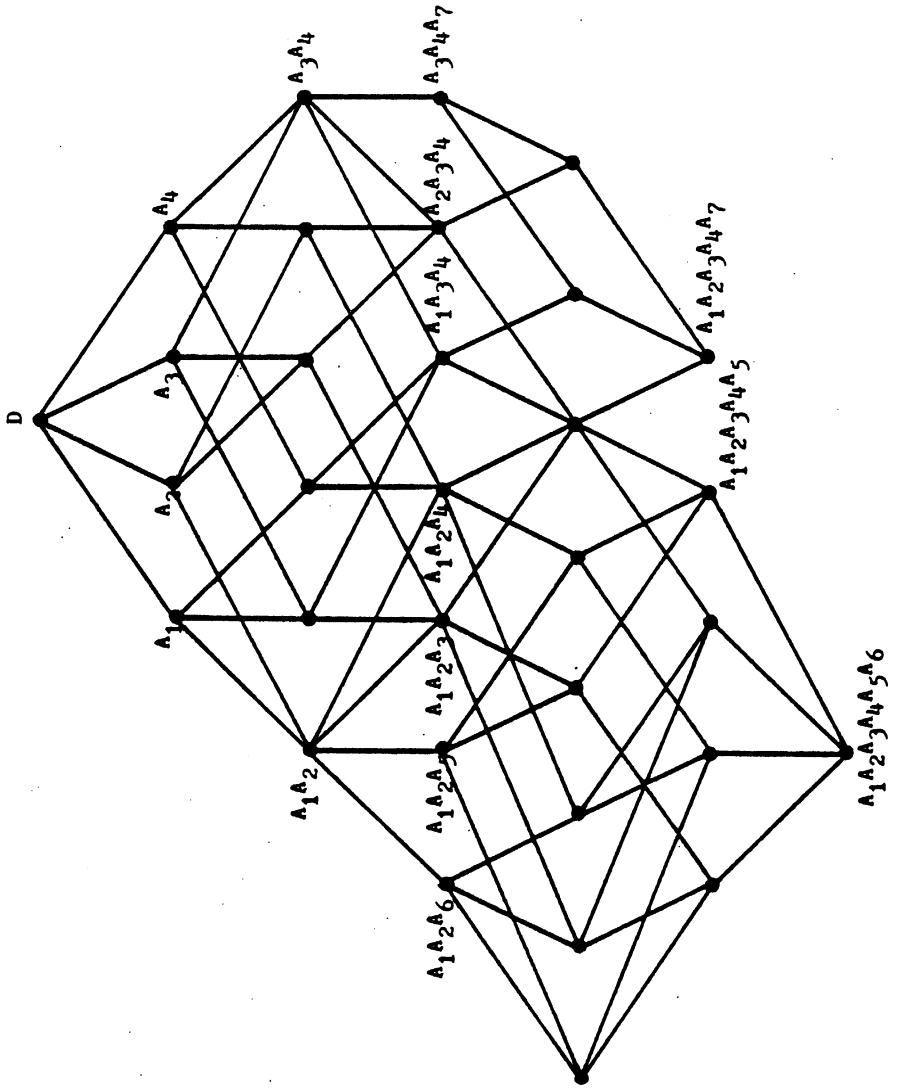
- Introduce A_5 to create $A_1A_2A_5$.
- Generate $E^{A_1A_2}$ with $G^{A_1A_2} = \{A_1A_2A_3, A_1A_2A_4, A_1A_2A_5\}$.
- Since $O = A_1A_2A_3A_4 \in E^{A_1A_2}$, $L^N = LUE^{A_1A_2}$.
- K_1 is reduced from A_1A_2 to $A_1A_2A_5$.

(5) $\phi_5 = f_{34}$



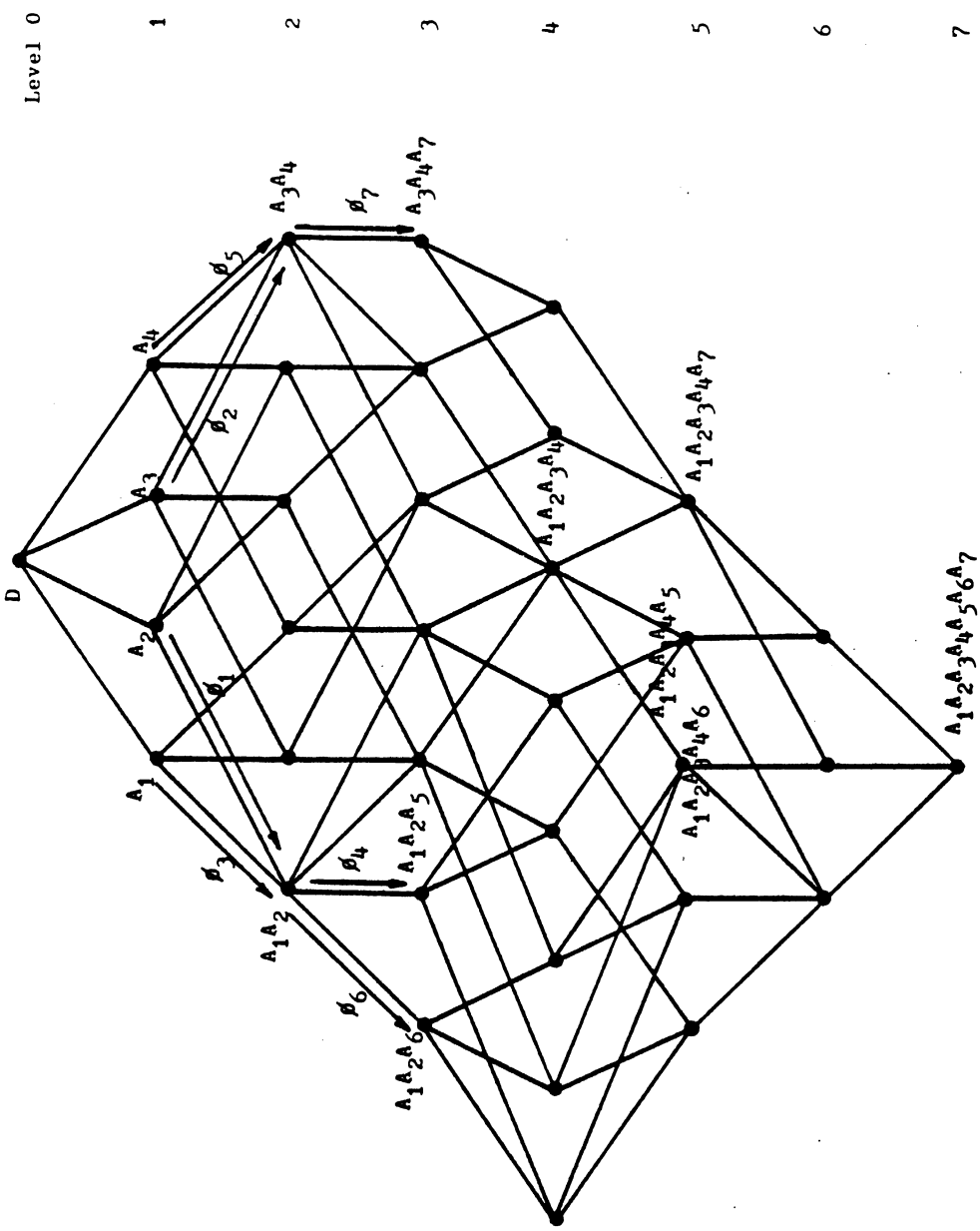
The Hasse Diagram of the Initial Lattice for Example 2.5

Figure 2.3



The Hasse Diagram of the Intermediate Poset for Example 2.5

Figure 2.4



The Hasse Diagram of the Expanded Lattice and the Effect of Each Semijoin For Example 2.5

Figure 2.5

- K_4 is reduced from A_4 to A_3A_4

(6) $\phi_6 = f_{kn}$

- Introduce A_6 to create $A_1A_2A_6$.
- Generate $E^{A_1A_2}$ with $G^{A_1A_2}$ = $\{A_1A_2A_3, A_1A_2A_4, A_1A_2A_5, A_1A_2A_6\}$.
- Since $O = A_1A_2A_3A_4A_5 \in E^{A_1A_2}$, $L^N = LUE^{A_1A_2}$.
- K_2 is reduced from A_1A_2 to $A_1A_2A_6$.

(7) $\phi_7 = f_{ji}$

- Introduce A_7 to create $A_3A_4A_7$.
- Generate $E^{A_3A_4}$ with $G^{A_3A_4} = \{A_1A_3A_4, A_2A_3A_4, A_3A_4A_7\}$.
- Since $O = A_1A_2A_3A_4A_5A_6 \notin E^{A_3A_4}$, $LUE^{A_3A_4}$ is not a lattice. $LUE^{A_3A_4}$ is shown in Figure 2.4. Only the greatest element, the least element and the generators of each expanded sublattice are labeled.
- In this example, $Y_1 = A_1A_2A_3A_4$. Generate L^{Y_1} with $G^{Y_1} = \{A_1A_2A_3A_4A_6, A_1A_2A_3A_4A_5, A_1A_2A_3A_4A_7\}$.
- Since $O \in L^{Y_1}$, $L^N = LUE^{A_3A_4}UL^{Y_1}$. L^N is shown in Figure 2.5.
- K_4 is reduced from A_3A_4 to $A_3A_4A_7$. ■

The expanded lattice is the generalization of the initial lattice L^I. The current set of all reachable sets has the same properties as those for RS^I given in Theorem 2.1. We summarize the above discussion in the following theorem:

Theorem 2.4: Given a block $B = \{a_1, a_2, \ldots, a_n\}$, the current set of all reachable sets RS at any point in the sequence of semijoins forms a lattice under set-inclusion with

$g.l.b.\{X,Y\} = XY$ and

$l.u.b.\{X,Y\} = \text{minimum } \{Z \in RS \mid X \subseteq Z \text{ and } Y \subseteq Z\}$

for any $X,Y \in RS$. In estimating the effect of f_{ij} or f_{ji} for $a_i, a_j \in B$,

(i) $K_i, K_j \in RS$,

(ii) The restricting set is $l.u.b.\{K_i, K_j\}$,

and (iii) The reduced set is $g.l.b.\{K_i, K_j\}$. ■

The formula to compute $|K_jN|$ resulting from f_{ij} can be obtained using Theorem 2.4. Substituting $|K| = |l.u.b.\{K_i, K_j\}|$ in (2.5), we have the following basic formula:

$$|K_jN| = |g.l.b.\{K_i, K_j\}|$$

$$= |K_i| \times |K_j| \;/\; |l.u.b.\{K_i, K_j\}| \qquad (2.7)$$

We have modelled the effect of a sequence of semijoins using a lattice and carried out a structural analysis of the lattice model to establish a basis for the future research. The theory developed in this chapter is general enough that it can be applied to a broad class of query processing problems. The lattice model for estimating the reduction of relations during query processing can be used for broadcasting communication networks as well as for point-to-point communication networks.

The lattice model not only enables us to compute $|K_i|$'s but also gives information about the containment relation among K_i's. This fact can be used to increase the efficiency of the query optimization algorithm. For example, if $K_i \subseteq K_j$, f_{ji} can be excluded from the set of the next possible semijoins because f_{ji} does not reduce any relation. Also if K_i c K_j, $K_jN = K_i$ after f_{ij}. In this case, $|K_jN|$ can be determined without any computation.

CHAPTER 3

OPTIMIZATION MODEL

In this chapter, we present a mathematical model for distributed query optimization. The cost reduction model of a sequence of semijoins is developed in terms of the cost and benefit associated with each member semijoin of the sequence. The optimization model is formulated using the cost reduction model and the database state transition model presented in Chapter 2.

3.1 Cost Reduction Model

In this section, the computation of the processing cost of a query is discussed. The expressions of the benefit achieved by a semijoin are derived for a few different cases depending on the change of the value of INFO by the semijoin. The net benefit of a sequence of semijoins is expressed in formulae.

Our cost measure of the distributed query is the total inter-site data flow transferring the necessary data to the user site. As mentioned in Chapter 1, the transmission delay between source and destination in packet switching networks is proportional to the volume of data being transmitted. We also include the message overhead incurred

by inter-site flow to the data traffic. Here, the term "message" is used not to designate a specific number of packets but a continuous stream of inter-site data flow.

A message overhead is defined in [KLEI 76] as all those characters that are transmitted but not exchanged between user processes in the attached host computers. The message overhead can be approximately divided into the following two parts which are measured in bytes:

(1) V_f: the fixed portion of the message overhead,

(2) V_p: the portion of the message overhead which is proportional to the length of the message.

The transmission cost to transfer the message of length M bytes is given by:

$$\begin{aligned} C(M) &= V_f + V_p + M \\ &= V_f + (1 + V_p/M) \times M \\ &= V_f + \upsilon \times M \end{aligned} \qquad (3.1)$$

where υ denotes the proportional coefficient and it is given by $\upsilon = 1 + V_p/M$. V_f and υ are the parameters determined by the communication network being used. By including the message overhead in the transmission cost function, we want to include the effect of the length of the sequence of semijoins in minimizing the total amount of data transferred in processing a query. In other words, any semijoin which has a negligible effect in reducing the amount of data transfer is excluded from the sequence to avoid the message

overhead.

After the initial local processing, there are one or more relations of R^+ at each site involved in the query. When the reduced relations are finally transferred to the user site, all the relations at one site can be sent by a single message. Consequently the message overhead to move the relations depends on the number of sites. The relations stored at the user site do not have to be moved. Hence we have to make use of the site information in formulating the query cost.

R^+ is partitioned into blocks such that any two relations in the same block are stored at the same site. This partition is called SITE. If there are m sites involved in the query,

$$SITE = \{ST_1, ST_2, \ldots, ST_m\}$$

The block ST_i can be considered a set of relations stored at site i.

Let ST_R^i be $ST_j \in SITE$ such that $R_i \in ST_j$ for $R_i \in R^+$ and let ST_U be the set of relations stored at the user site. If $ST_U \neq \emptyset$, $ST_U = ST_j$ for some $ST_j \in SITE$. The data distribution information is described by a set of parameters called DIST:

$$DIST = < ST_R, ST_U >$$

where $ST_R = \{ST_R^i \mid R_i \in R^+\}$.

The following example shows the derivation of the initial value of DIST from SITE and ST_U.

Example 3.1

Consider the relations presented in Example 2.1. It was assumed that the relations are all stored at different sites. Let R_4 = OFFER be the relation at the user site. Since there are five sites involved in the query,

$$SITE = \{ST_1, ST_2, ST_3, ST_4, ST_5\}$$

where $ST_i = \{R_i\}$ for i = 1, 2, 3, 4, 5.

Since $ST_U = \{R_4\}$, $ST_U = ST_4$.

In this case, the initial value of ST_R for data distribution information

$$DIST = < ST_R, ST_U >$$

is given by $ST_R^i = \{R_i\}$ for i = 1, 2, 3, 4, 5. ■

The data distribution information together with the state of the database gives a complete description of the distributed database to which the user query is issued. We define the state S of the distributed database as follows:

$$S = < INFO, PAR, DIST >$$

The state space is denoted by Σ.

The initial cost of a query is the total communication cost to retrieve all the relations in R^+ to the user site after initial local processing without using any semijoin. Let IC be the initial cost of a query and S(R) the size of a relation R.

$$IC = V_f \times (|SITE| - t_U) + \upsilon \times \sum_{R_i \in R^+ - ST_U} S(R_i)$$

$$= V_f \times (|SITE| - t_U) + \upsilon \times \sum_{R_i \in R^+ - ST_U} (|R_i| \times \sum_{a_j \in \alpha_i} w_j) \qquad (3.2)$$

where $t_U = \begin{cases} 1 & \text{if } ST_U \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$

The value of INFO makes it necessary to differenciate a few cases in computing the benefit achieved by a semijoin. The change in the value of INFO is caused by SJR. In Chapter 2, INFO is defined as follows:

$$INFO = < N_R, N_B, JR, SJR, EJA, AJA, \Phi >$$

If $SJR \neq \emptyset$, the initial value of INFO is not changed throughout the query processing.

Consider $a_i, a_j \in B_h$ for some $B_h \in \Pi$. Suppose $\mu(a_i) = R_i$, $\mu(a_j) = R_j$ and $R_i \in SJR$. After f_{ij}, all the values of K_i necessary to process the query are included in K_jN. Since the information contained in R_i has been completely transferred to R_j, R_i can be ignored. The elimination of R_i from R^+ causes the following changes:

(1) $N_B = \{|B_k| \mid B_k \in \Pi\}$

- Since a_i is ignored, if $|B_h| > 2$, $|B_hN| = |B_h| - 1$.
- If $|B_h| = 2$, $EJA_j = \emptyset$ after a_i is ignored.
  Hence if $a_i, a_j \notin T$ then B_h is ignored and $|B_hN| = 0$.

(2) $N_R = \{|\alpha_k| \mid R_k \in R^+\}$

- $|\alpha_iN| = 0$, which implies R_i is ignored.
- If $|B_hN| = 0$, a_j can also be ignored.
  In this case, $|\alpha_jN| = |\alpha_j| - 1$.

(3) SJR

- $SJRN = SJR - \{R_i\}$.

- If $|\alpha_j N| = 1$ and $R_j \in JR$, then R_j becomes a singleton joining relation. Hence $SJRN = SJR - \{R_i\} \cup \{R_j\}$.

(4) JR

- $JRN = JR - \{R_i\}$.

(5) EJA

- $EJA_h N = EJA_h - \{a_i\}$ for all h such that $a_h \in EJA_i$.

(6) AJA

- If $|B_h N| = 0$, a_j will not participate in any semijoin. Hence $AJA_g N = AJA_g - \{a_j\}$ for all g such that $a_g \in AJA_j$.

(7) Φ

- If $|B_h N| \neq 0$, $\Phi N = \Phi - \{f_{ix}, f_{xi} \mid a_x \in EJA_i\}$.
  If $|B_h N| = 0$, $\Phi N = \Phi - \{f_{ij}, f_{ji}\}$.

(8) DIST

- $ST_R^i N = ST_R^i - \{R_i\}$.
  If $ST_R^i N = \emptyset$, ST_R^i can be ignored.

(9) $C_R = \{|R_k| \mid R_k \in R^+\}$

- If R_j becomes a singleton joining relation, $|R_j N| = |K_j N|$.

The following example shows the change in the value of state S caused by a semijoin.

Example 3.2

Assume the query information model Q in Example 2.1, the initial value of INFO in Example 2.2, and the initial value of DIST in Example 3.1.

Consider the first semijoin $\phi_1 = f_{16}$. This semijoin is to perform $R_4 \langle a_6 = a_1] R_1$. Here $\mu(a_1) = R_1$, $\mu(a_6) = R_4$, and $\psi(a_1) = \psi(a_6) = B_1$. Since $R_1 \in SJR$, the value of S is changed after ϕ_1 as follows:

(1) N_B

- Since $|B_1| = 2$ and $a_1, a_6 \notin T$, $|B_1N| = 0$.

