A FORMAL APPROACH TO THE VERTICAL PARTITIONING PROBLEM IN DISTRIBUTED DATABASE DESIGN

By

JEYAKUMAR MUTHURAJ

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

1992

ACKNOWLEDGMENTS

First of all, I would like to express my deep gratitude to my advisor, Dr. Sharma Chakravarthy, for giving me the opportunity to work on the challenging *Distributed Database Design* project and for his continual encouragement and support. I would like to thank my supervisory committee co-chair Dr. Shamkanth Navathe for his invaluable ideas and criticisms. I would also like to thank Dr. Ravi Varadarajan for his useful suggestions and discussions.

Many thanks are due to Mrs. Sharon Grant for all the help and cooperation in time of need . I am grateful to Mrs. Marlene Hughes for her help and support. I will also take this opportunity to thank all the graduate students at the Database center for their help, moral support and friendship.

Finally, I would like to express my gratitude to my parents for their unlimited amounts of patience, prayer and encouragement.

This work was supported by the NSF grant IRI-9011216 and IRI-8716798.

TABLE OF CONTENTS

Abstract of Thesis
Presented to the Graduate School of the University of Florida
in Partial Fulfillment of the Requirements for the
Degree of Master of Science

A FORMAL APPROACH TO THE VERTICAL PARTITIONING PROBLEM IN DISTRIBUTED DATABASE DESIGN.

By

Jeyakumar Muthuraj

August, 1992

Chairman: Dr. Sharma Chakravarthy
Major Department: Computer and Information Sciences

The design of distributed databases is an optimization problem requiring solutions to several interrelated problems: data fragmentation, allocation, and local optimization. Each problem can be solved with several different approaches thereby making the distributed database design a very difficult task.

Although there is a large body of work on the design of data fragmentation, most of them are either *ad hoc* solutions or formal solutions for special cases (e. g., binary vertical partitioning). In this paper, we address the problem of n-ary vertical partitioning problem and derive an objective function that generalizes and subsumes earlier work. The objective function derived in this paper is being used for developing heuristic algorithms that can be shown to satisfy the objective function. The objective function is also being used for comparing previously proposed algorithms for vertical partitioning. We first derive an objective function that is suited to distributed transaction processing and then show how it can be extended to include additional

information, such as transaction types, different local and remote accessing costs and replication. Finally, we indicate the current status of implementation.

CHAPTER 1
INTRODUCTION

The design of distributed databases is an optimization problem requiring solutions to several interrelated problems: data fragmentation, allocation, and local optimization. Each problem phase can be solved with several different approaches thereby making the distributed database design a very difficult task. Traditionally database design has been heuristic in nature. Although the metric being optimized is not stated quantitatively, it is implicitly assumed to be the *processing cost* for a given set of *important* transactions that constitute the bulk of transaction load for the given database.

Figure 1.1 gives an outline of the overall distributed database design methodology [4]. Distributed database design deviates from conventional non-distributed database design only in the distribution aspect which is highlighted by the box titled distribution design in figure 1.1. The distribution design involves data acquisition, partitioning of the database, allocation and replication of the partitions and local optimization. Partitioning of the database is done in several ways: vertical, horizontal, and hybrid (also called mixed). Our long-term objective is to develop a distributed database design testbed in which different algorithms for various components of distribution design can be mixed and matched. This work is a first step in that direction and addresses the partitioning (or fragmentation) problem.

In this paper, we delimit our discussion to one of the data fragmentation problems, namely the vertical partitioning problem. Vertical Partitioning (also called attribute partitioning) is a technique that is used during the design of a database to improve the performance of transactions [20]. In vertical partitioning, attributes of a relation

R[1] are clustered into non-overlapping[2] groups and the relation R is projected into fragment relations according to these attribute groups. In distributed database systems, these fragments are allocated among the different sites. Thus the objective of vertical partitioning is to create vertical fragments of a relation so as to minimize the cost of accessing data items during transaction processing. If the fragments closely match the requirements of the set of transactions provided, then the transaction processing cost could be minimized. Vertical partitioning also has its use in partitioning individual files in centralized databases, and dividing data among different levels of memory hierarchies etc. [20, 25]. In the case of distributed database design, transaction processing cost is minimized by increasing the local processing of transactions (at a site) as well as by reducing the amount of accesses to data items that are not local. The aim of vertical partitioning technique (and in general data partitioning techniques) is to find a partitioning scheme which would satisfy the above objective.

It should be noted that the problem of partitioning can be addressed at various levels of detail by taking additional information into consideration. Figure 1.1 clearly distinguishes various levels and a feedback path is provided to refine the outcome of the earlier levels if it does not suit the objectives of the next level. In this thesis, we are taking only transaction information as input to keep the problem manageable. In essence, the global optimization problem (which includes a large number of parameters and a very complex metric) is partitioned into several smaller optimization problems to reduce the search space and the complexity of each problem. Other detailed information need to be considered on the outcome of this stage in order to obtain a design at the physical level.

[1]For most of our discussion it does not matter whether R is a Universal relation or a relation that is in some normal form as long as key attributes are identified. In the chapter on extensions, we discuss how we can take that information into account.

[2]Overlapping partitions (of non-primary key attributes) may also be considered when availability is an important criterion. This is discussed in the chapter on extensions.

Several vertical partitioning algorithms have been proposed in the literature. Hoffer and Severance [11] measure the affinity between pairs of attributes and try to cluster attributes according to their pairwise affinity by using the *bond energy algorithm* (BEA) [18]. Hammer and Niamir [10] use a file design cost estimator and a heuristic to arrive at a "bottom up" partitioning scheme. Navathe, et al [20] extend the BEA approach and propose a two phase approach for vertical partitioning. Cornell and Yu [6] apply the work of Navathe [20] to physical design of relational databases. Ceri, Pernici and Wiederhold [5] extend the work of Navathe [20] by considering it as a 'divide' tool and by adding a 'conquer' tool. Navathe and Ra [22] construct a graph-based algorithm to the vertical partitioning problem where the heuristics used includes an intuitive objective function which is not explicitly quantified. In addition to these vertical partitioning algorithms, there are many data clustering techniques [13], traditionally used in pattern recognition and statistics, some of which can be adapted to partitioning of a database. These data clustering algorithms include Square-error clustering [13], Zahn's clustering [32], Nearest-neighbor clustering [17] and Fuzzy [13]clustering.

The partitioning algorithms mentioned above use some heuristics to create fragments of a relation. The input to most of these algorithms is an Attribute Usage Matrix *(AUM)*. AUM is a matrix which has attributes as columns, and transactions as rows and the access frequency of the transactions as values in the matrix. Most of the earlier data fragmentation algorithms use an Attribute Affinity Matrix (*AAM*) derived from the AUM provided as input. An AAM is a matrix in which for each pair of attributes, the sum total of frequencies of transactions accessing that pair of attributes *together* is stored. The results of the different algorithms are sometimes different even for the same attribute affinity matrix indicating that the objective

functions used by these algorithms are different. Most of the proposed vertical partitioning algorithms do not have an objective function to evaluate the "goodness" of partitions that they produce. Also, there is no common criterion or objective function to compare and evaluate the results of these vertical partitioning algorithms.

1.0.1 Contributions

This thesis makes several contributions to the problem of data fragmentation in general and the design of vertical partitioning in particular. Specifically:

1. We have, perhaps for the first time, studied the applicability of some data clustering algorithms for distributed database design[3] proposed in areas such as pattern classification, statistics etc., [13], [32], [17], to data fragmentation problem. In fact, we start from one such objective function proposed for data clustering and modify and extend it to the specific problem at hand.

2. We have formulated an objective function for n-ary partitions, with two components that provide the desirable behavior for minimizing transaction processing cost.

3. Finally, we are using the approach of formulating an objective function (termed Partition Evaluator in this thesis) before developing (heuristic) algorithms for the partitioning problem. This approach enables us to study the properties of algorithms with respect to an agreed upon objective function, and also to compare different algorithms for "goodness" using the same criteria. The objective function formulated in this thesis is a step in this direction. Moreover, the objective function derived in this thesis can be easily extended to include

[3]Schkolnik [25] uses data clustering techniques for partitioning a hierarchical structure for an IMS database using detailed cost information which is different from the problem addressed in this thesis.

additional information (e. g., query types – retrieval/update, allocation information about the partitions, remote processing cost, and the transaction usage pattern at any particular site). Some of these extensions are discussed at the end of the thesis.