(2) N_R

- $|\alpha_1 N| = 0$.
- Since $|B_1N| = 0$, $|\alpha_4 N| = |\alpha_4| - 1 = 1$.

(3) SJR

- Since $|\alpha_4 N| = 1$ and $R_4 \in JR$,
  $SJRN = SJR - \{R_1\} \cup \{R_4\} = \{R_4\}$.

(4) JR

- $JRN = JR - \{R_1\} = \{R_4, R_5\}$.

(5) EJA

- $EJA_6N = EJA_6 - \{a_1\} = \phi$.

(6) AJA

- Since $|B_1N| = 0$, $AJA_7N = AJA_7 - \{a_6\} = \phi$.

(7) Φ

- Since $|B_1N| = 0$, $\Phi N = \Phi - \{f_{16}, f_{61}\}$
  $= \{f_{27}, f_{72}, f_{28}, f_{82}, f_{78}, f_{87}, f_{49}, f_{94}\}$.

(8) DIST

- $ST_R^1N = ST_R^1 - \{R_1\} = \phi$.

(9) C_R

- Since R_4 becomes a singleton joining relation,

$|R_4N| = |K_6N|$ ■

We derive expressions for the net benefit of a semijoin by using the parameters describing the database state. Consider f_{ij} with $\mu(a_i) = R_i$, $\mu(a_j) = R_j$ and $a_i, a_j \in B_h$ at any point during the query processing. Let n_{ij} be the net benefit of f_{ij}, c_{ij} the cost incurred by f_{ij} and b_{ij} the benefit achieved by f_{ij}.

The cost c_{ij} is the communication cost to transfer the current set of values of a_i.

$$c_{ij} = V_f + \upsilon \times (|K_i| \times w_i) \tag{3.3}$$

The benefit b_{ij} is classified into the following three different parts:

1. b_{ij}^1: The benefit due to the reduction of $|R_j|$ which results from the reduction of $|K_j|$.

$$b_{ij}^1 = t_{R_j}^U \times [\upsilon \times (|R_j| - |R_jN|) \times \textstyle\sum_{a_h \in \alpha_j} w_h] \tag{3.4}$$

where $t_R^U = \begin{cases} 1 & \text{if } R \notin ST_U, \\ 0 & \text{otherwise.} \end{cases}$

2. b_{ij}^2: The benefit due to the elimination of R_i.

If $R_i \notin ST_U$ and $R_i \in SJR$, the benefit is:

$$\upsilon \times S(R_i) = \upsilon \times (|K_i| \times w_i)$$

Furthermore, if $ST_R^iN = \emptyset$, V_f is eliminated.

$$b^2_{ij} = t^U_{R_i} \times t^S_{R_i} \times [V_f \times t^L_{R_i} + \upsilon \times (|K_i| \times w_i)] \qquad (3.5)$$

where $t^S_R = \begin{cases} 1 & \text{if } R \in SJR, \\ 0 & \text{otherwise;} \end{cases}$

and $t^L_{R_i} = \begin{cases} 1 & \text{if } ST^i_R N = \emptyset, \\ 0 & \text{otherwise.} \end{cases}$

3. b^3_{ij}: The benefit due to the elimination of a_j.

If $R_j \notin ST_U$ and $|B_h N| = 0$, the benefit is:

$$\upsilon \times |R_j N| \times w_j$$

In addition, if $|\alpha_j N| = 1$ and $a_k \in J$ where $\mu(a_k) = R_j$, the additional benefit can be given as follows:

$$\upsilon \times (|R_j N| - |K_k N|) \times w_k$$

$$b^3_{ij} = t^U_{R_j} \times t_{B_h} \times \upsilon \times [|R_j N| \times w_j + t_{\alpha_j} \times t_{a_k} \times (|R_j N| - |K_k N|) \times w_k] \qquad (3.6)$$

where $t_{B_h} = \begin{cases} 1 & \text{if } |B_h N| = 0, \\ 0 & \text{otherwise;} \end{cases}$

and $t_{\alpha_j} = \begin{cases} 1 & \text{if } |\alpha_j N| = 1, \\ 0 & \text{otherwise;} \end{cases}$

and $t_a = \begin{cases} 1 & \text{if } a \in J, \\ 0 & \text{otherwise.} \end{cases}$

From (3.4), (3.5) and (3.6),

$$b_{ij} = b^1_{ij} + b^2_{ij} + b^3_{ij} \qquad (3.7)$$

Using (3.3) and (3.7),

$$n_{ij} = b_{ij} - c_{ij} \tag{3.8}$$

We now compute the cost of a query associated with a sequence of semijoins. Consider the following sequence.

$$SEQ = \phi_1, \phi_2, \ldots, \phi_m$$

where $\phi_i \in \Phi$ for $i = 1, 2, \ldots, m$. Let n_i be the net benefit of ϕ_i, c_i the cost incurred by ϕ_i, b_i the benefit achieved by ϕ_i, and SC the cost of a query associated with SEQ.

$$\begin{aligned} SC &= IC + \sum_{i=1}^{m} c_i - \sum_{i=1}^{m} b_i \\ &= IC - \sum_{i=1}^{m} (b_i - c_i) \\ &= IC - \sum_{i=1}^{m} n_i \end{aligned} \tag{3.9}$$

$IC - \sum_{i=1}^{m} b_i$ corresponds to the cost of retrieving the remaining portions of the relations to the user site and $\sum_{i=1}^{m} c_i$ corresponds to the cost of semijoins incurred by SEQ. The cost reduced by SEQ is $\sum_{i=1}^{m} n_i$ which is the net benefit of SEQ.

3.2 Problem Formulation

In this section, we formulate the optimization problem of distributed query processing. The model for database state transition was presented in Chapter 2. The expressions for the cost reduction achieved by a sequence of semijoins were derived in Section 3.1 in terms of database state parameters. The optimization model is based on the database state transition model and the cost reduction

model.

The optimization problem of distributed query processing is described as follows:

Given:

- Query information model Q of the user query reduced by the initial local processing.

$$Q = < T, J, R^+, \Pi, \mu, \psi >$$

where T is the target list of the user query,
J is the set of joining attributes of the user query,
R^+ is the set of relations referenced by the user query,
Π is the partition of J induced by the equivalence relation '=',
μ: (T U J) --> R^+ specifies the relation to which each attribute belongs,
and ψ: J --> Π specifies the block of Π to which each joining attribute belongs.

- The value of PAR after initial local processing.

$$PAR = <\{|R_i| \mid R_i \in R^+\}, \{|D_i| \mid B_i \in \Pi\}, \{|K_i| \mid a_i \in J\}, \{w_i \mid a_i \in T \cup J\}>$$

where $|R_i|$ is the cardinality of tuples in relation R_i,
$|D_i|$ is the cardinality of values in the domain of the attributes in block B_i,
$|K_i|$ is the cardinality of the current set of values of joining attribute a_i,
and w_i is the width (in bytes) of attribute

$a_i \in T \cup J$.

- The data distribution information SITE and ST_U.

$$SITE = \{ST_1, ST_2, \ldots, ST_m\}$$

where m is the number of sites involved in the user query

and ST_i is the set of relations stored at the ith site.

ST_U is the set of relations stored at the user site.

Determine:

A sequence of semijoins $\phi_1, \phi_2, \ldots, \phi_w$

which has the minimal cost of the user query

$$IC - \sum_{i=1}^{w} n_i$$

where $\phi_i \in \Phi$, the set of possible semijoins to process the user query,

IC is the initial cost of the user query given by (3.2),

and n_i is the net benefit of ϕ_i given by (3.8). ■

Since IC is fixed, minimizing $IC - \sum_{i=1}^{w} n_i$ is equivalent to maximizing $\sum_{i=1}^{w} n_i$.

The initial state of the distributed database before using any semijoin is known:

- The initial value of INFO is derived from Q.
- The initial value of DIST is derived from SITE and ST_U.
- The initial value of PAR is given.

The subsequent change in the value of the distributed database state S caused by a sequence of semijoins can be

computed by using the methods presented in Chapter 2 and Section 3.1. The state transition model for the distributed database is considered a function which determines the next state given the current state and a semijoin.

The net benefit of a semijoin at any position in a sequence can be computed using the expressions derived in Section 3.1. The cost reduction model is considered a function which determines the net benefit of a semijoin given the current state and the next state of the distributed database and a semijoin.

Consider the ith semijoin ϕ_i.

Let S_i be the state of distributed database before ϕ_i,

S_{i+1} be the state of distributed database after ϕ_i,

and REAL be the set of real numbers.

Then S_{i+1} is a function of S_i and ϕ_i, and this definition can be expressed as follows:

$$S_{i+1} \stackrel{d}{=} \bar{s}(S_i, \phi_i) \tag{3.10}$$

where $\bar{s}$: Σ X Φ --> Σ is defined to be the state transition function. Further, n_i is a function c' of S_i, S_{i+1} and ϕ_i, which can be defined as follows:

$$\begin{aligned} n_i &\stackrel{d}{=} c'(S_i, S_{i+1}, \phi_i) \\ &= c'(S_i, \bar{s}(S_i, \phi_i), \phi_i) \\ &\stackrel{d}{=} \bar{c}(S_i, \phi_i) \end{aligned} \tag{3.11}$$

where $\bar{c}$: Σ X Φ --> REAL is defined to be the cost reduction

function.

Now, the optimization model can be described as follows:

Let Γ be a set of sequences of semijoins and let $\gamma \in \Gamma$ where $\gamma = \phi_1, \phi_2, \cdots, \phi_{\lambda(\gamma)}$.

The optimal sequence of semijoins can be obtained by solving the following:

$$\begin{aligned}
&\text{maximize} && \textstyle\sum_{i=1}^{\lambda(\gamma)} n_i \\
&\text{subject to} && S_1 \text{ is given,} \\
& && S_{i+1} = \bar{s}(S_i, \phi_i) && (i = 1, \ldots, \lambda(\gamma)), \\
& && n_i = \bar{c}(S_i, \phi_i) && (i = 1, \ldots, \lambda(\gamma)), \\
&\text{and} && \phi_i \in \Phi && (i = 1, \ldots, \lambda(\gamma)).
\end{aligned}$$

CHAPTER 4

DOMINANT TERM OPTIMIZATION

The purpose of this chapter is to develop efficient methods for computing the values of variables which need to be evaluated frequently in the query optimization procedure.

There are two important standards in evaluating a computer algorithm designed for an optimization problem. One is the optimality of the solution produced by the algorithm. The other is the efficiency of the algorithm itself. The execution time of an algorithm is determined by the product of the number of executions of the dominant term and the time required to execute the dominant term.

In solving our problem, we have to frequently compute the net benefits of semijoins. Hence it can be considered the dominant term in our case to find out S_{i+1} and n_i given S_i and ϕ_i. Especially efficient computations of the cardinalities of the sets of values of joining attributes reduced by semijoins are crucial in reducing the time required to execute the dominant term. In Chapter 2, we have developed a lattice model and a method which systematically generates the lattice in order to characterize the properties and the structure of the set of researchable sets for a block of joining attributes. The

efficiency in application, however, was not considered. Also we have to develop an efficient approximation for (2.6) which requires long computation time for a large value of k.

The dominant term optimization is important because it is common whether we develop an optimal algorithm or a heuristic algorithm for the query optimization.

4.1 Efficient Use of the Lattice Model

In this section, we present an efficient algorithm to compute the cardinality of the set of values of a joining attribute reduced by a semijoin from its equivalent joining attribute.

Since the query optimization will be done using a computer, all the operations involved in query processing have to be developed with computer implementations in mind.

First, we discuss the operations in the lattice. Consider a block $B = \{a_1, a_2, \ldots, a_n\}$ and the current set of all reachable sets RS for B. We have to identify l.u.b.$\{K_i, K_j\}$ and g.l.b.$\{K_i, K_j\}$ to apply Theorem 2.4 and Equation (2.7) in estimating the effect of f_{ij} or f_{ji} for $a_i, a_j \in B$.

A natural way of implementing a lattice in a computer is to use doubly linked lists. A node represents an element and pointers are used to specify the covering relationship among elements. However, searching for g.l.b. and l.u.b. by following pointers takes a long time apart from the cost of space.

In our case, the special structure of (RS, $\underline{c}$) and the naming rule of its elements can be used to implement efficient lattice operations. Suppose (RS, $\underline{c}$) contains m expanded sublattices. Let I be an index set {1, 2, ..., n, n+1, ..., n+m}. Since each reachable set in RS is the intersection of elements from the set $\{A_1, A_2, \ldots, A_n, A_{n+1}, \ldots, A_{n+m}\}$, for any X,Y $\in$ RS

$$X = \Omega_{i \in I_x} A_i \quad \text{for some } I_x \underline{c} I \tag{4.1.a}$$

$$\text{and } Y = \Omega_{j \in I_y} A_j \quad \text{for some } I_y \underline{c} I \tag{4.1.b}$$

Furthermore, from the naming rule of the elements in RS, X > Y if and only if I_x c I_y. Note that X = D if and only if $I_x = \emptyset$.

Hence,

$$\text{g.l.b.}\{X, Y\} = XY$$

$$= (\Omega_{i \in I_x} A_i) \; \Omega \; (\Omega_{j \in I_y} A_j)$$

$$= \Omega_{k \in I_z} A_k \tag{4.2.a}$$

where $I_z = I_x \cup I_y$

Similarly,

$$\text{l.u.b.}\{X, Y\} = \text{minimum}\{Z \in RS \mid X \underline{c} Z \text{ and } Y \underline{c} Z\}$$

$$= \Omega_{h \in I_w} A_h \tag{4.2.b}$$

where $I_w = I_x I_y$

In this way, we can identify l.u.b.$\{K_i, K_j\}$ and g.l.b.$\{K_i, K_j\}$ without storing the whole lattice and searching it.

The lattice model provides the theoretical basis for computing the state transition of the database. There may be many different algorithm designs based on the results presented in Chapter 2. Our goal is to find an algorithm which reduces the sum of the time for computing cardinalities and the time for maintaining information necessary for computation.

In order to use (2.7), we have not only to identify l.u.b.$\{K_i, K_j\}$ but also to know $|$l.u.b.$\{K_i, K_j\}|$. Since A_1, A_2, ..., A_n are the sets from which all other reachable sets are reached, their cardinalities have to be available. Suppose l.u.b.$\{K_i, K_j\} \notin \{D, A_1, A_2, \ldots, A_n\}$. In many cases for $n < 4$, l.u.b.$\{K_i, K_j\}$ is the reduced set by an earlier semijoin in the sequence. That is, $|$l.u.b.$\{K_i, K_j\}|$ has been computed previously by using (2.6) or (2.7). If this is always the case, we have only to store the cardinalities of A_1, A_2, ..., A_n and the reduced sets by the previous semijoins.

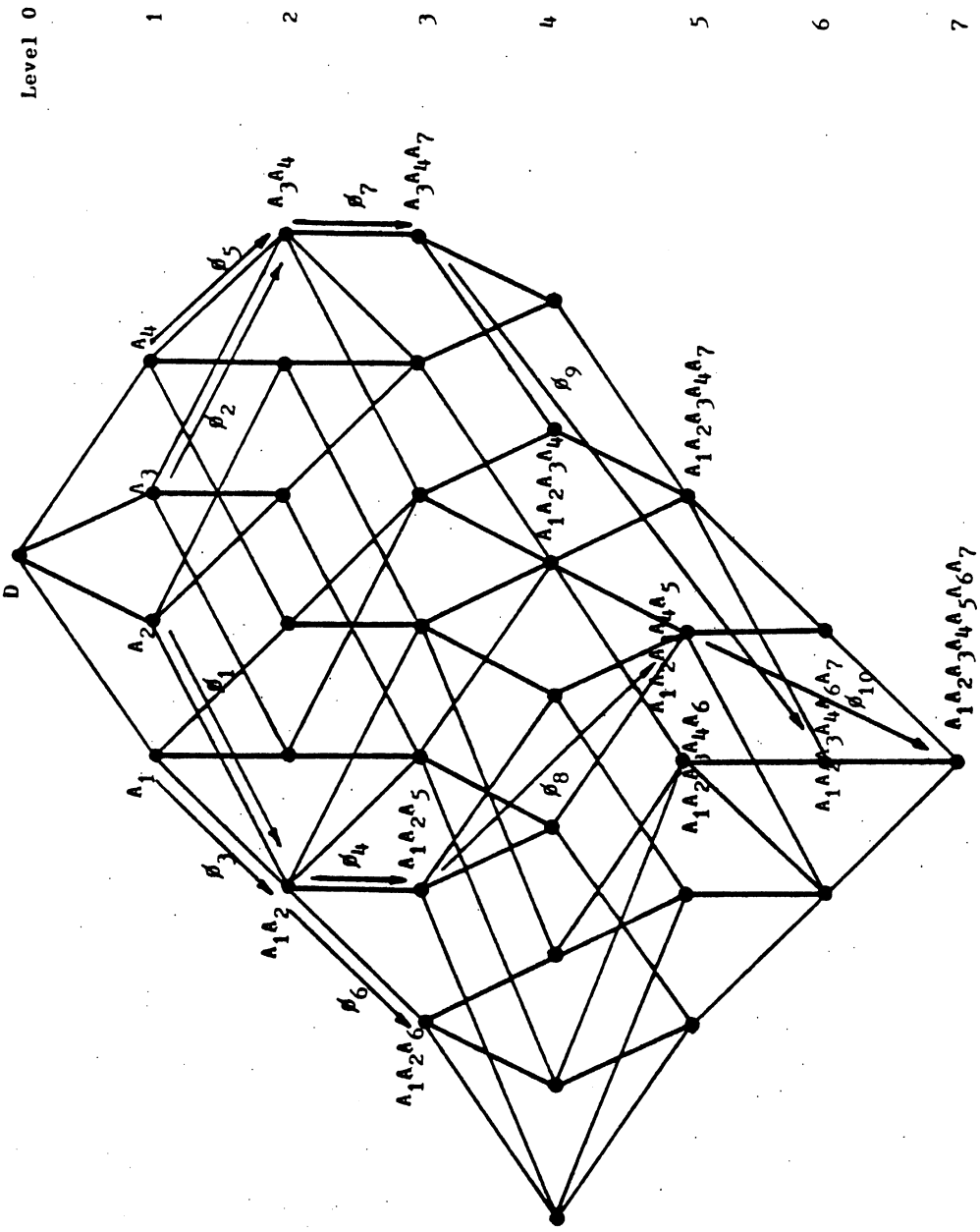
The following counter example shows that this conjecture is not true.

Example 4.1

In this example, we construct a sequence of semijoins such that for a semijoin in the sequence,

(1) l.u.b.$\{K_i, K_j\} \notin \{D, A_1, \ldots, A_n\}$

(2) l.u.b.$\{K_i, K_j\}$ is not a set reduced by any previous semijoin.



The Hasse Diagram of the Expanded Lattice and the Effect of Each Semijoin for Example 4.1

Figure 4.1

We use the expanded lattice generated in Example 2.5:

Let $B_1 = \{a_1, a_2, a_3, a_4\}$,

$B_2 = \{a_g, a_h\}$,

$B_3 = \{a_i, a_j\}$,

$B_4 = \{a_k, a_p, a_q\}$,

and $a_g \in AJA_1$, $a_q \in AJA_2$, $a_i \in AJA_4$.

Consider a sequence of semijoins:

$f_{12}, f_{43}, f_{21}, f_{hg}, f_{34}, f_{kn}, f_{ji}, f_{31}, f_{24}, f_{41}$

The expanded lattice and the effects of $\phi_1 = f_{12}$, ..., $\phi_7 = f_{ji}$ are shown in Figure 2.5. After ϕ_7, $K_1 = A_1A_2A_5$, $K_2 = A_1A_2A_6$, $K_3 = A_3A_4$ and $K_4 = A_3A_4A_7$.

The effects of the rest of semijoins are:

(1) $\phi_8 = f_{31}$

- K_1 is reduced from $A_1A_2A_5$ to $A_1A_2A_3A_4A_5$.
- l.u.b.$(K_1, K_3) = D$

(2) $\phi_9 = f_{24}$

- K_4 is reduced from $A_3A_4A_7$ to $A_1A_2A_3A_4A_6A_7$.
- l.u.b.$(K_2, K_4) = D$

(3) $\phi_{10} = f_{41}$

- K_1 is reduced from $A_1A_2A_3A_4A_5$ to $A_1A_2A_3A_4A_5A_6A_7$.
- l.u.b.$(K_1, K_4) = A_1A_2A_3A_4$

For ϕ_{10}, l.u.b.$(K_1, K_4) = A_1A_2A_3A_4$ is not a set reduced by any semijoin in the sequence. The effect of each semijoin is shown in Figure 4.1. ■

Hence it is not sufficient to store the cardinalities of the reduced sets and those of D, A_1, ..., A_n in order to find $|\text{l.u.b.}\{K_i, K_j\}|$ by searching. If we store the

cardinalities of all the elements of RS, we can definitely find the cardinality of a reduced set by using Theorem 2.4.

If the entire RS is to be stored with the cardinality of each reachable set, the cardinalities can be computed using the following properties:

(1) For any L^I-type lattice, the events corresponding to the generators are mutually conditionally independent given the event corresponding to the greatest element.

(2) Whenever an L^I-type lattice is generated to make a new expanded lattice, the cardinalities of the greatest element and generators of the L^I-type lattice have been computed and stored.

Although the above method is a conceptually simple way of finding the cardinalities of reduced sets, it involves many complex procedures. The major time consuming components are as follows:

(1) Procedure GEN_LAT has to be run for each expanded sublattice to generate a new expanded lattice.

(2) The cardinalities of all the elements in RS have to be computed and stored.

(3) g.l.b.$\{K_i, K_j\}$ has to be searched to retrieve its cardinality.

Since the lattice (RS, $\underline{c}$) may grow very large depending on the sequence of semijoins, this method is not efficient. We must develop an algorithm which does not need to store the entire lattice and the cardinality of each reachable set.

Therefore, we present a recursive algorithm which computes the cardinality of any reachable set in the lattice without storing the lattice. Once $|1.u.b.\{K_i, K_j\}|$ is computed by this algorithm, the cardinality of the reduced set is obtained by (2.7).

Since the number of tuples in each referenced relation is finite and a semijoin has to reduce at least one tuple to process a query, a sequence of semijoins to process a query is necessarily finite. As a result, the lattice of reachable sets corresponding to such a sequence is finite. Even for an infinite sequence of semijoins, unless a semijoin reduces a tuple in a relation, an expanded sublattice will not be generated. Therefore the lattice of reachable sets is always finite. Since a finite lattice is a complete lattice, every subset in (RS, $\underline{c}$) has a g.l.b. and a l.u.b.

From the completeness of an expanded lattice and the associative property of g.l.b., we have the following general result:

Lemma 4.1: Let the lattice L = (RS, $\underline{c}$) for a block B = $\{a_1, a_2, \ldots, a_n\}$ contain n initial sets $A_1, A_2, \ldots, A_n$ and m sets $A_{n+1}, A_{n+2}, \ldots, A_{n+m}$ generated by semijoins between the attributes in other blocks. Let I = {1, 2, ..., n, n+1, ..., n+m}. Then for any set $X \in RS$ with the index set $I_X \underline{c} I$,

$$X = g.l.b.\{X_1, X_2, \ldots, X_p\}$$

for some $X_j \in RS$, $j \in J = \{1, 2, \ldots, p\}$, if and only if

$$I_x = U_{j \in J} I_j$$

where I_j is the index set of X_j for $j \in J$.

Proof: (->) Since Λ is associative, g.l.b.$\{X_1, \ldots, X_p\}$ = g.l.b.$\{(X_1 \Lambda X_2), X_3, \ldots, X_p\}$ = $(\ldots((X_1 \Lambda X_2) \Lambda X_3) \ldots) \Lambda X_p$. From (4.2.a), $X_1 \Lambda X_2 = \Omega_{i \in I_1 U I_2} A_i$. Following the same procedure, g.l.b.$\{X_1, \ldots, X_p\} = \Omega_{i \in I_1 U I_2 \ldots I_p} A_i$. Hence g.l.b.$\{X_1, \ldots, X_p\} = X$ implies $U_{j \in J} I_j = I_x$.
(<-) If $U_{j \in J} I_j = I_x$, then g.l.b.$\{X_1, \ldots, X_p\}$
$= \Omega_{i \in I_1 U I_2 \ldots I_p} A_i = \Omega_{i \in I_x} A_i = X$. ∎

Definition 4.1: In (RS, $\underline{c}$) for a block $\{a_1, a_2, \ldots, a_n\}$ with m expanded sublattices, let $X = \Omega_{i \in I_x} A_i$ for $I_x \underline{c} I = \{1, 2, \ldots, n, n+1, \ldots, n+m\}$. The <u>index set of a reachable set</u> X is I_x.

Since the index set of a reachable set uniquely determines the reachable set, we will represent reachable sets by their index sets. Suppose the greatest element of the kth expanded sublattice is Z. Then $A_{n+k} = ZA_{n+k}$. The index set of A_{n+k} is given, by convention, by $I_{A_{n+k}} = I_z \cup \{n+k\}$.

Using (4.1.a), we represent a reachable set by its index set in the algorithms to determine the cardinality of a reachable set. In computing the cardinality of a set $X \in L$, we determine sets $X_1, X_2, \ldots, X_p$ such that $X_1, X_2, \ldots, X_p \in L$, g.l.b.$\{X_1, X_2, \ldots, X_p\} = X$ and the cardinalities of $X_1, X_2, \ldots, X_p$ are easily obtainable. In fact, the following algorithm generates the index sets of $X_1, \ldots, X_p$

such that either $X_i \in \{A_{n+1}, A_{n+2}, \ldots, A_{n+m}\}$ or $X_i \in L^I$ for $i = 1, \ldots, p$. If $X_i \in \{A_{n+1}, A_{n+2}, \ldots, A_{n+m}\}$, $|X_i|$ can be computed by (2.6). If $X_i \in L^I$, $|X_i|$ can be computed using the fact that the events corresponding to $A_1, A_2, \ldots, A_n$ are mutually independent. The cardinality of the set X can then be computed by using the associativity of g.l.b. and repetitive application of (2.7).

```
PROCEDURE SET_COVER (I_x)
  // I_x is the index set of X ∈ RS.  I_Aq is the //
  // index set of A_q for q = n+1, ..., n+m //
  C <- ø      ; initialize the cover being constructed.
  u <- max I_x
  WHILE    u > n                    ; n: the size of block
  DO
    BEGIN
      C <- C U {I_Au}
      I_x <- I_x - I_Au
      IF       I_x = ø
      THEN     u <- 0
      ELSE     u <- maximum I_x
    END
  IF  u ≠ 0  THEN C <- C U {I_x}
  RETURN (C)
END SET_COVER
```

Let I_i and I_j be the index sets of X_i, X_j, respectively and $X_i, X_j \in RS^I \cup \{A_{n+1}, \ldots, A_{n+m}\}$. Since A_{n+k} is in the kth expanded sublattice, $A_{n+g} > A_{n+h}$ implies $g < h$. Hence $X_i > X_j$ implies max $I_i <$ max I_j. Consequently, if $I_i \subset I_j$ then max $I_i <$ max I_j. Suppose $X_i > X_j \geq X$ and there is no $X_k \in RS^I \cup \{A_{n+1}, \ldots, A_{n+m}\}$ such that max $I_j <$ max $I_k \leq$ max

I_x, where I_k denotes the index set of X_k. Procedure SET_COVER avoids producing I_i by producing I_j first.

The following example illustrates the steps in procedure SET_COVER.

Example 4.2

Consider the lattice shown in Figure 4.1. $RS^I = \{A_1, A_2, A_3, A_4\}$ and m = 3. Since $A_5 = A_1A_2A_5$, $A_6 = A_1A_2A_6$ and $A_7 = A_3A_4A_7$, the index sets are: $I_{A_5} = \{1, 2, 5\}$, $I_{A_6} = \{1, 2, 6\}$, and $I_{A_7} = \{3, 4, 7\}$. We apply SET_COVER for $X = A_1A_2A_3A_4A_5A_6A_7$. The index set $I_x = \{1, 2, 3, 4, 5, 6, 7\}$.

(1) Initially $C = \emptyset$ and $u = \max I_x = 7$.

(2) First execution of WHILE statement:

Since $u = 7 > 4 = n$,

- $C = \emptyset \cup \{I_{A_7}\} = \{I_{A_7}\}$
- $I_x = \{1, 2, 3, 4, 5, 6, 7\} - \{3, 4, 7\} = \{1, 2, 5, 6\}$
- Since $I_x \neq \emptyset$, $u = \max I_x = 6$.

(3) Second execution of WHILE statement:

Since $u = 6 > n$,

- $C = \{I_{A_7}\} \cup \{I_{A_6}\} = \{I_{A_7}, I_{A_6}\}$
- $I_x = \{1, 2, 5, 6\} - \{1, 2, 6\} = \{5\}$
- Since $I_x \neq \emptyset$, $u = \max I_x = 5$.

(4) Third execution of WHILE statement:

Since $u = 5 > n$,

- $C = \{I_{A_7}, I_{A_6}\} \cup \{I_{A_5}\} = \{I_{A_7}, I_{A_6}, I_{A_5}\}$
- $I_x = \{5\} - \{1, 2, 5\} = \emptyset$
- Since $I_x = \emptyset$, $u = 0$.

(5) Since $u = 0$, $C = \{I_{A_7}, I_{A_6}, I_{A_5}\}$.

From $C = \{I_{A_7}, I_{A_6}, I_{A_5}\}$, we have: $X_1 = A_3A_4A_7$, $X_2 = A_1A_2A_6$, $X_3 = A_1A_2A_5$. Clearly g.l.b.$\{A_3A_4A_7, A_1A_2A_6, A_1A_2A_5\} = A_1A_2A_3A_4A_5A_6A_7$ and $A_3A_4A_7, A_1A_2A_6, A_1A_2A_5 \in \{A_5, A_6, A_7\}$.■

Once the index sets of $X_1, \ldots, X_p \in RS^I \cup \{A_{n+1}, \ldots, A_{n+m}\}$ are determined by using SET_COVER such that g.l.b.$\{X_1, \ldots, X_p\} = X$, we can compute $|X|$ by applying (2.7) p-1 times using $X_1, \ldots, X_p$.

We present an algorithm which takes the index set of $X \in RS$ and computes the cardinality of X. Let $|B| = n$ and $(RS, \underline{c})$ contain m expanded sublattices.

```
PROCEDURE CAL_CARD (I_x)
  // I_x is the index set of X.  SET_COVER takes I_x and   //
  // returns {I_1, ..., I_p} corresponding to {X_1, ..., X_p}//
  IF    max I_x ≤ n
  THEN  CARD <- |D| Π_{i∈I_x} P(A_i)   ; P(A_i) = |A_i| / |D|
  ELSE
    BEGIN
      SET_COVER (I_x)
      FOR  i = 1 UNTIL p   DO k_i <- max I_i
      IF   k_1 ≤ n              ; suppose k_1 ≤ ... ≤ k_p
      THEN C_GLB <- |D| Π_{i∈I_1} P(A_i)
      ELSE C_GLB <- |A_{k_1}|
      FOR  i = 1  UNTIL p-1
      DO
        BEGIN
          LUB <- I_i I_{i+1}          ; LUB is the index
                                      ; set of X_i v X_{i+1}
          IF     LUB = ø
          THEN   C_LUB <- |D|
          ELSE   C_LUB <- CAL_CARD (LUB)
```

```
        C_GLB <- C_GLB x |A_{k_{i+1}}| / C_LUB    ; use (2.7)
        I_{i+1} <- I_i U I_{i+1}
      END
    CARD <- C_GLB                  ; CARD = |X|
  END
END CAL_CARD
```

The following examples show how to compute the cardinalities of arbitrary reachable sets by Procedure CAL_CARD.

Example 4.3

In this example, we compute the cardinality of the reduced set by $\phi_{10} = f_{41}$ in Example 4.1. For (RS, c) shown in Figure 4.1, let $|D| = 10000$, $|A_1| = 2500$, $|A_2| = 4000$, $|A_3| = 5000$, $|A_4| = 6000$, $|A_5| = 400$, $|A_6| = 600$, $|A_7| = 1000$. For $\phi_{10} = f_{41}$, $K_1 = A_1A_2A_3A_4A_5$ and $K_4 = A_1A_2A_3A_4A_6A_7$. From the given cardinalities, $|K_1| = 120$ and $|K_4| = 60$. To compute $|K_1 \wedge K_4| = |A_1A_2A_3A_4A_5A_6A_7|$, we have to know $|K_1 \vee K_4| = |A_1A_2A_3A_4|$.

CAL_CARD computes $|X| = |A_1A_2A_3A_4|$ by taking $I_x = \{1, 2, 3, 4\}$ as inputs. Since $\max I_x = 4 = n$,

$$\begin{aligned} CARD &= |D|\, P(A_1)P(A_2)P(A_3)P(A_4) \\ &= 10000 \times (1/4) \times (2/5) \times (1/2) \times (3/5) \\ &= 300 \\ &= |A_1A_2A_3A_4| \end{aligned}$$

By (2.7), $|K_1 \wedge K_4| = |K_1| \times |K_4| / 300 = 24$. ■

Example 4.4

Suppose the lattice shown in Figure 4.1 is a sublattice of (RS, $\subseteq$) and l.u.b.$\{K_i, K_j\} = A_1A_2A_3A_4A_5A_6A_7$. We compute $|A_1A_2A_3A_4A_5A_6A_7|$ using CAL_CARD.

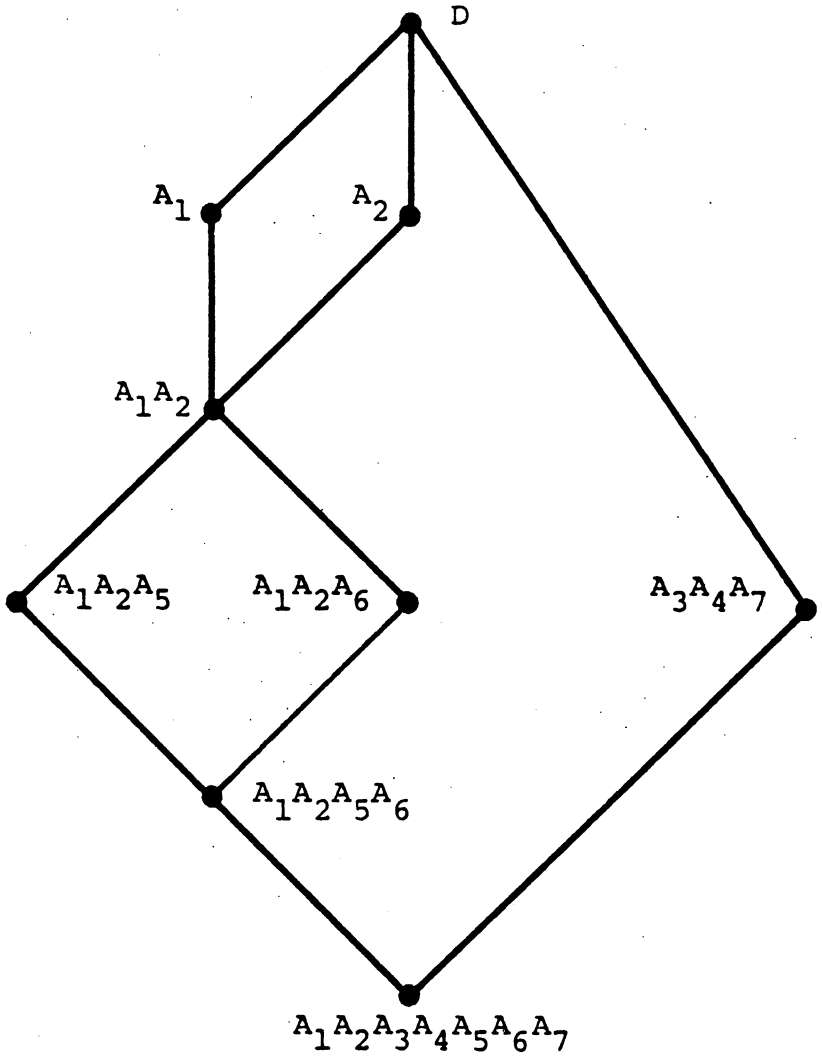
(1) Main procedure:

- Since $I_x = \{1, 2, 3, 4, 5, 6, 7\}$, max $I_x > n$.
- From Example 4.2, SET_COVER(I_x) returns $I_1 = \{1, 2, 5\}$, I_2 $\{1, 2, 6\}$ and $I_3 = \{3, 4, 7\}$.
- k_1 = max $I_1 = 5$, $k_2 = 6$, $k_3 = 7$.
- Since $k_1 > n$, C_GLB = $|A_5| = 400$.
- For i = 1, LUB = $I_1I_2 = \{1, 2\}$.
  Since LUB $\neq \emptyset$, C_LUB = CAL_CARD ($\{1, 2\}$).
  From the first recursion, C_LUB = 1000.
  C_GLB <- C_GLB $\times$ $|A_6|$ / C_LUB = 240.
  I_2 <- $I_1 \cup I_2 = \{1, 2, 5, 6\}$.
- For i = 2, LUB = $I_2I_3 = \emptyset$.
  Since LUB = $\emptyset$, C_LUB = $|D| = 10000$.
  C_GLB <- C_GLB $\times$ $|A_7|$ / $|D| = 24$.
  I_3 <- $I_2 \cup I_3 = \{1, 2, 3, 4, 5, 6, 7\}$.
- $|X|$ = C_GLB = 24.