Our long-term objective is to either extend this objective function or to develop new objective functions to take into account additional information pertaining to replication, storage, and transmission costs that are critical to a distributed environment. However, we view this work as filling a void that currently exists even at the conceptual level.

1.0.2 Thesis Organization

The organization of the thesis is as follows. Chapter 2 discusses previous related work on data fragmentation and explains three different vertical partitioning algorithms which will be evaluated using the Partition Evaluator that we will develop. In Chapter 3 we will explain a few data clustering algorithms used in other areas of applications and how to adapt these algorithms for data fragmentation. Chapter 4 summarizes the need for the development of our Partition Evaluator(PE) and derive the same with appropriate characteristics for distributed databases. In Chapter 5, we illustrate the use of our Partition Evaluator with an example. We show the actual behavior of the PE and compare it with the expected behavior. In Chapter 6, we discuss the extensions to the Partition Evaluator and the implementation of a prototype database design testbed. Chapter 7 concludes the thesis with suggestions on how to extend the Partition Evaluator by incorporating additional information available at the design stage. It also includes future work.

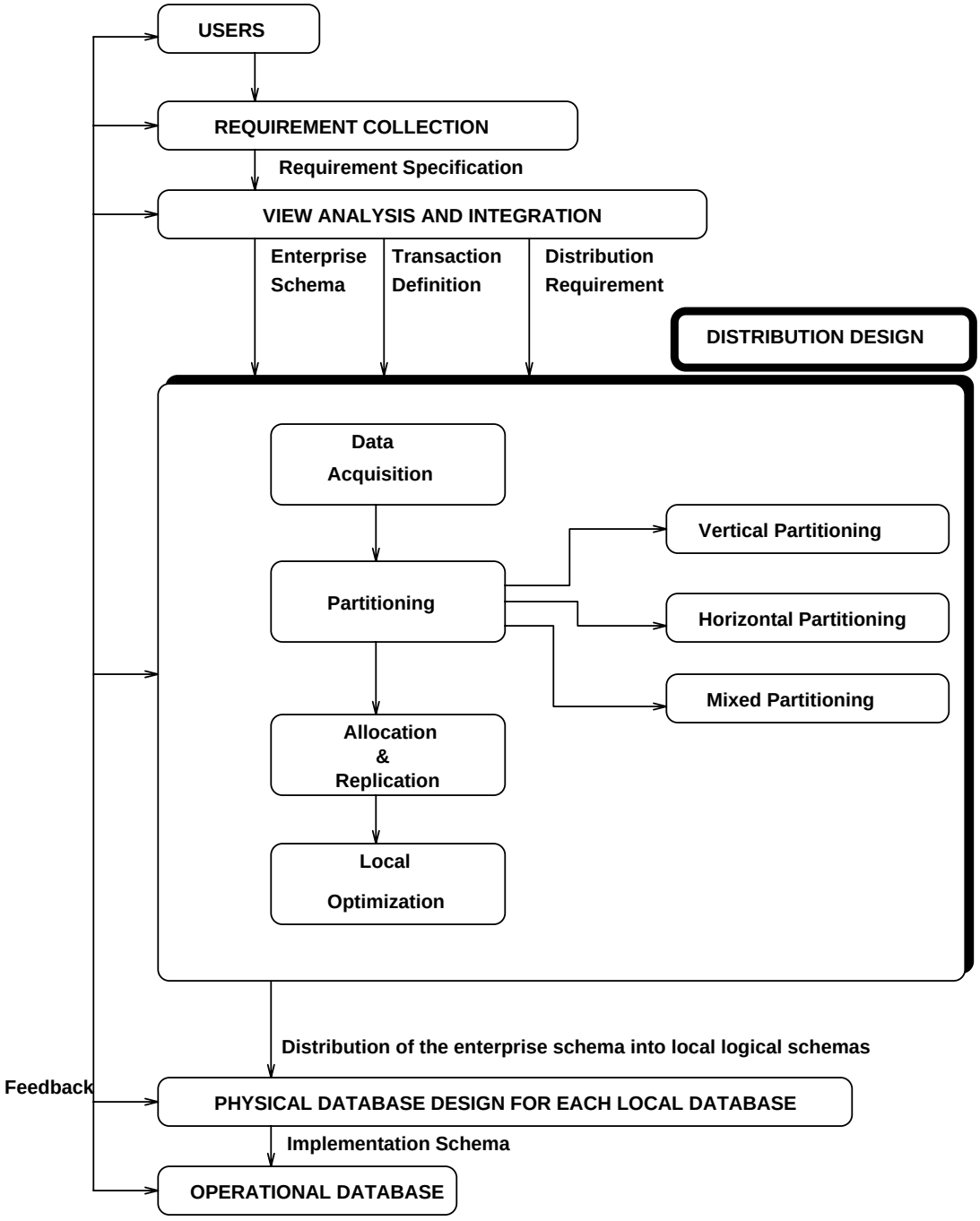

Figure 1.1. Distributed Database Design Methodology

CHAPTER 2
VERTICAL PARTITIONING ALGORITHMS

2.1 Previous Related Work

The concept of using fragmentation of data as a means of improving the performance of a database management system has often appeared in the literature on file design and optimization. Attribute partitioning and attribute clustering have studied earlier by Day [7], Seppala [26], Osman [24], Yue and Wong [31], Benner [3], Alsberg [1], Babad [2], Stocker and Dearnley [28],[8], Kennedy [15], [14], Eisner and Severance [9], March and Severance [19], Hoffer and Severance [11], Hammer and Niamir [10], Navathe et al. [20], and Navathe and Ra [22]. Stocker and Dearnley [28],[8] discuss the implementation of a self-reorganizing database management system that carries out attribute clustering. They also show that in a database management system where storage cost is low compared to the cost of accessing the subfiles, it is beneficial to cluster the attributes, since the increase in storage cost will be more than offset by the saving in access cost. Kennedy [15], [14] considers a mathematical model of attribute partitioning where each attribute a_i is of known length, and has probability p_i of being requested by a query. The joint probability that attributes a_i and a_j are requested by the same query is assumed to be $p_i p_j$. A cost function based on this assumption is derived, which reflects the expected amount of data that must be transmitted in order to answer to query. The objective here is to choose a partition such that this cost function is minimized. Hoffer [12] developed a non-linear, zero-one program which minimizes a linear combination of storage, retrieval and update costs, with capacity constraints for each file. Babad [2] formulated a less

restrictive vertical partitioning problem for variable length attributes as a non-linear zero-one program.

In the work of Eisner and Severance [9], a file can be partitioned into two subfiles: a primary and secondary subfile. Two forms of cost function are used in this approach. The first function is the sum of storage charges for subtuples in the primary subfile, and the cost of accessing all the subtuples residing in the secondary subfile. The second function is nonlinear, and measures the total costs of access, transfer, and storage for subtuples in both primary and secondary subfiles. The search cost for finding the optimal solution for the general nonlinear objective function is even higher than for the simplified linear cost function. The limitation of this approach is that at most two subfiles are allowed and the cost associated with processing a query is taken to be the cost of accessing the whole (primary or secondary) subfile in its entirety rather than the cost of retrieving just those subtuples of the subfile that are really needed to answer the query. Also the cost of finding the optimal partition using the linear objective cost function grows in the cube of the sum of the number of attributes and the number of queries; this cost is very large for practical purposes [23]. March and Severance [19] extended this model to incorporate block factors for both primary and secondary memories. The page sizes in the primary and the secondary subfiles are not necessarily the same, but the constraint is imposed that the sum of the primary subfile page size and the secondary subfile page size is constant. The nonlinear objective cost function they derive not only depends on how the attributes are partitioned among the two subfiles, but also on the page sizes selected for each of the primary and secondary subfiles. The model of March and Severance has the disadvantage that it does not contain an accurate model of the cost of accessing subtuples that are selected in queries. Rather, the primary and secondary subfiles are assumed to be accessed in their entirety whenever any of their attributes are

requested by a query. Using integer programming techniques March and Severance obtained the optimal partition for their model. Hoffer and Severance [11] grouped the attributes of a relation based on the extent to which they were used together (measured the "affinity between pairs of attributes"). This clustering of attributes based on their pairwise affinity was done using the bond energy algorithm (BEA). The BEA produced matrix in which an cost function was minimized for the entire matrix using the affinity attribute matrix. Clusters of objects can be identified such that every pair of objects within the cluster carries a large measure of similarity, and every pair of objects across cluster boundaries carries a small measure of similarity. They provide attributes as objects to the clustering algorithm. They left the creation of partitions to the subjective evaluation of the designer.