(2) First recursion:

- Since max I_x = max $\{1, 2\} < n$,
  CARD = $|D|P(A_1)P(A_2) = 1000$.
- $|X| = 1000$.

The cardinality of $X = A_1A_2A_3A_4A_5A_6A_7$ is identical to that obtained in Example 4.3. In this rather complicated example, only one extra recursion is necessary. The reachable sets used in this example form a sublattice of the



The Hasse Diagram of the Poset
of Reachable Sets Used in Example 4.4

Figure 4.2

lattice shown in Figure 4.1. This sublattice is depicted in Figure 4.2. ■

In order to use CAL_CARD, we have only to store the information of D, A_1, ..., A_n, A_{n+1}, ..., A_{n+m} instead of the whole lattice. Since n is usually small and it is rare that a reachable set becomes the greatest element of more than one expanded sublattice, CAL_CARD generally terminates after a few recursions.

4.2 Efficient Appoximation of Equation (2.6)

In this section, we derive an efficient approximate formula to compute the cardinality of the set of values of a joining attribute reduced by a semijoin to an attribute associated with it. Then we validate the accuracy of the approximate formula by providing data from computer simulation.

Suppose a_h and a_j are the attributes of R_g. The effect of f_{ij} on $|K_h|$ is determined using (2.6), as presented in Chapter 2, as follows:

$$|K_hN| = m \times [1 - \Pi_{i=1}^{k}\{(n \times d - i + 1)/(n - i + 1)\}] \qquad (2.6)$$

where $m = |K_h|$, $k = |R_gN|$, $n = |R_g|$ and $d = 1 - 1/m$

Since k is the number of tuples belonging to the reduced relation after a semijoin, k may be very large. In this case, k iterations required in (2.6) take a long computation time. We must eliminate the term involving the

iteration from (2.6) while preserving sufficient accuracy.

In the literature, the following piecewise linear approximation of (2.6) was suggested in [CCA 80b]:

$$|K_h N| = \begin{cases} k & \text{for } k < m/2 \\ (k+m) / 3 & \text{for } m/2 \le k < 2m \\ m & \text{for } 2m \le k \end{cases} \qquad (4.3)$$

The above approximation generally produces large error because of the discontinuity of the formula. Especially, the amount of error is prohibitive near $k = m/2$ and $k = 2m$.

We approximate (2.6) from its derivation procedure. It was shown in [YAO 77] that

$$\Pi_{i=1}^{k} \{(nd - i + 1)/(n - i + 1)\} = C_k^{nd} / C_k^n$$

where C_r^n denotes ${}_nC_r$. Since

$$C_k^{nd} / C_k^n = \{(n-n/m)!(n-k)!\}/\{n!(n-n/m-k)!\}$$

we have

$$\begin{aligned} C_k^{nd} / C_k^n &= \{(n-k)!/(n-n/m-k)!\}\{(n-n/m)!/n!\} \\ &\simeq \{(n-k) / n\}^{n/m} \\ &= (1 - k/n)^{n/m} \end{aligned} \qquad (4.4.a)$$

and

$$\begin{aligned} C_k^{nd} / C_k^n &= \{(n-n/m)!/(n-n/m-k)!\}\{(n-k)!/n!\} \\ &\simeq \{(n-n/m) / n\}^{k} \end{aligned}$$

$$= (1 - 1/m)^k. \qquad (4.4.b)$$

From (4.4.a)

$$|K_h N| = m \times (1 - C_k^{nd}/C_k^n)$$

$$\simeq m \times (1 - (1 - k/n)^{n/m}). \qquad (4.5.a)$$

From (4.4.b),

$$|K_h N| = m \times (1 - C_k^{nd}/C_k^n)$$

$$\simeq m \times (1 - (1 - 1/m)^k). \qquad (4.5.b)$$

By taking the smaller exponent in order to decrease the error from approximation, we obtain the following approximate formula:

$$|K_h N| = \begin{cases} m \times (1 - (1 - k/n)^{n/m}) & \text{if } n/m < k, \\ m \times (1 - (1 - 1/m)^k) & \text{otherwise.} \end{cases} \qquad (4.6)$$

As pointed out in [YAO 77], it is obvious that $|K_h N| = m$ if $k > n - n/m$ or $m = 1$. In summary, we can compute $|K_h N|$ as follows:

$$|K_h N| = \begin{cases} k & \text{if } m = n \\ 1 & \text{if } m = 1 \\ m & \text{if } 1<m<n \text{ and } k>n-n/m \\ m(1-(1-k/n)^{n/m}) & \text{if } 1<m<n \text{ and } n/m<k\leq n-n/m \\ m(1-(1-1/m)^k) & \text{if } 1<m<n \text{ and } k\leq \min(n-n/m,\ n/m) \end{cases} \qquad (4.7)$$

We have performed the computer simulation of (2.6), (4.3), (4.5.a) and (4.5.b) for n = 50, 100, 1000, 10000,

100000 and various values of m and k for each n. The result of the simulation shows that the errer generated by (4.6) is practically negligible and that the choice made between (4.5.a) and (4.5.b) depending on the values of k and n/m is always correct.

Some of the data obtained from the simulation are presented in Tables 4.1 through 4.4. Table 4.1 shows the values of $|K_h N|$ for n = 100 and m = 30. Note that (4.6) is still effective when n/m is not an integer. Table 4.3 shows the data for n = 10000 and m = 100. Since n/m = 100, (4.5.a) and (4.5.b) produce the same results for k = 100. For k < 100, (4.5.b) is more accurate, whereas (4.5.a) is more accurate for k > 100. The plots of %error incurred by using (4.6) vs. k for n = 1000, m = 500 and for n = 10000, m = 2000 are shown in Figure 4.3.

Table 4.1

Comparison of Approximations of Equation (2.6)
for n = 100 and m = 30

k	(2.6):YAO	(4.5.a)	(4.5.b)	(4.3):CCA
3	2.93	2.90	2.90	3.00
6	5.65	5.59	5.52	6.00
9	8.18	8.09	7.89	9.00
12	10.51	10.41	10.03	12.00
15	12.67	12.55	11.96	15.00
18	14.65	14.52	13.70	16.00
21	16.47	16.33	15.28	17.00
24	18.13	17.98	16.70	18.00
27	19.65	19.49	17.99	19.00
30	21.02	20.86	19.15	20.00
33	22.26	22.10	20.20	21.00
36	23.37	23.22	21.15	22.00
39	24.37	24.22	22.00	23.00
42	25.26	25.12	22.78	24.00
45	26.04	25.91	23.48	25.00
48	26.73	26.61	24.11	26.00
51	27.33	27.22	24.68	27.00
54	27.85	27.75	25.19	28.00
57	28.29	28.20	25.66	29.00
60	28.67	28.59	26.08	30.00
75	29.74	29.70	27.64	30.00
90	29.99	29.99	28.58	30.00

Table 4.2

Comparison of Approximations of Equation (2.6)
for n = 1000 and m = 500

k	(2.6):YAO	(4.5.a)	(4.5.b)	(4.3):CCA
50	48.78	48.75	47.63	50.00
100	95.05	95.00	90.72	100.00
150	138.82	138.75	129.70	150.00
200	180.09	180.00	164.97	200.00
250	218.86	218.75	196.89	250.00
300	255.12	255.00	255.76	266.67
350	288.89	288.75	251.88	283.33
400	320.14	320.00	275.51	300.00
450	348.89	348.75	296.90	316.67
500	375.14	375.00	316.24	333.33
550	398.89	398.75	333.75	350.00
600	420.13	420.00	349.58	366.67
650	438.87	438.75	363.91	383.33
700	455.11	455.00	376.87	400.00
750	468.85	468.75	388.60	416.67
800	480.08	480.00	399.21	433.33
850	488.82	488.75	408.81	450.00
900	495.05	495.00	417.50	466.67
950	498.77	498.75	425.36	483.33

Table 4.3

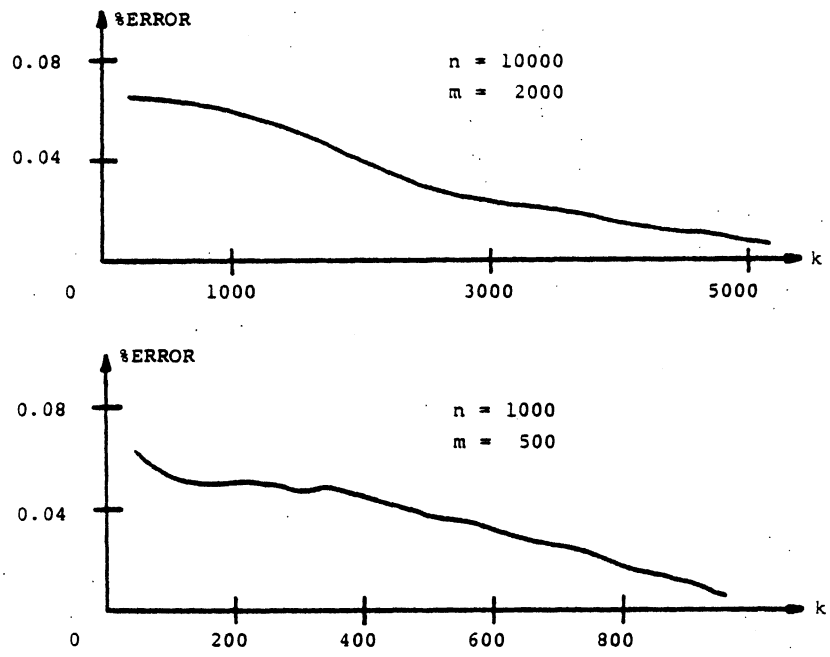
Comparison of Approximations of Equation (2.6) for n = 10000 and m = 100

k	(2.6):YAO	(4.5.a)	(4.5.b)	(4.3):CCA
10	9.57	9.52	9.56	10.00
20	18.23	18.14	18.21	20.00
30	26.06	25.95	26.03	30.00
40	33.16	33.02	33.10	40.00
50	39.58	39.42	39.50	50.00
60	45.38	45.22	45.28	53.33
70	50.64	50.46	50.52	56.67
80	55.39	55.21	55.25	60.00
90	59.69	59.51	59.53	63.33
100	63.58	64.00	64.00	66.67
110	67.10	66.92	66.90	70.00
120	70.28	70.10	70.06	73.33
130	73.16	72.98	72.92	76.67
140	75.76	75.58	75.51	80.00
150	78.11	77.94	77.85	83.33
160	80.23	80.07	79.97	86.67
180	83.89	83.74	83.62	93.33
200	86.87	86.74	86.60	100.00
300	95.32	95.24	95.10	100.00
500	99.42	99.41	99.34	100.00

Table 4.4

Comparison of Approximations of Equation (2.6)
for n = 10000 and m = 2000

k	(2.6):YAO	(4.5.a)	(4.5.b)	(4.3):CCA
400	369.49	369.25	362.61	400.00
1000	819.51	819.02	787.07	1000.00
1600	1164.16	1163.58	1101.50	1200.00
2200	1423.07	1422.57	1334.42	1400.00
2800	1613.44	1613.02	1506.96	1600.00
3400	1749.89	1749.53	1634.77	1800.00
4000	1844.77	1844.48	1729.45	2000.00
4600	1908.39	1908.17	1799.58	2000.00
5200	1949.18	1949.04	1851.54	2000.00
5800	1973.94	1973.86	1890.02	2000.00
8000	1999.36	1999.36	1963.40	2000.00



The Plots of %Error Incurred by Using Equation (4.6) vs. k

Figure 4.3

CHAPTER 5

AN OPTIMIZATION ALGORITHM

An optimization model for distributed query processing was described in Chapter 3, and a method to estimate intermediate results of a query was developed in Chapter 4 based on the lattice model in Chapter 2. In this chapter, an algorithm which solves the optimization model will be presented.

5.1 Complexity Consideration of Optimal Algorithms

In this section, the computational complexity involved in finding an optimal solution will be discussed.

It has been proven [HEVN 79a] that the distributed query optimization problem is NP-hard. Furthermore, it is unlikely that any of the existing optimization techniques can be used to solve the model because of the complexity of the state transition function $\bar{s}$ and the unconstraint of the final state in the optimization model.

We will examine the sizes of the solution spaces for distributed queries to measure the amount of computational effort when exhaustive searches are made to determine optimal solutions. Since it is not always possible to compute the size of the solution space of a distributed

query, a few different cases will be considered separately to take advantage of the structure of the query. In all cases, the number of relations referenced by a query is n.

CASE 1

There is one domain on which all joining attributes are defined. n relations referenced by a query are all singleton joining relations.

Since a singleton joining relation is ignored after an outgoing semijoin from the relation, the maximum length of a sequence of semijoins is n-1. The number of possible semijoins is ${}_nP_2$, where ${}_nP_r$ denotes the number of permutations of n elements, taken r at a time. Hence, the number of sequences of semijoins with length λ are as follows:

$\lambda=0$: 1

$\lambda=1$: ${}_nP_2$

$\lambda=2$: ${}_nP_2 \times {}_{n-1}P_2$

.

.

.

$\lambda=k$: ${}_nP_2 \times {}_{n-1}P_2 \times \cdots \times {}_{n-k+1}P_2$

The total number of sequences of semijoins involved in an exhaustive search is given by

$$1 + \sum_{\lambda=1}^{n-1} \left(\prod_{k=0}^{\lambda-1} {}_{n-k}P_2 \right) \qquad (5.1)$$

The value of (5.1) for n = 2, ..., 6 is given in Table

5.1.

Table 5.1

The Size of the Solution Space for CASE 1

n	Number of Sequences
2	3
3	19
4	229
5	4581
6	137431

CASE 2

There is one domain on which all the joining attributes are defined. Each of the n relations includes target lists.

The number of possible semijoins is ${}_nP_2$, which is n^2-n. Since each semijoin can appear only once in a sequence of semijoins, the maximum length of a sequence of semijoins is ${}_nP_2$. Hence the total number of sequences of semijoins involved in an exhaustive search in this case is given by

$$1 + \sum_{\lambda=1}^{n^2-n} {}_{n^2-n}P_\lambda \qquad (5.2)$$

The value of (5.2) for n = 2,3,4 is given in Table 5.2.

CASE 3

In this case, a distributed query in general is considered. There are more than one domain. There are

Table 5.2

The Size of the Solution Space for CASE 2

n	Number of Sequences
2	5
3	1957
4	$> 10^9$

relations which include joining attributes defined on different domains. Joining attributes in such relations are associated with other joining attributes.

Since a semijoin f_{ij} can appear more than once in a sequence of semijoins because of the semijoin to an attribute associated with a_i, the maximum length of a sequence of semijoins can not be computed. Consequently, the size of the solution space can not be derived.

From the above discussion, it is obvious that an exhaustive search method is prohibitive in finding an optimal solution for distributed query processing.

As observed in CASE 3, one of the major difficulties is the possible reoccurrence of a semijoin in an optimal sequence of semijoins. For this reason, the length of an optimal sequence of semijoins can not be determined a priori.

5.2 A Block-Oriented Heuristic Algorithm

In this section, we present the main features of our heuristic algorithm for deriving a sequence of semijoins. Further, we discuss the heuristics on which these features are based.

Hereafter, a modular approach will be used to explain algorithms. A high-level description of a procedure will be given in pseudo-codes. Parts of a procedure that need to be elaborated will be described as independent procedures.

So far, no general heuristic method has been developed in the field of optimization for NP-hard sequencing problems. One technique generally applicable for sequencing problems is a neighbourhood search technique [BAKE 74]. This technique consists of the construction of a seed sequence followed by exchanges of positions of elements in the seed sequence. Ih our case, however, the neighbourhood search technique is very inefficient, since the number of sequences that can be generated from a seed sequence of length n is n!-1 and the cost of each of these n! sequences must be computed independently of the cost of another sequence already computed.

The net benefit n_i of a semijoin $\emptyset_i$ is the difference between the benefit b_i, the amount by which it reduces a relation, and the cost c_i, the amount of data transferred for its execution. A semijoin may not have net benefit at all or it may not have significant net benefit. However, it may cause substantial net benefits for the subsequent semijoins. Therefore, placing such semijoins with low cost

in the front part of a sequence of semijoins can decrease the cost of a query considerably.

In order to achieve this in an efficient manner, we make use of the structure of a query. For a given query, a block is a set of attributes defined on the same domain. A semijoin is possible between any pair of attributes in a block. Since blocks are linked through attributes in the same relations, the reduction of the cardinality of the set of values of attributes in one block can reduce those in other blocks. Our strategy is to process a block as a unit. That is, we select a block and schedule a sequence of semijoins among the attributes of the block, such that the cost of this sequence is minimal and the reduction caused by the sequence is maximally utilized in processing the subsequent blocks. Since there are smaller number of blocks than that of semijoins, processing a block as a unit leads to an efficient solution algorithm. Block-oriented processing is divided into three major modules discussed in the following subsections.

5.2.1 Process Blocks

This subsection discusses how a block to be processed can be selected and processed.

First, we consider the processing of a selected block.

Definition 5.1: Let an attribute a_i be defined on a domain D_k. The <u>density</u> d_i of a_i is $|K_i|/|D_k|$.

Consider a block $B_k = \{a_1, \ldots, a_n\}$. Initially $K_i = A_i$ for all i, and f_{ij} reduces d_j to $d_i d_j$. Hence if $d_h < d_i$, then a_h has more reductive power. Suppose $d_1 \leq \ldots \leq d_n$ for B_k. The basic strategy to process a block is to perform $f_{12}, \ldots, f_{n-1,n}$ to achieve a maximal reduction within a block with a minimal cost. Processing a block by scheduling semijoins which go from the attribute with the smallest density in a block to the one with the largest density is called a block visit.

We make the following observations about visiting a block which are useful in reducing the query cost:

(1) It is generally beneficial to finish the visit of a block at an attribute associated with a joining attribute of an unvisited block. By reducing the values of that attribute with the maximal reductive power of the block being visited, the costs of subsequent block visits can be reduced.

(2) If an attribute a_i in a block B_k is not associated with any attribute in other block and if the visit to the block B_k does not end in a_i, then the attribute a_i can be excluded from further scheduling of semijoins because the values of a_i are contained in the current values of some other attribute in the block B_k.

After a block is visited, the remaining attributes are called active attributes, and the excluded ones inactive attributes.

Next, we consider the selection of a block to be processed. Suppose $B_k = \{a_1, \ldots, a_n\}$ with $d_1 \leq \ldots \leq d_n$. The amount of data[1] transferred by the first semijoin f_{12} in visiting a block B_k is $|K_1|w_1 = |D_k|d_1w_1$. Since $|K_2| = |D_k|d_2$ is reduced by f_{12} to $|D_k|d_1d_2$, the amount of data transferred by f_{23} is $|D_k|d_1d_2w_1$ and is less than the data transferred by f_{12}. Following this argument it is easy to see that the amount of data transferred by each semijoin in visiting B_k is bounded by $|K_1|w_1$. Therefore, we have elected to take $|K_1|w_1$ as a part of a measure for selecting the next block to be visited.

In order to use the visit of a block to reduce the cost of visiting other blocks, we consider not only the cost incurred, but also the reductive power achieved by visiting a block when the next block to be visited is selected. Since B_k contains at least two joining attributes, d_1d_2 is a rough approximation of the reduction achieved by visiting B_k. To complete the measure for selecting the next block, we, therefore, multiply $|K_1|w_1$ by a penalty factor $1 + d_1d_2$, and we define the block cost BC(k) of an unvisited block B_k as follows:

$$BC(k) = |K_1|w_1(1 + d_1d_2) \qquad (5.3)$$

The purpose of procedure PROCESS_BLOCKS is basically to

---

[1]The data transfer incurred by transmission overhead is not considered in order to avoid notational complexity. This does not cause any difference for the purpose of analysis.

select and visit an unvisited block from the set of unvisited blocks with the least value of the block cost defined by (5.3) until there is no unvisited block left. Procedure PROCESS_BLOCKS, which includes some control features, will be presented in the next section. The following variables are defined to explain the algorithms:

(1) β: the set of all blocks

(2) $U\beta$: the set of unvisited blocks

(3) $V\beta$: the set of visited blocks

(4) $SV\beta$: the sequence of visited blocks

(5) σ: the sequence of semijoins being scheduled

(6) A(B): the set of active attributes in block B

(7) SI(B): the sequence of inactive attributes in block B

$A\beta(a_i)$ is the set of blocks which contains the attributes associated with a_i. The elements of $A\beta(a_i)$ are called the associate blocks of a_i. The set of end associate blocks, END_$A\beta$, at the end of visiting a block B_k which ends in a_i is given by $A\beta(a_i) \cap U\beta$. Initially $U\beta = \beta$, $V\beta = \emptyset$, A(B) = B, $\sigma = \emptyset$, SI(B) = $\emptyset$, and END_$A\beta = \emptyset$.

Procedure BLOCK_VISIT is shown below.

```
PROCEDURE BLOCK_VISIT(B)
  //Suppose A(B)={a_i1, ..., a_in}.//
  sort A(B) in ascending order of |K_ij|
  //Suppose sorted list is {a_1, ..., a_n}//
  For i=1 To n-1
  Do
    BEGIN
```

```
    append f_{i,i+1} to σ and reflect the effect[2] of f_{i,i+1}
      on the database state
    IF a_i is unassociated

    THEN
      BEGIN
        A(B) <- A(B)-{a_i}

        append a_i to SI(B)
      END
  END
IF a_n is unassociated AND a ∈ A(B)-{a_n} is

   associated with an attribute in B_k ∈ Uβ U Vβ
THEN
//Suppose Aβ ⊆ Uβ U Vβ is the set of blocks in//
//which the attributes associated with an attribute//
//in A(B)-{a_n} are contained//

  BEGIN
    IF Aβ Ω Uβ ≠ ø
    THEN
      //associated with an attribute in an //
      //unvisited block//
      BEGIN
        FOR  {a_i ∈ A(B)-{a_n}| a_i is associated with

              an attribute in Aβ Ω Uβ}
        DO FOR {B_j ∈ Aβ Ω Uβ|a ∈ B_j is associated with a_i}

           DO compute block cost BC(j) after f_{ni}

        select a_i with minimum block cost after f_{ni}
      END
    ELSE
      select a_i associated with an attribute in most

        recently visited block in SVβ
    append f_{ni} to σ and reflect the effect of f_{ni}

      on the database state
    A(B) <- A(B)-{a_n}

    append a_n to SI(B)
  END
IF |A(B)| > 1
//B contains more than one active attribute//
THEN
  BEGIN
```

---

[2]The effect of a semijoin on the database state is discussed in chapters 2, 3 and 4.

```
      Vβ <- Vβ U {B}
      append B to SVβ
    END

END BLOCK_VISIT
```

5.2.2 Reverse Process Blocks

This subsection discusses how to make use of the reduction of sets of values of joining attributes after all blocks have been visited.

In procedure PROCESS_BLOCKS, the reductive power of a visited block is utilized by subsequent block visits whenever possible. Consequently, a block visited later benefits by the reductions achieved by the blocks visited earlier.

In order to reduce the sets of values of joining attributes in the blocks visited earlier, using the reductive power of a block visited later, roughly the order of visits are reversed with respect to the order of visits during PROCESS_BLOCKS. Note that in procedure BLOCK_VISIT, a block which has more than one active attribute after it has been visited is appended to the sequence of visited blocks, because at least two joining attributes are necessary in a block to perform a semijoin. Procedure REVERSE_PROCESS_BLOCKS is given below. In REVERSE_PROCESS_BLOCKS, we make sure that a block has more than one active attribute, since some control features which will be explained in the next section may have excluded active attributes after the block has been visited.

```
PROCEDURE REVERSE_PROCESS_BLOCKS(SVβ)

  REPEAT
    B <- the last block in SVβ
    IF B ∈ Vβ AND B has more than one active attribute
    THEN
      BEGIN
        //Suppose A(B)={a_i1, ..., a_in}//

        sort A(B) in ascending order of |K_ij|

        //Suppose sorted list is {a_n, ..., a_1}//
        FOR i=n DOWN TO 2
        DO IF f_i,i-1 reduces |K_i-1| at least by 1

           THEN
             BEGIN
               append f_i,i-1 to σ and reflect the effect

                 of f_i,i-1 on the database state

               IF a_i is unassociated

                 THEN
                   BEGIN
                     A(B) <- A(B)-{a_i}

                     append a_i to SI(B)
                   END
             END
        Vβ <- Vβ-{B}
      END
    delete B from SVβ
  UNTIL SVβ = ø

END REVERSE_PROCESS_BLOCKS
```

5.2.3 Completion

In this subsection, we present an algorithm which reduces the size of the relations containing inactive attributes and with at least one target attribute using the reductive power accumulated in active attributes.

The sets of values of active attributes are reduced not only by procedure PROCESS_BLOCKS but also by procedure REVERSE_PROCESS_BLOCKS. For each block, the active

attribute with the smallest set of values is selected, then a sequence of beneficial[3] semijoins from this attribute to inactive attributes is scheduled.

Procedure COMPLETION is shown below.

```
PROCEDURE COMPLETION

  FOR k=1 TO |β|
  DO IF A(B_k) ≠ ø AND SI(B_k) ≠ ø
     THEN
       BEGIN
         //Suppose A(B_k) = {a_1, ..., a_i-1} and //
         //SI(B_k) = a_i, a_i+1, ..., a_n //
         select a_j ∈ A(B_k) such that |K_j| is the minimum
         s <- j
         t <- i
         WHILE t ≤ n
         DO
           BEGIN
             IF n_st > 0
             THEN
               BEGIN
                 append f_st to σ and reflect the
                   effect of f_st on the database state
                 s <- t
               END
             t <- t+1
           END
       END

END COMPLETION
```

5.3 Control Features

In this section, we present additional procedures which increase the robustness of the block-oriented heuristic algorithm for random input data.

Throughout the development of the block-oriented

---

[3]A semijoin $ø_i$ is beneficial if $n_i = b_i - c_i > 0$

heuristic algorithm, the stress test of the algorithm has been performed. We have tried to break up the algorithm by providing unusual input data. Solution produced using the main features alone are sometimes not sufficiently close to optimum. Additional control features are, therefore, necessary.

A few control features have been incorporated in the algorithm to obtain better solutions. These control features are mainly designed to take care of unusual database states and complex queries, and, therefore, are not invoked for typical applications. Four control features are presented in the following subsections.

5.3.1 Initial Inactivation of Attributes

This subsection discusses the exclusion of unassociated joining attributes with high initial density to avoid semijoins which are neither beneficial themselves nor useful for subsequent semijoins.

Even if a semijoin f_{ij} is not beneficial, if it reduces $|K_j|$ significantly, it can be useful to increase the benefits of subsequent semijoins. However, if the initial density of a_i is high, f_{ij} may not be beneficial and the reduction of $|K_j|$ is only $|K_j|(1-d_i)$. Therefore, in case the initial density of an unassociated attribute a_i is high, it is better to exclude a_i before using procedure PROCESS_BLOCKS, and then perform a semijoin to a_i in procedure COMPLETION.

In the algorithm, an attribute a_i is rendered inactive if its initial density is equal to or greater than 0.8. This value was determined from many query examples. Suppose a block contains unassociated attributes a_i, ..., a_{i+j} with initial density equal to or greater than 0.8. These attributes are sorted in ascending order of their densities, and the sorted sequence of inactive attributes is appended by attributes inactivated by procedure BLOCK_VISIT. In this way, the reductive power of attributes a_i, ..., a_{i+j}, although insignificant, is maximally utilized.

Procedure INIT_ATTR_INACTIVATION is shown below.

```
PROCEDURE INIT_ATTR_INACTIVATION

  FOR k=1 TO |β|
  DO
    //Initially A(B_k) = B_k and SI(B_k) = ø//
    BEGIN
      //Suppose B_k = {a_k1, ..., a_kn}//

      FOR j=1 TO n
      DO IF d_kj ≥ 0.8 AND a_kj is unassociated

         //Suppose R is the relation of which a_kj is//
         //an attribute//
         THEN IF R ∉ SJR OR R in user site
              THEN
                BEGIN
                  A(B_k) <- A(B_k)-{a_kj}

                  append a_kj to SI(B_k)
                END
      IF |SI(B_k)| > 1

      THEN sort SI(B_k) in ascending order of d_kj

      IF |A(B_k)| < 2

      THEN Uβ <- Uβ-{B_k}
    END

END INIT_ATTR_INACTIVATION
```

5.3.2 Path Construction

In this subsection, we shall describe the way to make better use of the reductive power accumulated in visited blocks to reduce the cost of visiting an unvisited block.

Suppose B_C is a candidate for the next block to be visited and there is an attribute $a_C \in B_C$ such that a_C is associated with a_{j1} in a visited block B_{p1}, and a_{i1} is an associated attribute with the smallest set of values in B_{p1}. Since $f_{i1,j1}$ can reduce $|K_C|$, $f_{i1,j1}$ may help reduce the total query cost by decreasing the cost while increasing the benefit of visiting B_C if B_C is selected as the next block to be visited. Likewise, if a_{i1} is associated with a_{j2} in a visited block B_{p2}, $B_{p1} \neq B_{p2}$, and a_{i2} is an attribute with the smallest set of values in B_{p2}, then $f_{i2,j2}$ followed by $f_{i1,j1}$ may be more beneficial than $f_{i1,j1}$ alone.

In this way, we can construct a path consisting of a sequence of semijoins leading to a candidate for the next block to be visited. Each semijoin in the path is from a visited block. In order to construct the most profitable of many possible paths, we define the cost function of a path as follows:

Let π be a path consisting of a sequence of semijoins ϕ_{p1}, ..., ϕ_{pn} where ϕ_{pi}, $1 \leq i \leq n$, is a semijoin between two attributes in a visited block B_{pi}. Let B_C be a candidate for the next block to be visited. The cost function, of path π leading to B_C, used in determining a path is given by

$$P_n(\pi,C) = \sum_{i=1}^{n} c_{pi} - \sum_{i=1}^{n} b_{pi} + BC(C)$$

$$= -\sum_{i=1}^{n} n_{pi} + BC(C) \qquad (5.4)$$

Note that the block cost of B_C also depends on path π because the set of values of an attribute in B_C is reduced due to the semijoins in π.

We now define the symbols which are used in procedure BUILD_PATH given below.

$P\beta = \{B_{pi} \mid 1 \leq i \leq n\}$. For each $a \in B_C$ when $\pi = \emptyset$ and for each $a \in B_{p1}$ when $\pi \neq \emptyset$ we define CAND_P$\beta(a) = \{B_k \in A\beta(a) \cap V\beta \mid$ there exists $a_h \in B_k$ which is associated with a and $a_g \in A(B_k)$ such that $d_g < d_h\}$. $\alpha = \{a \in B_C \mid$ CAND_P$\beta(a) \neq \emptyset\}$. a^* is an element of α with minimal density.

```
PROCEDURE BUILD_PATH(B_C)

  Pβ <- ∅
  //Initialize the path with a null sequence of semijoins//
  π <- ∅
  IF α ≠ ∅
  THEN
    BEGIN
      select a* ∈ α
      a_0 <- a*
      β* <- CAND_Pβ(a_0)
      REPEAT
        select B_j ∈ β* such that if ø_Bj is the semijoin from
          the attribute with the smallest density in B_j to
          the attribute associated with a_0 then P_n(ø_Bj, π, C)
          is minimum
        Pβ <- Pβ U {B_j}
        π <- ø_Bj, π
```

```
        //Suppose ø_Bj = f_xy//
        a_0 <- a_x
        β* <- CAND_Pβ(a_0) - Pβ
      UNTIL β* = ø
      //Suppose π = ø_p1, ø_p2 , ..., ø_pn//
      select π_j = ø_Pj, ø_p,j+1, ..., ø_pn ⊆ π
        such that P_n(π_j,C) is minimum
      π <- π_j
    END
END BUILD_PATH
```

Procedure BUILD_PATH is an indecomposable control feature of procedure PROCESS_BLOCKS mentioned in Section 5.2. Now that procedure BUILD_PATH has been presented, we are in the position to elaborate on the procedure PROCESS_BLOCKS.

In selecting a block to be visited, we sometimes choose two candidate blocks. This also serves to prevent the degeneration of the performance of the distributed query processing algorithm for unusual database states.

After the initial local processing, END_Aβ = ø. After the block B_k is visited ending with a_i, the sets of values of attributes associated with a_i are reduced, not only by the reductive power of B_k, but possibly the reductive power of the previously visited blocks. Hence if END_Aβ ≠ ø, we select B_{c1} ∈ END_Aβ with the minimum block cost as a candidate block to be visited next. It is usually the case that B_{c1} has the smallest block cost among all unvisited blocks, and is visited next. However, if this is not the

case, we select another candidate block, B_{c2}, with the minimum block cost among the blocks in Uβ.

We build paths for both of the candidate blocks using (5.4) before selecting one as a block to be visited next. In comparing the two candidate blocks, we use another cost function as follows:

Let π and B_C be defined as before. The cost function, of path π and candidate block B_C, used in selecting the next block to be visited is given by

$$P_c(\pi,C) = \sum_{i=1}^{n} c_{pi} + BC(C) \qquad (5.5)$$

By using (5.4) in constructing a path for a selected block, we use the reductive power of visited blocks as much as possible. By using (5.5) in selecting the next block to be visited, the cost incurred by a sequence of semijoins is kept as low as possible.

In case END_Aβ = ø, only one candidate block, B_{c1}, is selected such that B_{c1} is with the minimum block cost among the blocks in Uβ.

Procedure PROCESS_BLOCKS is shown below.

```
PROCEDURE PROCESS_BLOCKS

  END_Aβ <- ø
  WHILE Uβ ≠ ø
  DO
    BEGIN
      find candidate blocks
      //NUM_CAND is the number of candidate blocks//
      //π_ci is the path for candidate block B_ci//

      FOR i = 1 TO NUM_CAND
      DO BUILD_PATH(B_ci)
```

```
      //B_N is the next block to be visited and π//
      //is the path to it//
      IF NUM_CAND = 1 OR P_c(π_c1, c1) ≤ P_c(π_c2, c2)
      THEN
        BEGIN
          B_N <- B_c1
          π <- π_c1
        END
      ELSE
        BEGIN
          B_N <- B_c2
          π <- π_c2
        END
      //Process the path and visit the block//
      append π to σ and reflect the effects of semijoins
        in π on the database state
      append the sequence of visited blocks, corresponding
        to the semijoins in π, to SVβ
      VISIT_BLOCK(B_N)  //Suppose visit ends with a ∈ B_N//
      Uβ <- Uβ - {B_N}
      END_Aβ <- Aβ(a) Ω Uβ
    END
END PROCESS_BLOCKS
```

5.3.3 Hill-Climbing

A hill-climbing technique can be adopted before using procedure COMPLETION to further decrease the query cost.

Only the semijoins between active attributes need to be considered. In order to maintain the efficiency of the algorithm at the same time decrease the query cost, the beneficial semijoin which has the least cost is appended to the scheduled sequence of semijoins until no such semijoin is available. Basically, the semijoins are checked in ascending order of their costs until a beneficial one is found.

Procedure HILL_CLIMBING is shown below.

```
PROCEDURE HILL_CLIMBING

  SET_ACTIVE <-  U_k A(B_k) where the union is performed  over
                 those k for which |A(B_k)| > 1
  TEMP_ACTIVE <- SET_ACTIVE
  WHILE TEMP_ACTIVE ≠ ø
  DO
    BEGIN
      select a_i ∈ TEMP_ACTIVE such that the cost of f_ix
        to its equivalent joining attribute a_x is the least
      select a_j ∈ TEMP_ACTIVE such that n_ij is the largest
      IF n_ij > 0
      THEN
        BEGIN
          append f_ij to σ and reflect the effect of
            f_ij on the database state
          TEMP_ACTIVE <- SET_ACTIVE - {a_i}
        END
      ELSE TEMP_ACTIVE <- TEMP_ACTIVE - {a_i}
    END

END HILL_CLIMBING
```

5.3.4 Screening

In this subsection, a procedure which deletes obviously unnecessary semijoins scheduled by the procedures previously described will be presented.

Suppose $\sigma = \phi_1, \ldots, \phi_t$. Specifically, the following improvements are considered:

(1) Let $\phi_h = f_{ij}$ and a_j be an attribute of relation R. If ϕ_m neither comes from nor goes to the attribute of R for all $m > h$, ϕ_h can be deleted from σ without affecting the costs or benefits of other

semijoins. In this case, if $n_{ij} \leq 0$, the deletion of ϕ_h decreases the query cost.

(2) Let ϕ_h and R be defined as in (1), and a_i be an attribute of R'. In addition, let R be at the user site. If ϕ_m does not go to an attribute of R' for all $m > h$, |R'| will not be reduced after ϕ_h. In this case, by moving R' to the user site instead of performing ϕ_h, the cost of ϕ_h is saved.