Hammer and Niamir [10] developed two heuristics, grouping and regrouping, and used them to perform the partitioning. The grouping heuristic starts by initially assigning each attribute to a trivial partition and generates all partitions that can be obtained by grouping together pairs of blocks in the trivial partition. The heuristic then evaluates all the generated partitions with the file cost estimator, and finds the partition whose performance cost is the least of all the generated partitions. On each iteration all possible grouping of these partitions is considered and the one with maximum improvement is chosen as the candidate grouping for the next iteration. During regrouping, attributes are moved between partitions to achieve any additional improvements possible.

Navathe et al [20] use a two step approach for vertical partitioning. In the first step, they use the given input parameters in the form of an attribute usage matrix to construct the attribute affinity matrix on which clustering is performed. After clustering, an empirical objective function is used to perform iterative binary partitioning. In the second step, estimated cost factors reflecting the physical environment

of fragment storage are considered for further refinement of the partitioning scheme. Cornell and Yu [6] propose an algorithm as an extension of Navathe et al [20] approach which decreases the number of disk accesses to obtain an optimal binary partitioning. This algorithm uses specific physical factors such as number of attributes, their length and selectivity, cardinality of the relation etc. Navathe and Ra [Nava 89] present a graph-based approach to the vertical partitioning problem. This approach is based on the observation that all pairs of attributes in a fragment must have high "within fragment affinity" but low "between fragment affinity". Reduction in complexity is claimed as the main advantage of their approach.

2.2 Input to the Vertical Partitioning Algorithms

The input to the Vertical Partitioning algorithms that we are going to explain is an Attribute Usage Matrix (AUM). An example AUM is given below.

Input: **Attribute Usage Matrix**

$Trans.\backslash Attrs.$	1	2	3	4	5	6	7	8	9	10
$T1$	25	0	0	0	25	0	25	0	0	0
$T2$	0	50	50	0	0	0	0	50	50	0
$T3$	0	0	0	25	0	25	0	0	0	25
$T4$	0	35	0	0	0	0	35	35	0	0
$T5$	25	25	25	0	25	0	25	25	25	0
$T6$	25	0	0	0	25	0	0	0	0	0
$T7$	0	0	25	0	0	0	0	0	25	0
$T8$	0	0	15	15	0	15	0	0	15	15

Algorithms such as Bond Energy Algorithm, Binary Vertical Partitioning Algorithm and Ra's Algorithm use the Attribute Affinity Matrix (AAM) formed from the AUM. Attribute affinity measures the bond between two attributes of a relation according to how they are accessed by applications. Attribute affinity between

Attribute	1	2	3	4	5	6	7	8	9	10
1	75	25	25	0	75	0	50	25	25	0
2	25	110	75	0	25	0	60	110	75	0
3	25	75	115	15	25	15	25	75	115	15
4	0	0	15	40	0	40	0	0	15	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	15	40
7	50	60	25	0	50	0	85	60	25	0
8	25	110	75	0	25	0	60	110	75	0
9	25	75	115	15	25	15	25	75	115	15
10	0	0	15	40	0	40	0	0	15	40

Figure 2.1. Attribute Affinity Matrix

attributes i and j is defined as

$$Aff_{ij} = \sum_{t=1}^{T} q_{t,ij} \tag{2.1}$$

where $q_{t,ij}$ is the number of accesses of transaction t referencing both attributes i and j.

So for the AUM given above the attribute affinity matrix is given in figure 2.1

Attribute Affinity Matrix

$Attrs.\backslash Attrs.$	1	2	3	4	5	6	7	8	9	10
$A1$	75	25	25	0	75	0	50	25	25	0
$A2$	25	110	75	0	25	0	60	110	75	0
$A3$	25	75	115	15	25	15	25	75	115	15
$A4$	0	0	15	40	0	40	0	0	15	40
$A5$	75	25	25	0	75	0	50	25	25	0
$A6$	0	0	15	40	0	40	0	0	15	40
$A7$	50	60	25	0	50	0	85	60	25	0
$A8$	25	110	75	0	25	0	60	110	75	0
$A9$	25	75	115	15	25	15	25	75	115	15
$A10$	0	0	15	40	0	40	0	0	15	40

2.3 Bond Energy Algorithm

The Bond Energy Algorithm [18] is used to group the attributes of a relation based on the attribute affinity values in AAM. It is considered appropriate for the following reasons [11]:

- It is designed specifically to determine groups of similar items as opposed to a linear ordering of the items. (ie. it clusters the attributes with larger affinity values together, and the ones with smaller values together).
- The final groupings are insensitive to the order in which items are presented to the algorithm.
- The AAM is symmetric, and hence allows a pairwise permutation of rows and columns, which reduces complexity.
- Because of the definition of Aff_{ij}, the initial AAM is already semiblock diagonal, in that each diagonal element has a greater value of any element along the same row or column.
- The computation time of the algorithm is reasonable. $O(n^2)$, where n is the number of attributes.

This algorithm takes as input the attribute affinity matrix, permutes its rows and columns, and generates a *clustered affinity matrix (CAM)*. The permutation is done in such a way to *maximize* the following *global affinity measure (AM)*.

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} Aff_{i,j} \left[Aff_{i,j-1} + Aff_{i,j+1} + Aff_{i-1,j} + Aff_{i+1,j} \right] \qquad (2.2)$$

where $Aff_{0,j} = Aff_{i,0} = Aff_{n+1,j} = Aff_{i,n+1} = 0$

The last set of conditions takes care of the cases where an attribute is being placed in CAM to the left of the leftmost attribute or to the right of the rightmost attribute during column permutations, and prior to the topmost row and following the last row during row permutations. Before explaining the algorithm we have to define some

more quantities. Let us define the *bond* between two attributes i and j as

$$bond_{ij} = \sum_{z=1}^{n} Aff_{zi} Aff_{zj} \tag{2.3}$$

The *net contribution* to the global affinity measure of placing the attribute k between i and j is

$$cont_{ikj} = 2 * bond_{ik} + 2 * bond_{kj} - 2 * bondij \tag{2.4}$$

Generation of the Clustered Affinity Matrix is done in three steps:

Initialization: Place and fix one of the columns of AAM arbitrarily into CAM.

Iteration: Pick each of the remaining n-i columns (where i is the number of columns already placed in CAM) and try to place them in the remaining i+1 positions in the CAM matrix. Choose the placement that makes the greatest contribution to the global affinity measure described above. Continue this until no more columns remain to be placed.

Row Ordering: Once the column ordering is determined, the placement of the rows should also be changed so that their relative positions match the relative positions of the columns [29].

When the CAM is big, usually more than two clusters are formed and there are more than one candidate partition. The result of applying Bond Energy Algorithm is shown in figure 2.2 The following algorithm extends bond energy algorithm to identify all the clusters in the CAM matrix.

2.4 Binary Vertical Partitioning

The Bond Energy Algorithm determines an ordering of attributes, but it is still left to the subjective judgment of the designer to decide how to "clump" the attributes together to form fragments. Similarity of pairs of attributes can be inadequate if the similarity among larger groups of attributes is not taken into account. Navathe

Attributes	5	1	7	2	8	3	9	10	4	6
5	75	75	50	25	25	25	25	0	0	0
1	75	75	50	25	25	25	25	0	0	0
7	50	50	85	60	60	25	25	0	0	0
2	25	25	60	110	110	75	75	0	0	0
8	25	25	60	110	110	75	75	0	0	0
3	25	25	25	75	75	115	115	15	15	15
9	25	25	25	75	75	115	115	15	15	15
10	0	0	0	0	0	15	15	40	40	40
4	0	0	0	0	0	15	15	40	40	40
6	0	0	0	0	0	15	15	40	40	40

Figure 2.2. Clustered Affinity Affinity Matrix

et al. [20] extended the results of Hoffer and Severance [11] by giving algorithms to quantitatively "clump" the attributes together and by taking into account "blocks" of attributes with similar properties. The approach taken in this algorithm is splitting rather than grouping. The rationale behind this approach is that the "optimal" solution, is much closer to the group composed of all attributes, assumed to be the starting point, than to groups that are single attribute partitions. The objective of splitting activity is to find sets of attributes that are accessed solely, or for the most part, by distinct sets of applications. The binary vertical partitioning algorithm uses the clustered affinity matrix to partition an object into two non-overlapping fragments. Assume that a point x is fixed along the main diagonal of the clustered AA matrix, as shown in figure 2.3 The point x defines two blocks U (for "upper") and L (for "lower"). Each block defines a vertical fragment given by the set of attributes in that block.