Procedure SCREENING is shown below.

```
PROCEDURE SCREENING

  //Suppose σ = φ1, ..., φt//

  FOR h = t DOWN TO 1
    //Suppose φh = fij and aj is an attribute of R//

    IF nij ≤ 0

    THEN IF φm neither comes from nor goes to the attribute
                of R for all m > h
         THEN delete φh from σ

  //Suppose σ = φ1, ..., φs.  STU is the set of//

  //relations at the user site//
  IF STU ≠ ∅

  THEN FOR h = s DOWN TO 1
       //Suppose φh = fij, and ai and aj are attributes//

       //of R' and R, respectively//
       IF R ∈ STU

       THEN IF φm does not go to the attribute of R'
                    for all m > h
            THEN
              BEGIN
                delete φh from σ

                insert "move R'" at the position of φh
              END
```

END SCREENING

5.4 Algorithm H and Its Complexity Analysis

In summary, Algorithm H which generates a sequence of semijoins to process a distributed query consists of the procedures presented in Sections 5.2 and 5.3.

Algorithm H is presented below.

Algorithm H

INPUT: User query and the initial database state

OUTPUT: Sequence of semijoins

```
INIT_ATTR_INACTIVATION
PROCESS_BLOCKS
IF Vβ ≠ ø THEN REVERSE_PROCESS_BLOCKS
HILL_CLIMBING
COMPLETION
SCREENING
```

We now consider the time complexity of Algorithm H. The measure of the complexity is the number of sequences of semijoins generated in accordance with the complexity of an exaustive search method discussed in Section 5.1.

The existence of procedure HILL_CLIMBING in Algorithm H makes it difficult to derive a narrow-bound time complexity. Suppose there are s possible semijoins and m relations involved in processing a query, then there are s choices for

the ith semijoin in the sequence of semijoins being generated by procedure HILL_CLIMBING. The length λ of the sequence of semijoins, however, cannot be determined a priori as mentioned before. Since a beneficial semijoin has to reduce at least one tuple in a relation, in the worst case $\lambda = \sum_{i=1}^{m} |R_i|$. Hence the worst case complexity of procedure HILL_CLIMBING is $O(s\sum_{i=1}^{m}|R_i|)$. Fortunately, procedure HILL_CLIMBING is a refinement feature and very seldom utilized.

The derivation of the worst case complexity of procedure BUILD_PATH is as follows:

Let $m = |\beta|$ and λ the length of the path being constructed by procedure BUILD_PATH. Since $U\beta \neq \emptyset$ when procedure BUILD_PATH is invoked, $\lambda \leq |V\beta| \leq m-1$. In the worst case, procedure BUILD_PATH generates $\lambda(m-\lambda)$ sequences when $\lambda = 1, 2, \ldots, m-1$. Therefore, the total number of sequences generated is

$$\sum_{\lambda=1}^{m-1} \lambda(m - \lambda)$$

$$= m\left(\sum_{\lambda=1}^{m-1} \lambda\right) - \sum_{\lambda=1}^{m-1} \lambda^2$$

$$= m\{(m-1)m/2\} - (m-1)m(2m-1)/6$$

$$= (m^3 - m)/6.$$

Hence the worst case complexity of procedure BUILD_PATH is $O(|\beta|^3)$. Just like procedure HILL_CLIMBING, procedure BUILD_PATH is very seldom utilized.

Since most queries are handled only by the main

features of Algorithm H, we shall consider the complexity of the main features of Algorithm H.

Theorem 5.1: The worst case complexity of the main features of Algorithm H without procedure BUILD_PATH is $O(n^*|\beta|)$, where $n^* = \max \{|B| \mid B \in \beta\}$.

Proof: Procedures PROCESS_BLOCKS, REVERSE_PROCESS_BLOCKS, and COMPLETION, the main features of Algorithm H, are sequentially invoked. Since some of the active attributes become inactive after the invocation of procedure PROCESS_BLOCKS, the number of sequences generated by procedure PROCESS_BLOCKS is greater than or equal to that by either of the other two procedures. Hence we have only to consider the complexity of procedure PROCESS_BLOCKS. Procedure PROCESS_BLOCKS has two major loops, one embedded within the other. For each block in β, procedure BLOCK_VISIT generates sequences of semijoins. Consider $B = \{a_1, \ldots, a_n\} \in \beta$ such that $d_1 \leq \ldots \leq d_n$. Procedure BLOCK_VISIT schedules a sequence $f_{12}, \ldots, f_{n-1,n}$, which may be appended by a semijoin f_{nj} where $1 \leq j \leq n-1$. Therefore, procedure BLOCK_VISIT generates one sequence of length λ, for $\lambda = 1, \ldots, n-1$, and maximum $n-1$ sequences of length n. The maximum number of sequences that can be generated by procedure BLOCK_VISIT is $2(n-1)$. Since $n \leq n^*$, the worst case complexity of procedure PROCESS_BLOCKS is $O(n^*|\beta|)$. ■

CHAPTER 6

TESTING THE SOLUTION ALGORITHM

Algorithm H presented in the previous chapter has been implemented in PASCAL and runs on Amdahl 470V/8 under the Michigan Terminal Systems at the University of Michigan.

The purpose of this chapter is to evaluate the performance of Algorithm H with respect to the effectiveness of the data reduction as well as the efficiency. The results of the test runs of the PASCAL program implementing Algorithm H are presented. In addition, comparisons with the results of other algorithms are made.

6.1 Data Traffic Reduction

The goal of distributed query optimization is to minimize the data traffic incurred by a distributed query in a computer network. Reference solutions are needed to evaluate the solutions produced by Algorithm H. Since no optimal algorithm is available except an exhaustive search method for distributed query optimization, and the use of exhaustive search methods is prohibitive even for small-size queries as discussed in Section 5.1, the solutions produced by other algorithms in this area are used as a basis for comparison.

In order to make objective comparisons, we have selected examples from recently published papers [HEVN 79b, BERN 81, CHEU 82], which have similar formulations of the problem to ours, as benchmarks for tests. Further, an example is constructed to illustrate a particular feature of Algorithm H. Using these examples, we also show the execution steps of Algorithm H for different queries and database states.

Throughout the computations involved in algorithms, the costs, benefits and cardinalities are first computed in real numbers, then the results are given in integers by rounding the real numbers. In actual databases, the cardinalities of tuples in relations or those of attribute values are integers. However, integer arithmetic is not only inappropriate to the nature of the statistical estimation method being used, but is also the source of computational overhead.

Example 6.1

The example by Hevner and Yao [HEVN 79b] is considered. This example is also used by Cheung [CHEU 82]. The database has the following four relations each of which is located at a different site:

EMPLOYEE(E#, ENAME, SEX)

COURSE(C#, CNAME, LEVEL)

STUDENT_COURSE(E#, C#)

TEACHER_COURSE(E#, C#, ROOM)

The relation TEACHER_COURSE is at the user site. The query is: for all male employees who are teaching advanced courses in Room 103 and are students in at least one course, list the employees' names and the courses they are teaching. The relational form of the query is as follows:

```
FIND  (EMPLOYEE.ENAME, COURSE.CNAME)
WHERE (EMPLOYEE.E# = STUDENT_COURSE.E#)
 AND  (EMPLOYEE.E# = TEACHER_COURSE.E#)
 AND  (TEACHER_COURSE.C# = COURSE.C#)
 AND  (COURSE.LEVEL = 'Advanced')
 AND  (TEACHER_COURSE.ROOM = '103')
 AND  (EMPLOYEE.SEX = 'M')
```

The parameters for the query and the database are defined as follows:

R_1 = COURSE, R_2 = TEACHER_COURSE

R_3 = EMPLOYEE, R_4 = STUDENT_COURSE

a_1 = COURSE.C#

a_2 = TEACHER_COURSE.C#

a_3 = TEACHER_COURSE.E#

a_4 = EMPLOYEE.E#

a_5 = STUDENT_COURSE.E#

a_c = COURSE.CNAME

a_e = EMPLOYEE.ENAME

After initial local processing, the reduced query and the given initial database state are shown below.

The reduced query:

FIND (a_c, a_e)
WHERE $(a_1 = a_2)$
AND $(a_3 = a_4)$
AND $(a_4 = a_5)$

From the reduced query, we have $B_1 = \{a_1, a_2\}$ and $B_2 = \{a_3, a_4, a_5\}$

The initial database state is:

$|R_1| = 100, \quad |R_2| = 300, \quad |R_3| = 200, \quad |R_4| = 600$
$|A_1| = 100, \quad |A_2| = |A_3| = |A_4| = 200, \quad |A_5| = 600$
$w_i = 1$ for $i = 1, \ldots, 5$
$w_c = 11, \quad w_e = 9$
$|D_1| = 400, \quad |D_2| = 1000$

In accordance with Hevner and Yao's example, the communication network parameters, fixed overhead V_f and proportional coefficient υ, are set to 10 and 1, respectively. Then the initial cost of moving R_1, R_3 and R_4 after initial local processings to the user site is 3830. For Hevner and Yao's method, the reported cost of the query [HEVN 79b] was 1324. The cost reported by Cheung's method [CHEU 82] was also 1324. We compute the cost for the same example by Algorithm H and the SDD-1 algorithm [BERN 81] using our estimation method for the cardinalities of reduced relations.

Query cost by Algorithm H

The lattices L_1 and L_2 for B_1 and B_2, respectively, and the changes of K_i's during the application of σ are shown in Figure 6.1. The changes of values of database state variables after each semijoin in σ are shown in Table 6.1 along with b_i, c_i and n_i for each semijoin.

1. INIT_ATTR_INACTIVATION

Since $d_i < 0.8$ for all a_i, all the attributes are initially active.

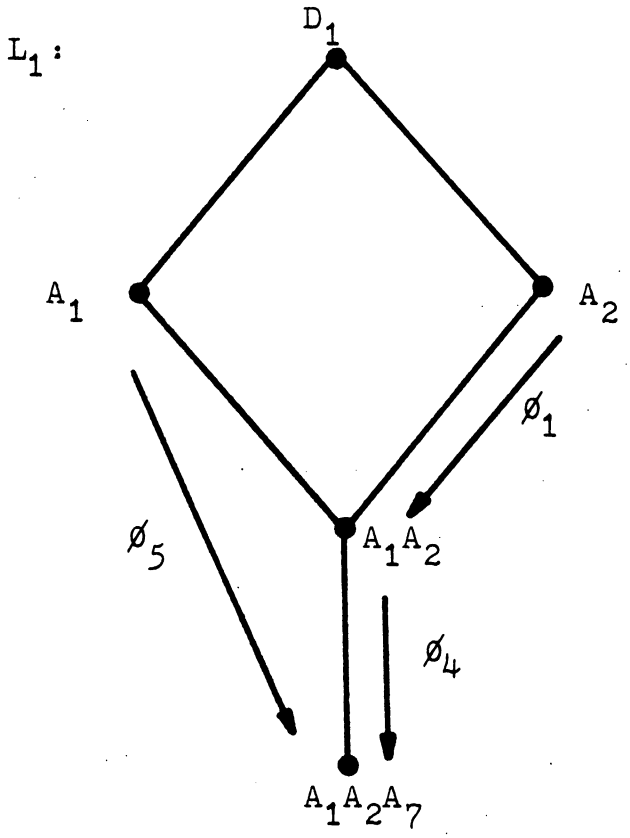
2. PROCESS_BLOCKS

(1) Since initially END_Aβ = $\emptyset$, a block to be visited is selected from Uβ = $\{B_1, B_2\}$ with the minimum block cost. Using (5.3),

$$BC(1) = |K_1| w_1 (1 + d_1 d_2)$$
$$= 100 \times 1 \times (1 + 0.25 \times 0.5)$$
$$= 112.5$$
$$BC(2) = |K_3| w_3 (1 + d_3 d_4)$$
$$= 200 \times 1 \times (1 + 0.2 \times 0.2)$$
$$= 208$$

Since $BC(1) < BC(2)$, B_1 is selected for the visit.

(2) Since Vβ = $\emptyset$, no path is built for B_1. By procedure BLOCK_VISIT, $\phi_1 = f_{12}$ is appended to σ, and a_1 is inactivated. After f_{12}, a new set A_3A_6 which represents the reduced K_3 is formed. After B_1 is visited, Uβ = END_Aβ = $\{B_2\}$. Since B_1 has only one active attribute, B_1 is not included in Vβ.



The Expansions of the Lattices by the Sequence of Semijoins Generated by Algorithm H for Example 6.1

Figure 6.1

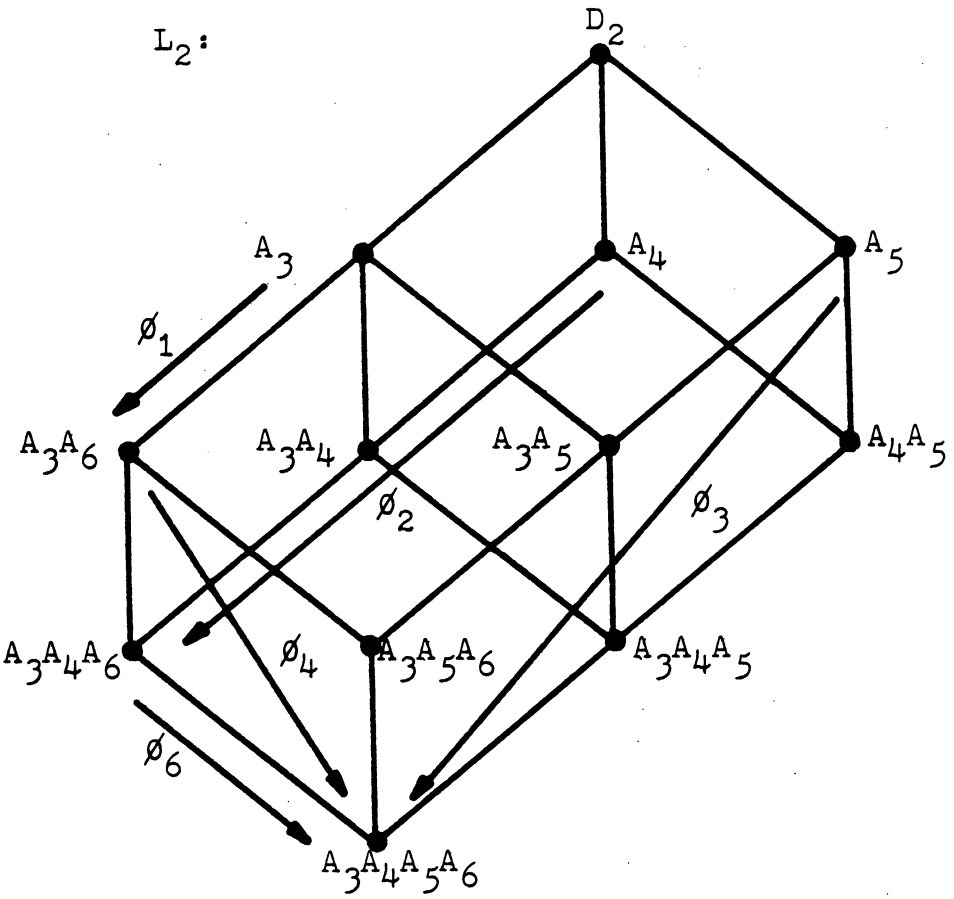


Figure 6.1 continued.

(3) Since $U\beta = \{B_2\}$, B_2 is the next block to be visited. Since $V\beta = \emptyset$, no path is built for B_2. In procedure BLOCK_VISIT, $\emptyset_2 = f_{34}$, $\emptyset_3 = f_{45}$ are appended to σ, and a_4 becomes inactive. $V\beta$ becomes $\{B_2\}$ and $U\beta$ becomes $\emptyset$.

3. REVERSE_PROCESS_BLOCKS

Since $V\beta = \{B_2\}$, $\emptyset_4 = f_{53}$ is appended to σ. After f_{53}, a new set $A_1A_2A_7$ which represents the reduced K_2 is formed. Since $R_4 \in SJR$, R_4 is ignored after f_{53}.

4. HILL_CLIMBING

Since none of the semijoins are beneficial, procedure HILL_CLOMBING does not append any semijoin to σ.

5. COMPLETION

(1) For B_1, a_2 is the only active attribute, while a_1 is the only inactive attribute. Hence $\emptyset_5 = f_{21}$ is appended to σ.

(2) For B_2, a_3 is the only active attribute while a_4 is the only inactive attribute. Hence $\emptyset_6 = f_{34}$ is appended to σ.

6. SCREENING

No semijoin is deleted from σ by procedure SCREENING.

From Table 6.1, the cost of the query QC by Algorithm H is:

$$QC = IC - \sum_{i=1}^{6} n_i$$

$$= 3830 - 3352$$

Table 6.1

The Sequence of Semijoins by Algorithm H
and Its Effect for Hevner and Yao's Example

i	1	2	3	4	5	6
ϕ_i	f_{12}	f_{34}	f_{45}	f_{53}	f_{21}	f_{34}
$\|R_1\|$					8.7	
$\|K_1\|$					8.7	
$\|R_2\|$	75.0			9.0		
$\|K_2\|$	50.0			8.7		
$\|K_3\|$	70.0			8.4		
$\|R_3\|$		14.0				8.4
$\|K_4\|$		14.0				8.4
$\|R_4\|$			8.4			
$\|K_5\|$			8.4			
b_i	0.0	1860.0	591.6	18.4	1095.3	56.0
c_i	110.0	80.0	24.0	18.4	18.7	18.4
n_i	-110.0	1780.0	567.6	0.0	1076.6	37.6

= 478

Query cost by the SDD-1 algorithm

We follow the same procedure used in the SDD-1 algorithm and compute the cost of transmitting the data

during the sequence of semijoins and transmitting the reduced relations to the assembly site. However, for the SDD-1 algorithm, we further include the cost of transmitting the assembled answer from the assembly site to the user site.

1. Hill-Climbing

In this phase, the semijoin with the largest net benefit is appended to the sequence of semijoins being constructed until there is no remaining beneficial semijoin. It should be noted that the reductions of relations at the user site are considered as benefits in the SDD-1 algorithm. Hence in computing the query cost, the costs of semijoins are used instead of their net benefits. The sequence of semijoins scheduled in this phase and its effect are summarized in Table 6.2.

From Table 6.2, the part of the query cost QC_1 incurred by the sequence of semijoins is:

$$QC_1 = \sum_{i=1}^{7} c_i$$

$$= 410.0$$

2. Assembly

In this phase, the reduced relations are moved to the site where the largest relation is located. Let S(R) denote the size of the relation R. After the hill-climbing phase, the size of each relation is as follows:

$$S(R_1) = |R_1|(w_1 + w_c) = 104.7$$

Table 6.2

The Sequence of Semijoins by the SDD-1 Algorithm and Its Effect for Hevner and Yao's Example

i	1	2	3	4	5	6	7
ϕ_i	f_{34}	f_{45}	f_{53}	f_{21}	f_{54}	f_{12}	f_{34}
$\|R_1\|$				8.7			
$\|K_1\|$				8.7			
$\|R_2\|$			36.0			9.0	
$\|K_2\|$			34.9			8.7	
$\|K_3\|$			24.0			8.4	
$\|R_3\|$	40.0				24.0		8.4
$\|K_4\|$	40.0				24.0		8.4
$\|R_4\|$		24.0					
$\|K_5\|$		24.0					
b_i	1600.0	576.0	528.0$^+$	1095.3	160.0	54.0$^+$	155.9
c_i	210.0	50.0	34.0	44.9	34.0	18.7	18.4

$^+$The actual benefits of these semijoins are 0, since R_2 is at the user site.

$$S(R_2) = |R_2|(w_2 + w_3) = 18$$

$$S(R_3) = |R_3|(W_4 + w_e) = 84.1$$

$$S(R_4) = |R_4|w_5 = 24$$

Hence the site of R_1 is selected as an assembly site. The part of the query cost QC_2 to move R_2, R_3 and R_4 to the assembly site is:

$$QC_2 = \sum_{i=2}^{4} \{10 + S(R_i)\}$$

$$= 156.1$$

3. Enhancements

In this phase, the sequence of semijoins scheduled in the hill-climbing phase is examined for possible reordering and/or deletion of semijoins. Since neither reordering nor deletion is applicable in this case, no enhancements are made.

4. Answer Move

Since the assembly site is the site of R_1, and the user site is the site of R_2, it is necessary to move the query answer assembled at the assembly site to the user site. Let R_A be the answer relation. Then

$$R_A = \{R_1[a_1 = a_2]R_2[a_3 = a_4]R_3[a_4 = a_5]R_4\}[a_c, a_e].$$

$|R_A| = 9$, since $|R_1| = |K_1|$, $K_1 = K_2$, $|R_3| = |K_4|$, $K_4 = K_3$, $K_4 \subset K_5$ and $|R_2| = 9$. The width of R_A is $w_c + w_e$. Hence the part of the query cost QC_3 to move R_A to the user site is:

$$QC_3 = S(R_A) + 10$$

$$= 190$$

The cost of the query QC by the SDD-1 algorithm is:

$$QC = QC_1 + QC_2 + QC_3$$

$$= 756$$

The query costs for Hevner and Yao's example are summarized in Table 6.3.

Table 6.3

Query Costs for Hevner and Yao's Example

Algorithm	Query Cost
Initial Cost	3830
Hevner and Yao	1324
SDD-1	756
Cheung	1324
Algorithm H	478

Example 6.2

An example by Bernstein et al. [BERN 81] is considered. Their example is incomplete, since the user site is not specified. We assume that the user site is not one of the sites at which the relations referenced by the user query are located. The necessary informations to process the query are as follows:

Database:

S(S#, NAME, LOCATION)

Y(S#, P#)

P(P#, NAME, TYPE)

Each relation is stored at a separate site.

Query:

```
FIND    (S.S#, S.NAME, S.LOCATION,
         Y.S#, Y.P#,
         P.P#, P.NAME, P.TYPE)
WHERE   (S.LOCATION = 'MA')
 AND    (P.TYPE = 'Micro')
 AND    (S.S# = Y.S#)
 AND    (Y.P# = P.P#)
```

Parameters:

a_1 = S.S#, a_2 = Y.S#, a_3 = Y.P#, a_4 = P.P#

a_s = S.NAME, a_l = S.LOCATION

a_p = P.NAME, a_t = P.TYPE

The reduced query after initial local processing:

FIND $(a_1, a_s, a_l, a_2, a_3, a_4, a_p, a_t)$

WHERE $(a_1 = a_2)$

AND $(a_3 = a_4)$

From the reduced query, $B_1 = \{a_1, a_2\}$ and $B_2 = \{a_3, a_4\}$.

The initial database state:

|S| = 200, |Y| = 100000, |P| = 2000

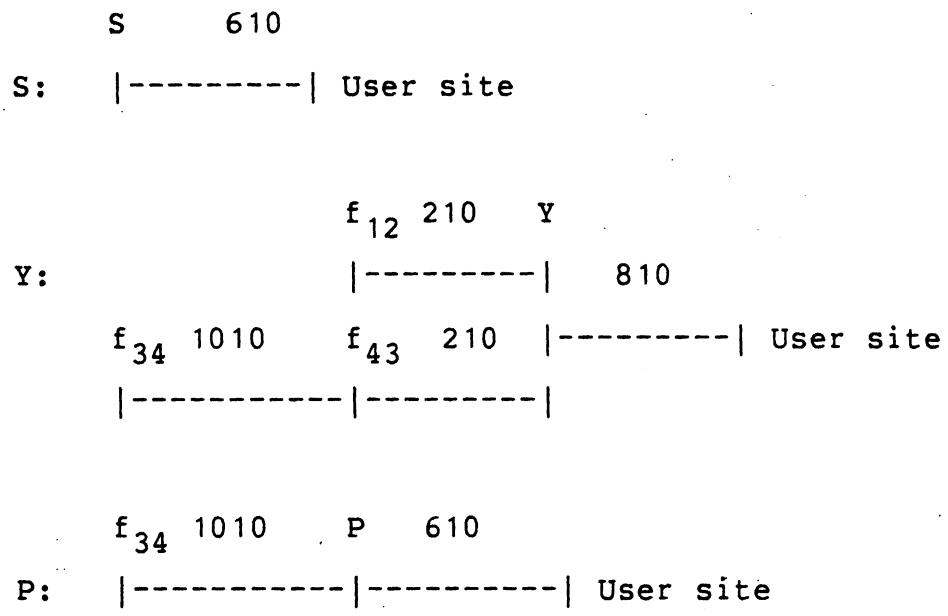
$|A_1| = 200$, $|A_2| = |A_3| = 1000$, $|A_4| = 2000$
All attributes have a width of 1. $|D_1| = |D_2| = 10000$

The communication network parameters:
$V_f = 10$, $\upsilon = 1$

The initial cost of the query to move S, Y and P to the user site after initial local processing is 206630. We compute the cost of the query using different algorithms.

Query cost by Hevner and Yao's algorithm

The integrated schedule for each relation is shown in Figure 6.2.



The Schedule by Hevner and Yao's Algorithm for Example 6.2

Figure 6.2

Adding all the edge values in Figure 2, the query cost

is 4470.

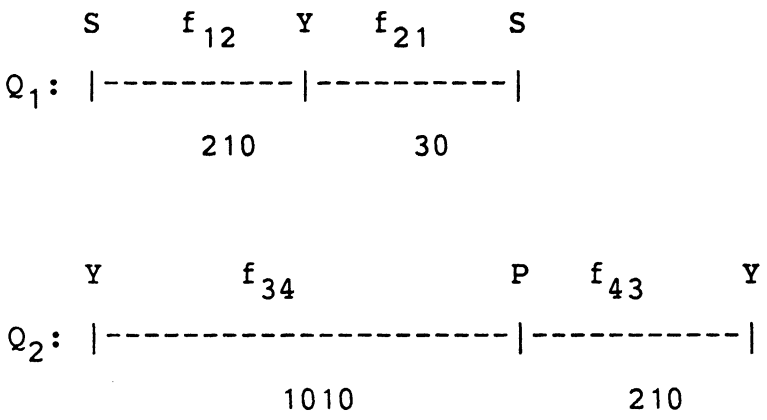
Query cost by Cheung's algorithm

In Cheung's method, the qualification clause of the query is decomposed into simple query clauses as follows:

$$q_1 \equiv a_1 = a_2$$

$$q_2 \equiv a_3 = a_4$$

where q_i is the qualification clause of the simple query Q_i for i = 1, 2. Then, a serial schedule is constructed for each simple query as shown in Figure 6.3.



The Serial Schedules by Cheung's Algorithm for Example 6.2

Figure 6.3

After the semijoins in Figure 6.3, S(S) = 60, S(Y) = 800 and S(P) = 600. Adding all the edge values in Figure 6.3 to the cost of moving reduced relations, the query cost is 2950.

Query cost by Algorithm H

The sequence of semijoins generated by Algorithm H and its effect are shown in Table 6.4. The expansions of lattices L_1 and L_2 corresponding to B_1 and B_2, respectively, are shown in Figure 6.4.

From Table 6.4, the query cost QC is:

$$QC = IC - \sum_{i=1}^{4} n_i$$

$$= 2711$$

Query cost by the SDD-1 algorithm

The sequence of semijoins generated by the hill-climbing phase is the same as that generated by Algorithm H. However, since the site of Y is selected as an assembly site, $\phi_3 = f_{43}$ is deleted in the enhancement phase. Therefore, the part of the query cost QC_1 by a sequence of semijoins is as follows:

$$QC_1 = c_1 + c_2 + c_4$$

$$= 1117.4$$

The part of the query cost QC_2 by moving the relations to the assembly site is:

$$QC_2 = S(S) + S(P) + 20$$

$$= 600.5$$

Let R_A be the answer relation assembled at the site of Y. Before assembling R_A, $K_1 = K_2$, $K_3 = K_4$ and $|Y| = 400$.

Table 6.4

The Sequence of Semijoins by Algorithm H and Its Effect for the Example Given by Bernstein et al.

ϕ_i	f_{12}	f_{34}	f_{43}	f_{21}
$\|S\|$				20.0
$\|K_1\|$				20.0
$\|Y\|$	2000.0		400.0	
$\|K_2\|$	20.0		20.0	
$\|K_3\|$	867.4		173.5	
$\|P\|$		173.5		
$\|K_4\|$		173.5		
b_i	196000.0	5479.5	3200.0	540.0
c_i	210.0	877.4	183.5	30.0
n_i	195790.0	4602.1	3016.5	510.0

Hence $|R_A| = 400$ and the width of R_A is 6. The part of the query cost QC_3 to move R_A to the user site is:

$$QC_3 = S(R_A) + 10$$

$$= 2410$$

The query cost QC is:

$$QC = QC_1 + QC_2 + QC_3$$

$$= 4128$$

Table 6.5 shows the costs of the query for this example obtained by different algorithms.

Table 6.5

Query Costs for the Example Given by Bernstein et al.

Algorithm	Query Cost
Initial Cost	206630
Hevner and Yao	4470
SDD-1	4128
Cheung	2950
Algorithm H	2711

If the site of Y is the user site, procedure SCREENING in Algorithm H eliminates $\phi_3 = f_{43}$ from σ. Also, since the assembly site and the user site are the same for the SDD-1 algorithm, the assembled answer does not have to be moved. In this case, Algorithm H and the SDD-1 algorithm produce identical sequence of semijoins and query cost. However, if the user site is the site of S or P, the query cost according to the SDD-1 algorithm is also 4128. The query cost obtained by Algorithm H is 2611 when S is at the user site whereas the cost is 2181 when P is at the user site.

Example 5.3

An example by Cheung [CHEU 82] is considered. The query and the database are as follows:

The database has the same relations as in Example 5.1:

R_1 = TEACHER_COURSE R_2 = STUDENT_COURSE

R_3 = COURSE R_4 = EMPLOYEE

The relation EMPLOYEE is at the user site. Let

a_1 = EMPLOYEE.E#

a_2 = STUDENT_COURSE.E#

a_3 = TEACHER_COURSE.E#

a_4 = TEACHER_COURSE.C#

a_5 = COURSE.C#

The reduced query after initial local processing:

FIND (a_1)
WHERE ($a_1 = a_2$)
 AND ($a_2 = a_3$)
 AND ($a_4 = a_5$)

From the reduced query, $B_1 = \{a_1, a_2, a_3\}$ and $B_2 = \{a_4, a_5\}$.

The initial database state:

$|R_1| = 300$, $|R_2| = 300$, $|R_3| = 400$, $|R_4| = 200$

$|A_1| = 200$, $|A_2| = 300$, $|A_3| = 200$, $|A_4| = 300$, $|A_5| = 400$

$w_i = 1$ for all a_i

$|D_1| = 400$, $|D_2| = 600$

The communication network parameters:

$V_f = 10$, $\upsilon = 1$

For this example, the initial cost of the query is 1330. The cost of the query reported using Hevner and Yao's

algorithm is 1240, whereas the cost using Cheung's algorithm is 865. We compute the cost of the query using Algorithm H and the SDD-1 algorithm.

Query cost by Algorithm H

The sequence of semijoins produced by Algorithm H is $\sigma = f_{13}, f_{32}, f_{23}, f_{45}, f_{54}$. After σ is processed R_2, R_3, R_4 are ignored, since they are singleton joining relations. Since a_1, a_2 and a_3 are joining attributes defined on the same domain D_1, a_3 can be used as a target attribute after a_1 is ignored. The cost of the query QC given by Algorithm H is:

$$QC = IC - \sum_{i=1}^{5} n_i$$

$$= 1330 - 647$$

$$= 683$$

Query cost by the SDD-1 algorithm

1. Hill-Climbing

The sequence of semijoins constructed in this phase is $\sigma = f_{13}, f_{45}, f_{32}, f_{54}, f_{31}$. The part of the query cost QC_1 incurred in this phase is:

$$QC_1 = \sum_{i=1}^{5} c_i$$

$$= 639.6$$

2. Assembly

After processing σ, $S(R_1) = 150$, $S(R_2) = 75$, $S(R_3) = 75$ and $S(R_4) = 64.6$. Hence the site of R_1 is selected as an assembly site. The assembly cost QC_2 is:

$$QC_2 = \sum_{i=2}^{4} \{10 + S(R_i)\}$$

$$= 244.6$$

3. Enhancements

In this phase, two enhancements are made:

(1) In σ, the positions of f_{32} and f_{54} are switched. Consequently, the cost of f_{32} is reduced from 110 to 74.6. Also, $S(R_2)$ is reduced from 75 to 48.5.

(2) By the choice of an assembly site, f_{54} is deleted where the cost of f_{54} is 85. The total reduction QC_R of the query cost by enhancements is:

$$QC_R = (110 - 74.6) + (75 - 48.5) + 85$$

$$= 146.9$$

4. Answer Move

Since the assembly site and the user site are different, the answer relation R_A has to be moved. $|R_A| = 48.5$ and its width is 1. The cost QC_3 of moving R_A is 58.5. The cost of the query QC by the SDD-1 algorithm is:

$$QC = QC_1 + QC_2 + QC_3 - QC_R$$

$$= 795.8$$

The query costs for Cheung's example by different

algorithms are compared in Table 6.6.

Table 6.6

Query Costs for Cheung's Example

Algorithm	Query Cost
Initial Cost	1330
Hevner and Yao	1240
SDD-1	796
Cheung	865
Algorithm H	683

Example 5.4

An example is constructed to illustrate the chain effect on singleton joining relations. Consider a chain query referencing five relations as follows:

Database:

$R_1(a_1)$, $R_2(a_2, a_3)$, $R_3(a_4, a_5)$, $R_4(a_6, a_7)$, $R_5(a_8, a_9)$

Each relation is stored at a separate site and the user is also at a different site.

The reduced query after initial local processing:

FIND (a_9)

WHERE $(a_1 = a_2)$

AND $(a_3 = a_4)$

AND $(a_5 = a_6)$

AND $(a_7 = a_8)$

From the reduced query, $B_1 = \{a_1, a_2\}$, $B_2 = \{a_3, a_4\}$ $B_3 = \{a_5, a_6\}$ and $B_4 = \{a_7, a_8\}$.

The initial database state:

$|R_1| = 100$, $|R_2| = 200$, $|R_3| = 350$, $|R_4| = 550$, $|R_5| = 800$

$|A_1| = 100$, $|A_2| = 150$, $|A_3| = 200$, $|A_4| = 250$

$|A_5| = 350$, $|A_6| = 400$, $|A_7| = 550$, $|A_8| = 600$

All attributes have widths of 1.

$|D_1| = 250$, $|D_2| = 450$, $|D_3| = 800$, $|D_4| = 1200$

The communication network parameters:

$V_f = 10$, $\upsilon = 1$

Algorithm H produces a sequence of semijoins $\sigma = f_{12}$, f_{34}, f_{56}, f_{78}. Note that R_{i+1} becomes a singleton joining relation after ϕ_i for $i = 1, 2, 3$. Consequently, σ completely solves the query. In order to avoid repetitive details, only the results are given in Table 6.7.

The results in this section are summarized in Table 6.8. It is observed that Algorithm H performs uniformly better than other algorithms.

Table 6.7

Query Costs for Example 5.4

Algorithm	Query Cost
Initial Cost	3950
Hevner and Yao	2966
SDD-1	1646
Cheung	3112
Algorithm H	364

Table 6.8

Summary of Query Cost Comparisons

Examples by

Algorithm	H & Y	Bernstein	Cheung	Example 5.4
Init. Cost	3830	206630	1330	3950
H & Y	1324	4470	1240	2966
SDD-1	756	4128	796	1646
Cheung	1324	2950	865	3112
Algorithm H	478	2711	683	364

6.2 Efficiency of the Algorithm

We have measured the execution time of the program which implements Algorithm H. The execution time of the program is recorded at three points: after reading inputs, after scheduling a sequence of semijoins, and after printing out results. Since reading inputs and printing outputs are related to system I/O and common to other algorithms, only the time taken to schedule a sequence of semijoins is relevant to the efficiency of the algorithm.

The scheduling time for the examples presented in Section 6.1 using Algorithm H is shown in Table 6.9. The efficiency of Algorithm H is mainly achieved by the following factors:

(1) Since the number of blocks is considerably less than the number of semijoins, the block-oriented nature of Algorithm H leads to a significant reduction of search space.

(2) The estimation using the lattice model provides an efficient computation method for the dominant term in Algorithm H.

Table 6.9

Scheduling Time of a Sequence of Semijoins using Algorithm H

Example by	No. Sites	Scheduling Time (Seconds)
Hevner and Yao	4	0.0039
Bernstein et al.	3	0.0027
Cheung	4	0.0043
Example 5.4	5	0.0055

CHAPTER 7

CONCLUSION

The problem of query optimization in distributed relational database systems has been addressed. In order to process a query which references data from multiple sites in a computer network, portions of the database at other sites have to be transferred to the user's site. Due to the rapid increase of computing power in recent years, it has been observed that the delay caused by inter-site data communication has become the more dominant factor in processing a distributed query. Therefore, the main objective in processing a distributed query is the minimization of the inter-site data traffic in a communication network.

The methodology which has been used in this research consists of reducing the referenced relations using a sequence of semijoin operations after initial local processing. The semijoin strategy involves the following subproblems:

(1) Estimation of the size of the relation reduced by each semijoin of a sequence of semijoins.

(2) Design of an algorithm to determine an optimal sequence of semijoins which incurs the minimal

total inter-site data transfer.

The previous semijoin strategies for distributed query optimization either produce erroneous estimations due to an unrealistic assumption or derive sequences of semijoins which do not sufficiently reduce inter-site data transfer.

A query information model has been established which provides a compact represention of a user query. Especially, the concept of a block has been introduced, which plays a central role for developing a model and a solution algorithm for distributed query optimization. From the query information model, necessary variables which describe the database state have been defined.