Let A_t be the set of attributes used by transaction t defined as follows:

$$A_t = (i|q_{ti} > 0) \tag{2.5}$$

Attributes	5	1	7	2	8	3	9	10	4	6
5	75	75	50	25	25	25	25	0	0	0
1	75	75	50	25	25	25	25	0	0	0
7	50	50	85	60	60	25	25	0	0	0
2	25	25	60	110	110	75	75	0	0	0
8	25	25	60	110	110	75	75	0	0	0
3	25	25	25	75	75	115	115	15	15	15
9	25	25	25	75	75	115	115	15	15	15
10	0	0	0	0	0	15	15	40	40	40
4	0	0	0	0	0	15	15	40	40	40
6	0	0	0	0	0	15	15	40	40	40

Figure 2.3. Partitioned Attribute Affinity Matrix

Using A_t, it is possible to compute the following sets:

$$T = (t|t\ is\ a\ transaction)$$
$$LT = (t|A_t \subseteq L)$$
$$UT = (t|A_t \subseteq U)$$
$$IT = T - (LT \cup UT)$$

T represents the set of all transactions. LT and UT represent the set of transactions that "match" the partitioning, as they can be entirely processed using attributes in the lower or upper block, respectively; IT represents the set of transactions that needs to access both fragments.

$$CT = \sum_{t \epsilon T} q_t$$
$$CL = \sum_{t \epsilon LT} q_t$$
$$CU = \sum_{t \epsilon UT} q_t$$
$$CI = \sum_{t \epsilon IT} q_t$$

CT counts the total number of transaction accesses to the considered object. CL and CU count the total number of accesses of transactions that need only one fragment; CI counts the total number of accesses of transactions that need both fragments. Totally $n-1$ possible locations of point x along the diagonal is considered, where n is the size of the input matrix (ie. the number of attributes). A non-overlapping partition is obtained by selecting the point x along the diagonal such that the following objective function z is maximized:

$$max\ z = CL * CU - CI^2 \tag{2.6}$$

The partitioning that corresponds to the maximal value of the z function is accepted if z is positive, and is rejected otherwise. The objective function shown above comes from an empirical judgment of what should be considered a "good" partitioning. The function is increasing in CL and CU and decreasing in CI. For a given value of CI, it selects CL and CU in such a way that the product $CL * CU$ is maximized. This results in selecting values for CL and CU that are as nearly equal as possible. Thus the above z function will produce fragments that are "balanced" with respect to the transaction load. This algorithm has the disadvantage of not being able to partition an object by selecting out an embedded "inner" block. This disadvantage can be avoided by using the procedure SHIFT, which moves the leftmost column of the AAM to the extreme right, and the topmost row of the matrix to the bottom. SHIFT is called a total of n times, so that every diagonal block gets the opportunity of being brought to the upper left corner in the matrix. When the SHIFT procedure is used, the complexity of the algorithm increases by factor n.Experience has shown that the use of the SHIFT procedure improves the solution of the binary vertical partitioning problem in several cases. Designing an n-way partitioning is possible

but computationally expensive. The solution is to recursively apply the binary partitioning algorithm to each of the fragments obtained during the previous iteration.

2.5 Graph-based Vertical Partitioning Algorithm

The input to this algorithm is the Attribute Affinity Matrix. In previous approaches, a clustering algorithm is applied to the AAM. In this approach the AAM is considered as a complete graph called the affinity graph in which an edge value represents the affinity between the two attributes. Then, forming a linearly connected spanning tree, the algorithm generates all meaningful fragments in one iteration by considering a cycle as a fragment. A "linearly connected" tree has only two ends. Figure 2.4 shows the affinity graph corresponding to the affinity matrix of figure 2.1. A note about the attributes: in this proposed technique as well as in the previous techniques, the set of attributes considered may be

(a) the universal set of attributes in the whole database.

(b) the set of attributes in a single relation.

By using (a), the fragments generated may be interpreted as relations or record types. By using (b), fragments of a single relation are generated.

In the previous approaches, they apply a clustering algorithm to the AAM. In this approach, however, the AAM is considered to be a complete graph called *affinity graph* in which an edge value represent the affinity between the two attributes. Then, forming a linearly connected spanning tree, the algorithm generates all meaningful fragments in one iteration by considering a cycle as a fragment.

The major advantages of the proposed method over that in [20] are as follows:

(1) There is no need for iterative binary partitioning. The major weakness of iterative binary partitioning is that at each step two new problems are generated increasing the complexity.

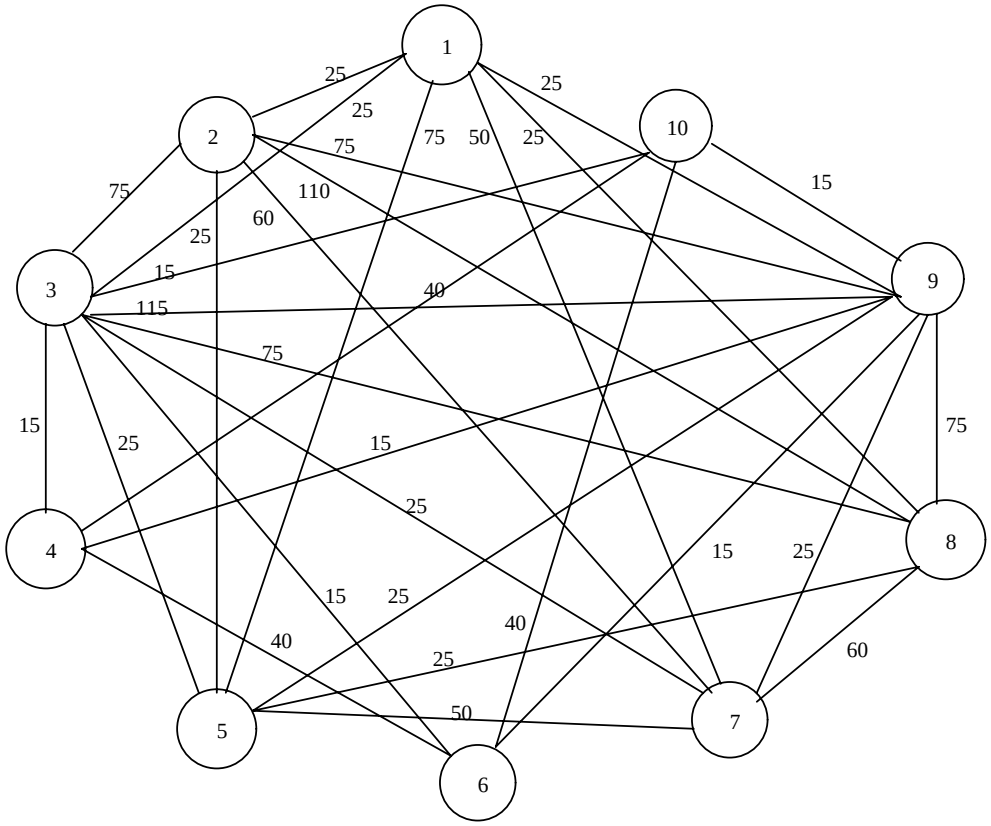

Figure 2.4. Affinity Graph

(2) The method obviates the need for using any empirical objective functions as in [20].

(3) The method requires no complementary algorithms such as the SHIFT algorithm of [20].

(4) The complexity of the algorithm is $O(n_2)$, better than the $O(n^2 logn)$ complexity of the previous algorithms [21].

2.5.1 Definitions and Notations

- A,B,C, ... denotes nodes.

- a,b,c, ... denotes edges.

- p(e) denotes the affinity value of an edge e.