A mathematical model has been developed to determine an optimal sequence of semijoins which minimizes the total inter-site data flow in processing a distributed query. The net benefit of a semijoin in a sequence of semijoins has been expressed in terms of its contribution in reducing the total amount of inter-site data transfer in processing a distributed query.

The core of the optimization model is a method which accurately estimates the size of an intermediate result of a query. In particular, the assumption that joining attributes are independent during the processing of a query by a sequence of semijoins has been relaxed. The data reduction due to a semijoin has been estimated using conditional probabilities and these reductions due to a sequence of semijoins have been modeled by a lattice.

A structural analysis of the lattice model has been carried out to establish a basis for developing algorithms which make use of the lattice model. A method that systematically generates the lattice has been developed. This method allows us to identify the relationships among the elements in the lattice, thus to construct examples of query processing by a sequence of semijoins. It has been proven that the lattice which models the data reduction is a leveled lattice. This property can be used to reduce the search space when some elements of the lattice are stored and they are searched to retrieve the data associated with those elements.

In distributed query optimization, the computation of the reduction of the set of values of a joining attribute by a semijoin is a dominant term. Therefore, an efficient method for estimating the reduction is crucial in increasing the efficiency of the optimization algorithm. A special structure and a labeling rule of the elements in the lattice have been used to design a recursive algorithm which is essential in computing that reduction. It has been observed that this algorithm generally terminates without any recursion in processing resonable size queries.

When a relation referenced by a query consists of only one joining attribute after initial local processing, this relation can be ignored after an outgoing semijoin from this attribute. In addition, if the block containing this attribute has only two attributes and neither is in the

target list, then both of them can be ignored after the semijoin. Since this situation can cause a chain effect to other relations, an additional reduction of inter-site data transfer can be achieved. This feature has been incorporated into our methodology.

Since the distributed query optimization problem is known to be NP-hard, a heuristic algorithm has been developed to determine a low-cost sequence of semijoins. The algorithm decreases the cost of a query by selecting the low-cost, highly reductive semijoins first.

The main feature of this algorithm is to select and "visit" an unvisited block with the least value of a heuristic cost function defined on the set of blocks until no more unvisited block exists. By "visiting" a block we mean that semijoins are scheduled which go from the attribute with the smallest cardinality in the block towards the one with the largest cardinality. After all the blocks have been visited, the visits are reversed. Several control features have been incorporated in the algorithm to increase the robustness for random input data.

The time complexity of the main features of our algorithm has been analytically derived. It has been proven that the number of sequences of semijoins that can be generated by the main features of our algorithm is $O(n^*|\beta|)$ in the worst case, where β is the set of blocks and n^* is the size of the largest block. The comparisons of the query cost produced by our algorithm with those by the existing

algorithms, which schedule seminjoin strategies for general distributed queries, have been made. The examples in the articles which present existing algorithms have been used as benchmarks. It has been observed that our algorithm performs uniformly better than existing algorithms for those benchmarks.

The algorithm has been implemented in PASCAL. The tests have shown that the scheduling time for a sequence of semijoins for a distributed query which references data from less than or equal to five sites is less than 0.01 seconds when the program is executed by Amdahl 470V/8. It is considered that the efficiency of our distributed query optimization algorithm is mainly achieved by the block-oriented nature of the algorithm and the estimation of data reductions using the lattice model.

APPENDICES

APPENDIX A

Some Concepts of Relational Model

1. Structures

In a relational database, the data is logically arranged in two dimensional tables. Each table corresponds to an entity. Here, a relationship between entities is also considered as an entity. An entity consists of several distinct attributes to describe it. Each row of the table is called a tuple, which corresponds to an occurrence of the entity with a specific value for each attribute. Each column of the table corresponds to the values of a component attribute. The set of all values that an attribute can take is called a domain. Note that many different attributes can take values from the same domain. The table consisting of a set of tuples is called a relation with its name the same as that of the corresponding entity. The number of attributes of a relation is called the degree of the relation. Relations of degree n are called n-ary and the tuples in them are called n-tuples. Hence the relational database consists of a collection of time-varying tabular relations. Figure A.1 illustrates example relations.

2. Relational Algebra

The operators for the manipulation of relations must be defined compatible with the data structure. The definitions presented here are based on [BERN 79, CODD 79].

SUPPLIER

S#	SNAME	CITY
S1	Smith	N.Y.
S2	Jones	L.A.
S3	Clark	L.A.
S4	Adams	N.Y.

SUPPLY

S#	P#	QUANTITY
S1	P1	400
S1	P3	290
S1	P4	240
S1	P5	160
S1	P6	380
S2	P2	300
S2	P3	290
S2	P5	160
S3	P1	200
S3	P2	350
S3	P4	240
S3	P6	300

PART

P#	PNAME	MATERIAL
P1	Nut	Steel
P2	Pipe	Plastic
P3	Screw	Steel
P4	Screw	Aluminium
P5	Bolt	Plastic
P6	Wire	Aluminium

The Examples of Relations

Figure A.1

SELECTION:

SUPPLIER [CITY = 'N.Y.']

S#	SNAME	CITY
S1	Smith	N.Y.
S4	Adams	N.Y.

PROJECTION:

SUPPLY [QUANTITY, P#]

QUANTITY	P#
400	P1
290	P3
240	P4
160	P5
380	P6
300	P2
200	P1
350	P2
300	P6

The Examples of Relational Operations

Figure A.2

JOIN:

SUPPLIER [S# = S#] SUPPLY

S#	SNAME	CITY	S#	P#	QUANTITY
S1	Smith	N.Y.	S1	P1	400
S1	Smith	N.Y.	S1	P3	290
S1	Smith	N.Y.	S1	P4	240
S1	Smith	N.Y.	S1	P5	160
S1	Smith	N.Y.	S1	P6	380
S2	Jones	L.A.	S2	P2	300
S2	Jones	L.A.	S2	P3	290
S2	Jones	L.A.	S2	P5	160
S3	Clark	L.A.	S3	P1	200
S3	Clark	L.A.	S3	P2	350
S3	Clark	L.A.	S3	P4	240
S3	Clark	L.A.	S3	P6	300

SEMIJOIN:

SUPPLIER <S# = S#] SUPPLY

S#	SNAME	CITY
S1	Smith	N.Y.
S2	Jones	L.A.
S3	Clark	L.A.

Figure A.2 continued.

SELECTION

Let A be an attribute of a relation R and D be the domain from which A takes values. For v $\in$ D, the selection of R on A for v, denoted by R[A=v], is defined as {r $\in$ R | r.A=v}

PROJECTION

$R[A_1, A_2, \ldots, A_n]$ is the relation obtained by dropping all columns of R except those specified by $A_1, A_2, \ldots, A_n$ and then dropping redundant duplicate rows.

JOIN

Let A be an attribute of a relation R and B be an attribute of a relation S with A and B defined on the same domain. Then the join of R and S on A and B, denoted by R[A=B]S, is defined as {rs|r $\in$ R and s $\in$ S and r.A=s.B} where rs is a concatenation of r and s.

SEMIJOIN

Let A,B,R and S be the same as in the definition of join. Let A_r be the attributes of R. The semijoin of R by S on A and B, denoted by R<A=B]S, is defined as $(R[A=B]S)[A_r]$.

The selection, join and semijoin can be defined using binary relations other than equality. R[A=B]S contains two identical columns, one derived from A and the other from B. The NATURAL JOIN is the same as JOIN except that redundant columns generated by the join are removed. The examples of relational operations are shown in Figure A.2.

3. Relational Query

A relational query consists of a target list and a qualification clause. A qualification clause is a boolean combination of terms each of which equate two attributes or relate an attribute to a value [ULLM 80]. If the qualification clause is pure conjunctions of terms, the query is called a conjunctive query. The qualification clause specifies qualified tuples from the referenced relations and the target list specifies the attributes to be projected out. The following example shows the formulation of a relational query.

Example A.1

Consider a user request to the database shown in Figure A.1:
For a supplier located in N.Y. who supplies parts made of steel more than 320 units, find the supplier name, the corresponding part name and the supply quantity.

The relational query formulation is:

```
FIND  (SUPPLIER.SNAME, PART.PNAME, SUPPLY.QUANTITY)
WHERE (SUPPLIER.CITY = 'N.Y.')
 AND  (PART.MATERIAL = 'Steel')
 AND  (SUPPLY.QUANTITY > 320)
 AND  (SUPPLIER.S# = SUPPLY.S#)
 AND  (SUPPLY.P# = PART.P#)
```

■

It is clear that a relational query can be answered by applying a proper sequence of selections, projections and joins to the referenced relations. The term equating two

attributes is called a join term because it can be processed by join. Similarly, the term relating an attribute to a value is called a selection term. The attributes in a join term is called joining attributes.

APPENDIX B

Some Definitions from Lattice Theory

The definitions presented here are from [BIRK 67] and [GRAE 78].

Definition A.1: A lattice is a partially ordered set (poset) P any two of whose elements have a g.l.b. or "meet" denoted by $x \wedge y$, and a l.u.b. or "join" denoted by $x \vee y$.

Definition A.2: A bijection $\theta: P \dashrightarrow Q$ from a poset P to a poset Q is an isomorphism if and only if

$$x \leq y \text{ implies } \theta(x) \leq \theta(y)$$

$$\text{and } \theta(x) \leq \theta(y) \text{ implies } x \leq y.$$

Definition A.3: A bijection $\theta: P \dashrightarrow Q$ from a poset P to a poset Q is a dual isomorphism if and only if

$$x \leq y \text{ implies } \theta(x) \geq \theta(y)$$

$$\text{and } \theta(x) \leq \theta(y) \text{ implies } x \geq y.$$

Definition A.4: A lattice L is complete when each of its subsets has a g.l.b. and l.u.b. in L.

Definition A.5: A lattice L with the greatest element I and the least element O is complemented if for all $x \in L$ there exists $y \in L$ such that

$$x \wedge y = O \text{ and } x \vee y = I.$$

Definition A.6: A lattice L is distributive if and only if $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ for all $x,y,z \in L$.

Definition A.7: A boolean lattice is a complemented and distributive lattice.

Definition A.8: A Sublattice of a lattice L is a subset X of L such that $a \in X$, $b \in X$ imply $a \wedge b \in X$ and $a \vee b \in X$.

Definition A.9: A join-semilattice is a poset P such that $x \vee y \in P$ for all $x,y \in P$. A meet-semilattice is dually defined.

Definition A.10: A chain is a poset P such that $x \leq y$ or $y \leq x$ for all $x,y \in P$.

BIBLIOGRAPHY

BIBLIOGRAPHY

[ADIB 78] Adiba, M. et al., "Issues in distributed database management system: a technical overview," Proc. of the Intl. Conf. on VLDB, West-Berlin, Sep. 1978, pp. 89-110.

[BAKE 74] Baker, K.R., Introduction to sequencing and scheduling, John Wiley, 1974.

[BERN 79] Bernstein, P.A. and Goodman, N., "The theory of semi-joins," Technical Report No. CCA-79-27, Computer Corporation of America, November 1979.

[BERN 81] Bernstein, P.A. et al., "Query processig in a system for distributed databases (SDD-1)," ACM TODS, Vol. 6, No. 4, Dec. 1981, pp. 602-625.

[BIRK 67] Birkhoff, G., Lattice theory, 3rd Edition, American Mathematical Society, RI., 1967.

[BREI 70] Breipohl, A.M., Probabilistic systems analysis, John Wiley, 1970.

[CASE 72] Casey, R. G., "Allocation of copies of files in an information network," AFIPS Conference Proceedings, Vol. 41, 1972, pp. 617-625.

[CASE 73] Casey, R. G., "Design of tree networks for distributed data," AFIPS Conference Proceedings, Vol. 42, 1973, pp. 251-257.

[CCA 80a] Computer Corporation of America, "A distributed database management system for command and control applications: Final technical report - Part I," Technical Report No. CCA-80-03, Jan. 1980.

[CCA 80b] Computer Corporation of America, "A distributed database management system for command and control applications: Final technical report - Part II," Technical Report No. CCA-80-04, Jan. 1980.

[CHAM 77] Champine, G.A., "Six approaches to distributed databases," Datamation, Vol. 23, No. 5, Technical Publishing Company, Barrington, Ill., May 1977, pp. 69-72.

[CHAN 77] Chandy, K.M., "Models of distributed systems," Proc. of the Intl. Conf. on VLDB, Tokyo, Oct. 1977, pp. 105-120.

[CHEU 82] Cheung, T., "A method for equijoin queries in distributed relational databases," IEEE Trans. on Computers, Aug. 1982, pp. 746-751.

[CHU 69] Chu, W.W., "Optimal file allocation in a multiple computer system," IEEE Trans. on Computers, Oct. 1969, pp. 885-889.

[CHU 79] Chu, W.W. and Hurley, p., "A model for optimal query processing for distributed databases," Digest of Papers, COMPCON 79 Spring.

[CHU 82] Chu, W.W. and Hurley, P., "Optimal query processing for distributed database systems," IEEE Trans. on Computers, Vol. C-31, No. 9, Sep. 1982, pp. 835-850.

[CHUN 82] Chung, C. W. and Irani, K. B., "A methodology for query optimization in distributed database systems," Database Engineering: a quarterly bulletin of the IEEE Computer Society Technical Committee on Database Engineering, Sep. 1982, pp. 19-23.

[CODD 70] Codd, E.F., "A relational model of data for large shared data banks," Comm. ACM, Vol, 13, June 1970, pp. 377-387.

[CODD 71] Codd, E.F., "Relational completeness of database sublanguages," In Database Systems, Courant Computer Science Symposia Series, Vol. 6, Englewood Cliffs, N.J., Prentice-Hall, 1971, pp. 65-98.

[CODD 79] Codd, E.F., "Extending the database relational model to capture more meaning," ACM TODS, 1979, pp. 29-52.

[DATE 77] Date, C.J., An introduction to database systems, 2nd Edition, Addison-Wesley, MA., 1977.

[DAVE 70] Davenport, W.B., Probability and random processes, McGraw-Hill, 1970.

[DAYA 79] Dayal, U. and Bernstein, P.A., "The fragmentation problem: lossless decomposition of relations into files," Technical Report No. CCA-78-13, Computer Corporation of America, Nov. 1978.

[EPST 80] Epstein, R. and Stonebraker, M., "Analysis of distributed database processing strategies," Proc. of the Intl. Conf. on VLDB, 1980, pp. 92-101.

[EPST 78] Epstein, R., Stonebraker, M., Wong, E., "Distributed query processing in a relational database system," Proc. 1978 ACM SIGMOD Conference, June 1978, pp. 169-180.

[ESWA 74] Eswaran, K.P., "Placement of records in a file and file allocation in a computer netwwork," IFIP Conference Proceedings, Aug. 1974, pp. 304-307.

[GARE 79] Garey, M.R. and Johnson, D.S., Computers and intractability, W.H. Freeman and Company, CA., 1979.

[GOUD 81] Gouda, M.G. and Dayal, U., "Optimal semijoin schedules for query processing in local distributed database systems," Proc. 1981 ACM SIGMOD Conference, May 1981, pp.164-173.

[GRAE 78] Graetzer, G., General lattice theory, stuttgart: Birkhaeuser, 1978.

[HEVN 78] Hevner, A.R. and Yao, S.B., "Query processing on a distributed database," Proc. 1978 Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, CA., Aug. 1978, pp. 91-107.

[HEVN 79a] Hevner, A.R., The optimization of query processing on distributed database systems, Ph.D. dissertation, Purdue University, 1979.

[HEVN 79b] Hevner, A.R. and Yao, S.B., "Query processing in distributed database systems," IEEE Trans. on Software Eng., Vol. SE-5, No.3, May 1979, pp. 177-187.

[HORO 78] Horowitz, E. and Sahni, S., Fundamentals of computer algorithms, Computer Science Press, MD., 1978.

[IRAN 82] Irani, K.B. and Khabbaz, N.G., " A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems," IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 419-434.

[KLEI 76] Kleinrock, L., Queueing systems, Vol.II: Computer Applications, John Wiley, 1976.

[LEVI 75] Levin, K.D. and Morgan, H.L., "Optimizing distributed databases -A framework for research," AFIPS Conference Proceeding, Vol. 44, 1975, pp. 473-478.

[LIEN 78] Lien, Y.E. and Ying, J.H., "Design of a distributed entity-relationship database system," Proc. COMPSAC 78.

[MAHM 76] Mahmoud, S. and Riordan, J.S., "Optimal allocation

of resources in distributed information networks," ACM TODS, Vol. 1, No.1, March 1976, pp. 66-78.

[MART 77] Martin, J., Computer database organization, 2nd Edition, Prentice-Hall, N.J., 1977.

[MURT 76] Murty, K.G., Linear and combinatorial programming, John Wiley, 1976.

[NOLA 73] Nolan, R.L., "Computer databases: the future is now," Harvard Business Review: Sept.-Oct. 1973, pp.98-114.

[PAIK 79] Paik, I. and Delobel, C., "A strategy for optimizing the distributed query processing," The 1st Intl. Conf. on Distributed Computing Systems, Huntsville, AL., Oct. 1979, pp. 686-698.

[PELA 79] Pelagatti, G. and Schreiber, F.A., "Evaluation of transmission requirements in distributed database access," Proc. 1979 ACM SIGMOD Conference, May 1979, pp. 102-108.

[RAMA 79] Ramamoorthy, C.V. and Wah, B.W., "The placement of relations on a distributed relational database," The 1st Intl. Conf. on Distributed Computing Systems, Huntsville, AL., Oct. 1979, pp. 642-650.

[ROTH 77] Rothnie, J.B. and Goodman, N., "A survey of research and development in distributed database management," Proc. of the Int. Conf. on VLDB, Tokyo, Oct. 1977, pp. 48-62.

[ROTH 80] Rothnie, J.B. et al., "Introduction to a system for distributed Databases (SDD-1)," ACM TODS, Vol.5, No.1, March 1980, pp.1-17.

[STON 77] Stonebraker, M. and Neuhold, E., "A distributed database version of INGRES," Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Lab., Univ. of California, Berkeley, CA., May 1977, pp. 19-36.

[ULLM 80] Ullman, D.J., Principles of database systems, Computer Science Press, MS., 1980.

[WONG 76] Wong, E. and Youssefi, K., "Decomposition-A strategy for query processing," ACM TODS, Vol. 1, No. 3, Sept. 1976, pp. 223-241.

[WONG 77] Wong, E., "Retrieving dispersed data from SDD-1: A system for distributed databases," Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Lab., Univ. of California,

Berkeley, CA., May 1977, pp 217-235.

[YAO 77] Yao, S.B., "Approximating block accesses in database organizations," Comm. ACM. Vol. 20, No. 4, April 1977, pp. 260-261.

[ZANI 79] Zaniolo, C., "Design of relational views over network schemes," Proc. 1979 ACM SIGMOD Conference, May 1979, pp. 179-190.