- "primitive cycle" denotes any cycle in the affinity graph.
- "cycle completing edge" denotes a "to be selected " edge that would complete a cycle.
- "cycle node" is that node of the cycle completing edge, which was selected earlier.
- "affinity cycle" denotes a primitive cycle that contains a cycle node. It is assumed that a cycle means an affinity cycle, unless otherwise stated.
- "former edge" denotes an edge that was selected prior to the cycle node.
- "cycle edge" is any of the edges forming a cycle.
- "extension of a cycle" refers to a cycle being extended by pivoting at the cycle node.

2.5.2 Fundamental Concepts

Based on the above definitions the mechanism of forming cycles could be explained. For example, in figure 2.5, suppose edges 'a' and 'b' were selected already and 'c' was selected next. At this time, since 'c' forms a primitive cycle, we have to check if it is an affinity cycle. This can be done by checking the possibility of a cycle. *Possibility of a cycle* results from the condition that *no former edge* exists, or *p(former edge)* $\leq$ *p(all the cycle edges)*. The primitive cycle a,b,c is an affinity cycle because it has no former edge and satisfies the possibility of a cycle. Therefore the primitive cycle a,b,c is marked as a candidate partition and node A becomes a cycle node.

Now let us explain how the extension of a cycle is performed. In figure 2.5, after the cycle node is determined, suppose edge 'd' was selected. At this time, 'd' is

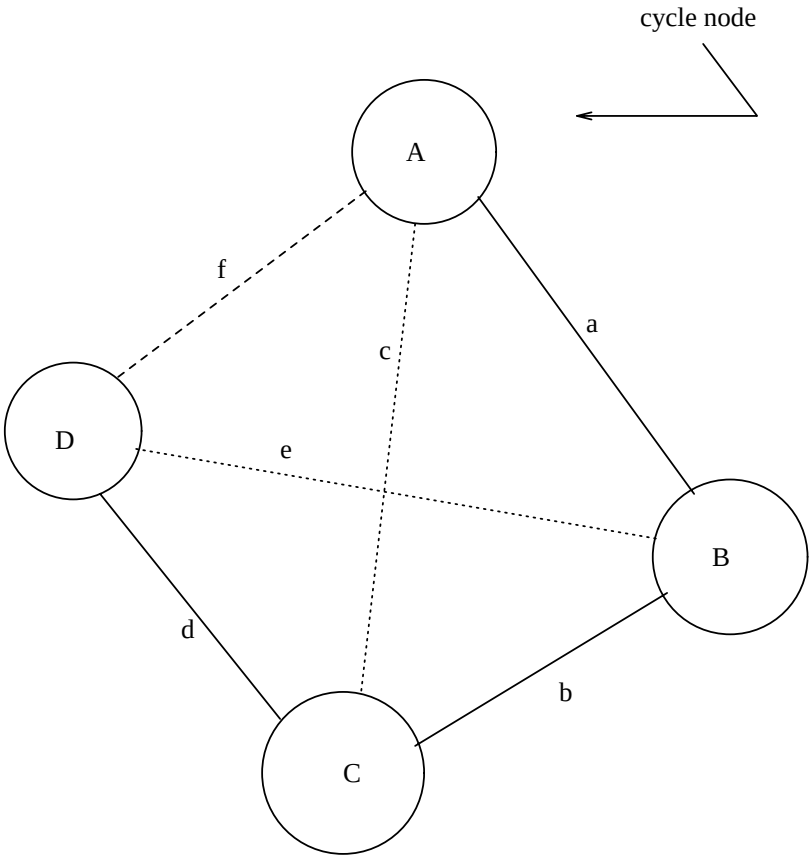

Figure 2.5. Cycle and Extension

checked as a potential edge for extension. It can be done by checking the possibility of extension of the cycle by 'd'. *Possibility of extension* results from the condition of *p(edge being considered)* or *cycle completing edge* $\geq$ *p(any one of the cycle edges)*. Thus the old cycle a,b,c is extended to the new cycle a,b,d,f if the edge 'd' under consideration, or the cycle completing edge f, satisfies the possibility of extension which is: p(d) or p(f) $\geq$ minimum of (p(a),p(b),p(c)). Now the process is continued: suppose 'e' was selected as the next edge. But we know from the definition of the extension of a cycle that 'e' cannot be considered as a potential extension because the primitive cycle d,b,e does not include the cycle node A. Hence it is discarded and the process is continued. There are two cases in partitioning. (1) Creating a partition with a new edge.

In the event that the edge selected next for inclusion (eg. d in figure 2.5) was not considered before, we call it a *new edge*. If a new edge by itself does not satisfy the possibility of extension, then we continue to check an additional new edge called cycle completing edge (eg. f in figure 2.5) for the possibility of extension. In figure 2.5, new edges d and f would potentially provide such a possibility of extension of the earlier cycle formed by edges a,b,c. If d,f meet the condition for possibility of extension stated above (namely p(d) or p(f) $\geq$ minimum of (p(a),p(b),p(c))), then the extended new cycle would contain edges a,b,d,f. If the condition were not met, we produce a cut on edge d (called the *cut edge*) isolating the cycle a,b,c. This cycle can now be considered a partition.

(2) Creating a partition with a former edge.

After cutting in (1), if there is a former edge, then change the previous cycle node to that node where the cut edge was incident, and check for the possibility of extension of the cycle by the former edge. For example, in figure 2.6, suppose that a,b, and c form a cycle with A as the cycle node, and that there is a cut on d, and that the former edge w exists. Then the cycle node A is changed to C because the cut edge d originates in C. We are now evaluating the possibility of extending the cycle a,b,c into one that would contain the former edge w. Hence we consider the possibility of the cycle a,b,e,w. Assume that w or e does not satisfy the possibility of extension, i.e., if "p(w) or p(e) $\geq$ minimum of (p(a),p(b),p(c))" is not true. Then the result is the following:

(i) w will be declared as a cut edge.

(ii) C remains as the cycle node, and

(iii)a,b,c becomes a partition.

Alternately, if the possibility of extension is satisfied, the result is:

(i) cycle a,b,c is extended to cycle w,a,b,e,

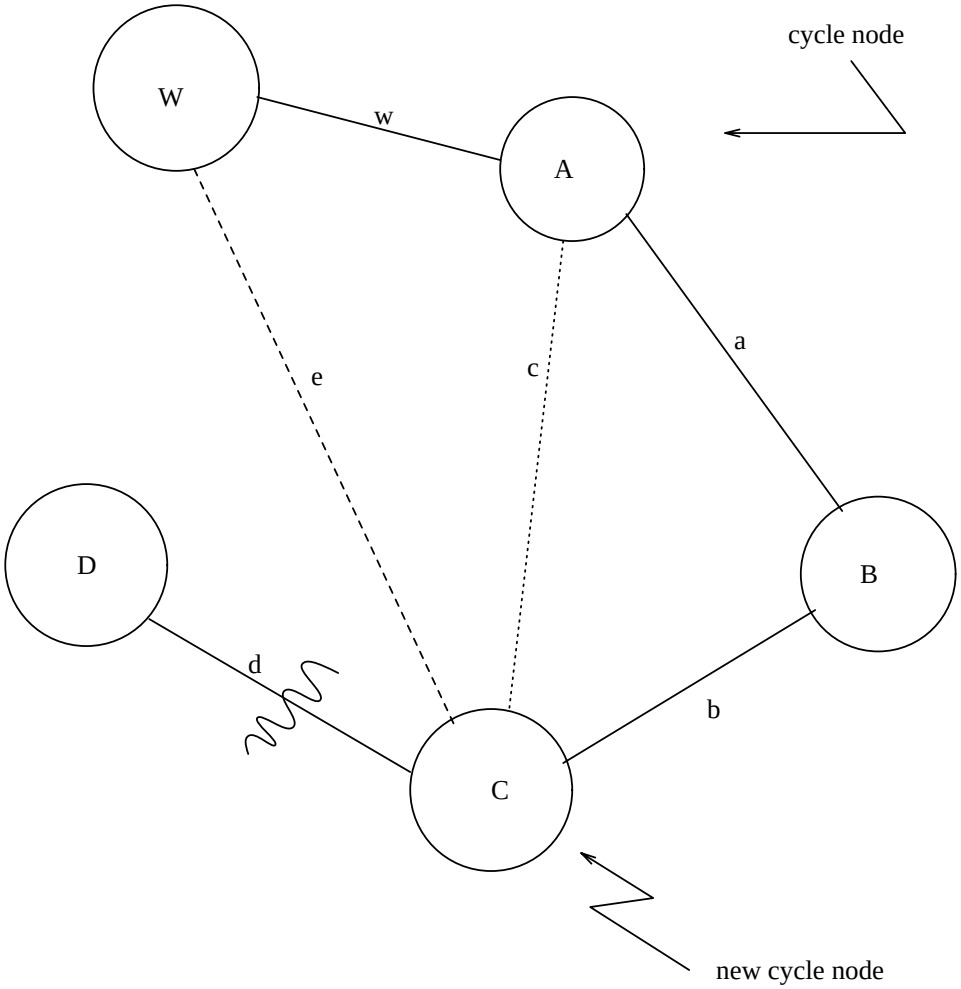

Figure 2.6. Partition

(ii) C remains as the cycle node, and

(iii) no partition can yet be formed.

The definitions above are used in this algorithm to process the affinity graph and to generate possible cycles from the graph. Note that each cycle gives rise to a vertical fragment. [22]

2.5.3 Description of the Algorithm

- 1. Construct the affinity graph of the attributes of the object being considered. Note that the AAM is itself an adequate data

structure to represent this graph. No additional physical storage of data would be necessary.

- 2. start from any node.

- 3. Select an edge which satisfies the following conditions:

  a) It should be linearly connected to the tree already constructed.

  b) It should have the largest value among the possible choices of edges at each end of the tree. Note that if there are several largest values, anyone can be selected.

  This iteration will end when all nodes are used for tree construction.

- 4. When the next selected edge forms a primitive cycle,

  a) If a cycle node does not exist, check for the "possibility of a cycle" and if the possibility exists, mark the cycle as an affinity cycle. Consider this cycle as a candidate partition. "Possibility of a cycle results from the condition that no former edge exists, or p(former edge) $\geq$ p(all the cycle edges).

  Go to step 3.

- 5. When the next selected edge does not form a cycle and a candidate partition exists.

  a) If no former edge exists, check for the possibility of extension of the cycle by this new edge. "possibility of extension" results from the condition of p(being considered edge or cycle completing edge) $\geq$ p(any of

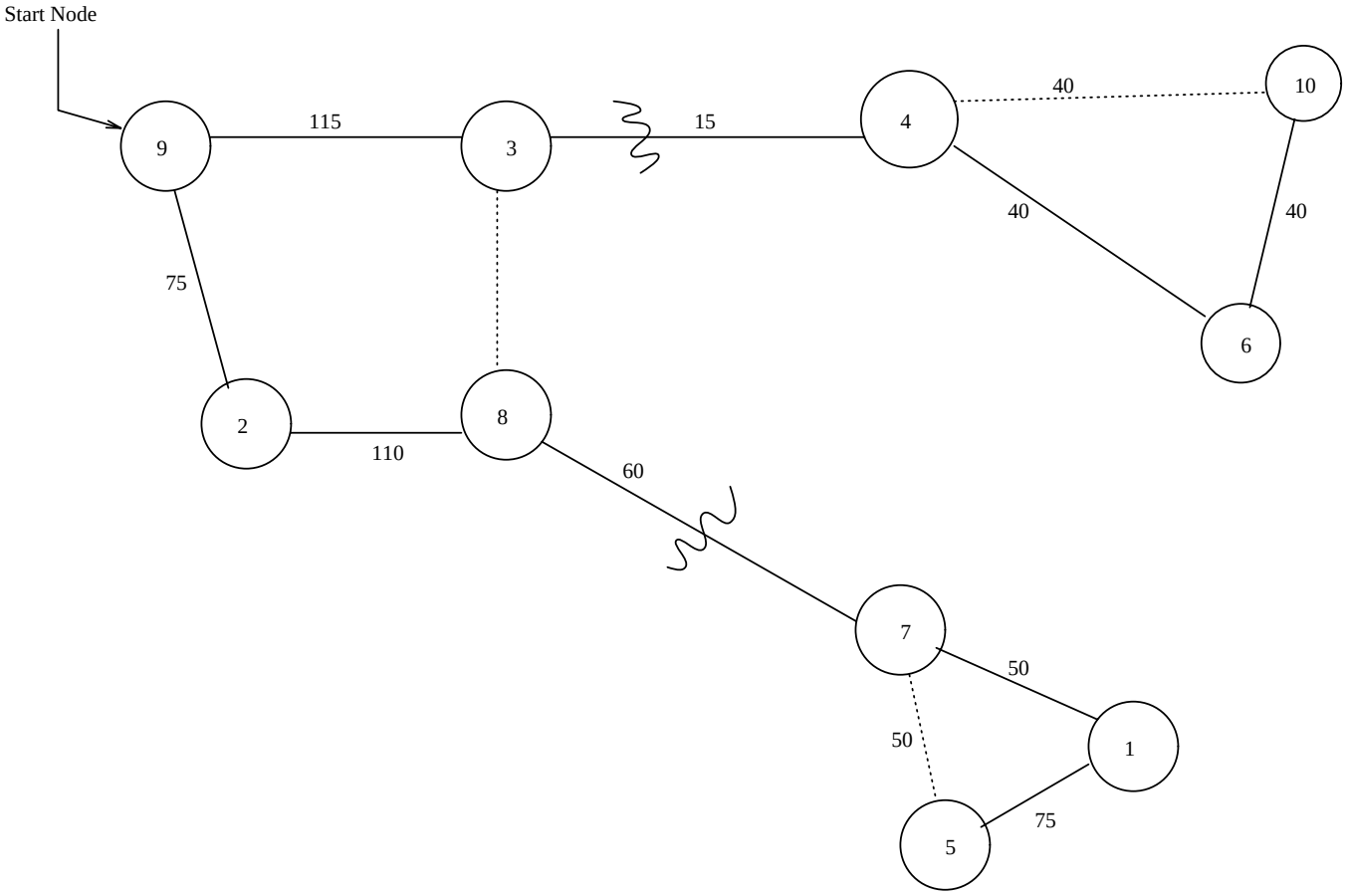

Figure 2.7. Partitioned Affinity Graph

the cycle edges). If there is no possibility, cut this
edge and consider the cycle as a partition. Go to step 3. b) If a former edge exists, change the cycle node and check
for the possibility of extension of the cycle by the
former edge. If there is no possibility, cut the former
edge and consider the cycle as a partition. Go to step 3.

The result of applying this algorithm to figure 2.4 is given in figure 2.7.

CHAPTER 3
ADAPTATION OF DATA CLUSTERING ALGORITHMS FOR DATA FRAGMENTATION

3.1 Data Clustering Problem

The problem clustering can be formally stated as follows: Given n attributes in a d-dimensional metric space, determine a partition of the attributes into M groups, or clusters, such that the attributes in a cluster are more similar to each other that to attributes in different cluster. The value of M may or may not be specified. A clustering criterion, such as square-error, must be adopted. A global criterion represents each cluster by a prototype and assigns attributes to clusters according to most similar prototypes. A local criterion forms clusters by utilizing local structure in the data. A simple theoretical solution to this partitional problem is to select a criterion, evaluate it for all possible partitions containing M clusters, and pick the partition that optimizes the criterion. The first difficulty encountered is selecting a criterion that translates one's intuitive notions about "cluster" into a reasonable mathematical formula. The second difficulty with this approach is that the number of partitions is astronomical, even for moderate numbers of attributes, so evaluating even the simplest criterion over all partitions is impractical. The total number of partitions for a given number of attributes is given by the Bell Number, which is defined as follows.

$$B_{n+1} = \sum_{i=0}^{n} \binom{n}{i} B_i \tag{3.1}$$

For example $B_{30} = 10^23$. To avoid this combinatorial explosion, a small set of "reasonable" partitions. This implies that heuristics must be used. In the heuristic

approach, an optimal or near optimal partition is found for the attributes by a process of stepwise minimization. An attribute partitioning heuristic which is based upon stepwise minimization starts with a given partition (eg. the trivial partition), and attempts to derive from it a new partition that is incrementally superior to the original one, in that the database partitioned according to the new partition will have a lower performance cost. When this is achieved, the heuristic further tries to improve upon the newly derived partition. Each time an improved partition is derived, the performance cost of the database is reduced. The stepwise minimization process is continued until no improvement can be made to the latest partition. This last partition will then be returned as the result of the attribute partitioning heuristic [10].

3.2 Previous Related Work

In the previous chapter, we have discussed only those algorithms that were developed specifically for Data fragmentation problem. However, a number of data clustering algorithms have been developed in other application areas such as statistics and pattern classification and analysis which could be adapted, with some changes, to the data fragmentation problem. Criteria used in clustering algorithms impose a certain structure on the data, and if the data happen to conform to the requirements of a particular criterion, the true clusters are recovered.

The most commonly used partitioning clustering strategy is based on the square-error criterion [13]. The general objective is to obtain that partition which, for a fixed number of clusters, minimizes the square-error. Minimizing square-error, or within-cluster variation, has been shown to be equivalent to maximizing the between-cluster variation. Clusters can also be viewed as regions of the attribute pattern space in which the patterns are dense, separated by regions of low attribute pattern density. In the mode-seeking partitioning algorithm due to Torn [30], clusters are identified

by searching for regions of high density, called modes, in the pattern space. Each mode is associated with a cluster center and each pattern is assigned to the cluster with the closest center.

Zahn [32] has demonstrated how the minimum spanning tree (MST) can be used to detect clusters. His choice of MST was influenced by the Gestalt principle, which favors the grouping of attribute patterns based on Euclidean distance measure. Shaffer et al [27] demonstrate the similarity of the mode-seeking partitioning algorithm [16] to the graph algorithm of Zahn [32] based on minimum spanning trees. Lu and Fu [17] used another graph-based approach called Nearest-Neighbor clustering algorithm to cluster patterns used in character recognition [13].

3.3 Square-Error Clustering Algorithm

The most commonly used partitional clustering algorithm is based on the square-error criterion. The general objective is to obtain that partition which, for a fixed number of clusters, minimizes the square-error.

Let us assume that n attributes have been partitioned into M fragments $(P_1, P_2, ...P_M)$ with n_i attributes in each fragment. Thus $\sum_{i=1}^{M} n_i = n$. The mean vector V_i for fragment i is defined as follows.

$$V_i = \frac{1}{n_i} \sum_{j=1}^{n_i} A_{ij} \qquad\qquad 0 < i \leq M \tag{3.2}$$

This mean vector represents an average access pattern of the transactions over all attributes of fragment i. For an attribute vector A_{ij}, $(A_{ij} - V_i)$ is called the "difference vector" for attribute j in fragment i. The square-error for the fragment P_i is the sum of the squares of the lengths of the difference vectors of all the attributes in fragment

i. It is given by

$$e_i^2 = \sum_{j=1}^{n_i} (A_{ij} - V_i)^T (A_{ij} - V_i) \qquad 0 < i \leq M \tag{3.3}$$

If $A_{ij} = V_i$ then e_i^2 will be zero. This will occur for the trivial case when there is a single attribute in each fragment or for the case when all the attribute in each fragment are relevant to all the transactions that access that fragment. It is the latter case that we are interested in and to avoid the former case, we will use the second component.

The square-error for the entire partition scheme containing M fragments is given by

$$E_M^2 = \sum_{i=1}^{M} e_i^2 \tag{3.4}$$

The objective of a square-error clustering method is to find a partition containing M fragments that minimizes E_M^2 for a fixed M. The square-error criterion views the centroids of clusters as prototypes. The error represents deviations of the patterns from the centroids. A general algorithm for iterative partitional clustering method is given below.

Step 1. Select an initial partition with M clusters.

Repeat steps 2 through 5 until the cluster membership stabilizes.

Step 2. Generate a new partition by assigning each pattern to its closest cluster center.

Step 3. Compute new cluster centers as the centroids of the clusters.

Step 4. Repeat steps 2 and 3 until an optimum value of the criterion

function is found.

Step 5. Adjust the number of clusters by merging and splitting existing clusters or by removing small, or outlier, clusters.

The details of the above algorithm can be found in [13].

3.4 Clustering by Graph Theory

Various kinds of geometric structures or graphs for analyzing multidimensional patterns have led to some useful algorithms which can identify clusters. A graph is constructed whose nodes represent the patterns to be clustered and whose edges represent relations between the nodes. In the simplest case, every node is connected to the remaining $(n-1)$ nodes, resulting in the complete graph. The edge weights are distances between pairs of patterns. Several graph structures, such as minimum spanning trees, relative neighborhood graphs, have been imposed on the set of patterns to capture perceptual grouping. These graphs choose a subset of the $n(n-1)/2$ edges in the complete graph to reflect the "structure" or the inherent separation among clusters. The edges in these graphs mostly correspond to small interpoint distances. These graphs depend only on the ordering of the lengths of edges. Clustering methods decompose the graphs into connected components by identifying and deleting "inconsistent" edges. Each component represents a cluster.

3.4.1 Zahn's Clustering Algorithm

Step 1. Construct the minimum spanning tree for the set of n attributes given.

Step 2. Identify inconsistent edges in the MST.

Step 3. Remove the inconsistent edges to form connected components and call them clusters.

Zahn considers several criteria for inconsistency. In one, an edge is inconsistent if its weight is significantly larger than the average of nearby edge weights. Thus

the inconsistent edges are related to cluster separation. The number of standard deviations by which an edge weight differs from the average of nearby edge weights and the ratio of the edge weight to the average of nearby edge weights are two means for identifying inconsistent edges [13].

3.5 Adaptation of Data Clustering Algorithms for Data Fragmentation

There are important differences in the criteria that is used in traditional clustering problems and data fragmentation problem. In data clustering algorithms, the number of clusters is usually fixed. Otherwise, the extreme case of only a single cluster in the partition will minimize the inter-cluster variation. However in the database design application, there is a need to determine the *number* of clusters as well and hence the objective function used in data clustering algorithms cannot be borrowed without any changes to vertical partitioning in databases. Algorithms such as Bond Energy Algorithm, Binary Vertical Partitioning, Ra's algorithm and Zahn's algorithm etc. use affinity matrix as the input. The attribute affinity is a measure of an imaginary bond between *a pair* of attributes. Because only a pair of attributes is involved, this measure will not be able to reflect the closeness or affinity when *more than two* attributes are in a partition. Hence the algorithms which use attribute affinity matrix are using a measure that has no bearing on the affinity as measured with respect to the entire cluster. As a consequence, we believe, it was difficult to show or even characterize affinity values for clusters having more than two attributes.

As we wanted to obtain a general objective function and a criterion for describing affinity value for clusters of different sizes, our approach does not assume an attribute affinity matrix. The input model that we consider is a matrix which consists of attributes (columns) and the transactions (rows) with the frequency of access to the attributes for each transaction, as the values in the matrix. With this input model we

overcome the disadvantage that is inherent to approaches based on attribute affinity matrix.

As we can see from the above discussion, there are a number of partitioning algorithms available both in the database design area and in other application areas. Many of these algorithms use different objective criteria to arrive at a partitioning scheme. The objective function used by one algorithm is not suitable for evaluating the "goodness" of other algorithms. Thus we do not have a common objective function to compare and evaluate the results of these partitioning algorithms, or in general evaluate the "goodness" of a particular partitioning scheme. Hence we need a partition Evaluator to compare and evaluate different algorithms, that use the same input in the database design process. Since attribute usage matrix is the most commonly used input available during the design stage, we first design an Evaluator which can be used to evaluate the "goodness" of partition arrived at using this input. This Partition Evaluator can be used as a basis for developing algorithms to create fragments of a relation. With this approach, there is hope that admissibility aspects of algorithms can be shown. In addition, this Partition Evaluator has the flexibility to incorporate any other information such as type of queries (retrieval/updates), allocation information about the partitions, remote processing cost(transmission cost) and the transaction usage pattern at any particular site. In the next chapter we will discuss the development of the Partition Evaluator in detail.

CHAPTER 4
DEVELOPMENT OF THE PARTITION EVALUATOR

4.1 Need for an objective function

Algorithms such as Bond Energy, Binary Vertical Partitioning, Ra's algorithm and Zahn's algorithm etc. use affinity matrix as the input. The attribute affinity is a measure of an imaginary bond between *a pair* of attributes. Because only a pair of attributes is involved, this measure does not reflect the closeness or affinity when *more than two* attributes are involved. Hence the algorithms which use attribute affinity matrix are using a measure (that is an ad hoc extrapolation of pairwise affinity to cluster affinity) that has no bearing on the affinity as measured with respect to the entire cluster. As a consequence, we believe, it was difficult to show or even characterize affinity values for the resulting clusters having more than two attributes.

As we wanted to obtain a general objective function and a criterion for describing affinity value for clusters of different sizes, our approach does not assume an attribute affinity matrix. The input model that we consider is a matrix which consists of attributes (columns) and the transactions (rows) with the frequency of access to the attributes for each transaction, as the values in the matrix. With this input model we overcome the limitations that are inherent to approaches based on attribute affinity matrix.

As is evident from the discussion in the previous section, there are a number of partitioning algorithms available both in the database design area and in other application areas. Many of these algorithms use different criteria to arrive at a partitioning scheme. The objective function used by one algorithm is not suitable for

evaluating the "goodness" of other algorithms. Thus we do not have a common objective function to compare and evaluate the results of these partitioning algorithms, or in general evaluate the "goodness" of a particular partitioning scheme. Hence we need a partition Evaluator to compare and evaluate different algorithms, that use the same input in the database design process. Since attribute usage matrix is the most commonly used input available during the initial design stage, we first design an Evaluator which can be used to evaluate the "goodness" of partitions arrived at using this input. This Partition Evaluator can be used as a basis for developing algorithms to create fragments of a relation. With this approach, there is hope that admissibility aspects of algorithms can be shown. In addition, this Partition Evaluator has the flexibility to incorporate other information, such as type of queries (retrieval/updates), allocation information about the partitions, remote processing cost (transmission cost) and the transaction usage pattern at any particular site.

In any practical database application, a transaction does not usually require all the attributes of the tuples of a relation being retrieved during the processing of the transaction. When a relation is *vertically* divided into data fragments, the attributes stored in a data fragment that are irrelevant (i.e., not accessed by the transaction) with respect to a transaction, add to the retrieval and processing cost, especially when the number of tuples involved in the relation is very large. In a centralized database system with memory hierarchy, this will lead to too many accesses to the secondary storage. In a distributed database management system, when the relevant attributes (i.e., attributes accessed by a transaction) are in different data fragments and allocated to different sites, there is an additional cost due to remote access of data. Thus one of the desirable characteristics of a distributed database management systems that we wish to achieve through partitioning is the *local accessibility* at any

site. In other words, each site must be able to process the transactions *locally* with *minimal access* to data located at remote sites.

Ideally, we would like any transaction to access only the attributes in a single data fragment with no or minimal access of irrelevant attributes in that fragment. But this is impossible to achieve in the general cased since transactions access different and overlapping subsets of attributes of a relation. Moreover, transactions are run at different sites and hence some of the data fragments that contain relevant attributes of a transaction may reside in remote sites. The overall transaction processing cost in a distributed environment thus consists of local transaction processing cost and the remote transaction processing cost. Though it is possible to replicate the data to avoid remote processing cost, for the first step we assume no data redundancy to avoid modeling overhead to ensure data integrity and consistency and also additional storage costs. In this paper, we assume that during the database design process, "partitioning" phase is followed by the "allocation" phase during which the non-overlapping data fragments obtained during the partitioning phase are allocated to different sites possibly with some replication. Hence the partition evaluator we propose will evaluate vertical partitioning schemes wherein the data fragments are *non-overlapping* in the attributes. Here non-overlapping refers only to non-primary key attributes. The primary key is preserved in each partition. This is necessary to obtain the original relation without any loss or addition of spurious tuples. Discussion of our approach to relations that are not in Boyce-Codd normal form is in a later section.

The goal of partitioning the attributes and allocating to different sites, is to arrive at a minimum processing cost for any transaction originating at any site. Since the transaction processing cost has two components, one due to local processing and another due to remote processing, our Partition Evaluator that measures the

"goodness" of a vertical partitioning scheme also has two corresponding component terms; we refer to these terms as "irrelevant local attribute access cost" and "relevant remote attribute access cost" terms respectively. For simplicity, we assume that a single access of a data fragment corresponds to an unit cost; this assumption can easily be relaxed if more information is available regarding the access methods, network parameters etc. The irrelevant local attribute cost term measures the local processing cost of transactions that is due to irrelevant attributes of data fragments assuming that all the data fragments required by a transaction are available locally. The relevant remote attribute access term measures the remote processing cost due to relevant attributes of data fragments that are accessed remotely by transactions; note that the contribution to remote access cost due to irrelevant attributes is already included in the first term. Since we do not know during partitioning how the data fragments are allocated, we compute the second term assuming that data fragments needed by a transaction are located at different sites. In the absence of any information regarding the transaction execution strategies, we can compute the second term either by determining the average of all the remote access costs obtained by running the transaction at every site containing a fragment needed by the transaction or by assuming the transaction to be run at one of the fragment sites. If more information is available regarding the transaction strategies, we can incorporate it in the second term.

The *ideal* characteristic of a Partition Evaluator is summarized in Figure 4.1 where the behavior of the two components as a function of partition size (number of data fragments) is illustrated. The first component should be maximum for a partition size of 1 and should be zero for a partition size equal to number of attributes in the relation. The second component on the other hand should be zero and maximum respectively for these two extremes. In between the two extremes, the

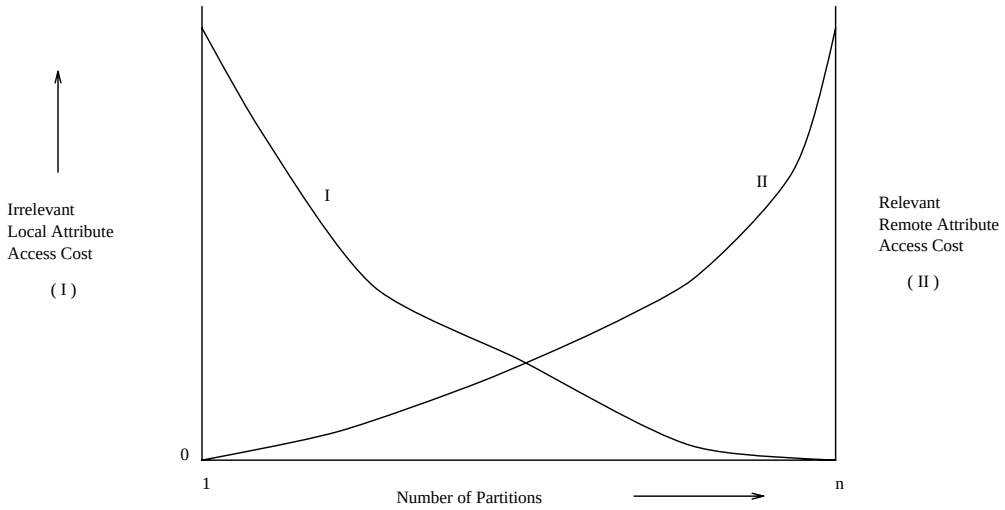

Figure 4.1. Ideal characteristics of PE components

first component should be more responsive to smaller partition sizes while the second component should be more responsive to larger partition sizes. This is necessary to avoid unnecessary bias toward one component or another since we know that in a distributed database design, it is undesirable to have the two extreme cases of the partition size. In addition to being responsive to partition sizes, the evaluator function should discriminate between different distributions of the attributes among the fragments for a fixed partition size. Later, we will present experimental evidence to demonstrate that our Partition Evaluator meets these criteria reasonably well.

4.2 Derivation of the Partition Evaluator

4.2.1 Database and Transaction Specification

We derive a partition evaluator in this chapter without making any assumptions on the input. The input assumed is a relation (consisting of a set of attributes) and an attribute usage matrix. We really do not have to make any assumptions as to whether the input relation is a Universal relation or whether it is in a particular normal form.

The partitioning obtained is non-overlapping except for the key attributes in the input relation. The key attributes are part of each partition. In a later chapter we indicate how normalization and functional dependency information can be easily added to our approach.

The input model that we use is an attribute usage matrix [AUM(t,j)] which consists of the attributes(j) in a relation as columns and the transactions(t) as rows with the frequency of access to the attributes for each transaction as the values in the matrix. Note that we are avoiding the construction of any attribute affinity matrix which has its own limitation pointed out earlier.

A representative attribute usage matrix[1] is shown below:

$Trans \backslash Attrs$	$A1$	$A2$	$A3$	$A4$	$A5$
$T1$	0	$q1$	0	$q1$	$q1$
$T2$	$q2$	$q2$	$q2$	0	$q2$
$T3$	$q3$	0	0	$q3$	$q3$
$T4$	0	$q4$	$q4$	0	0
$T5$	$q5$	$q5$	$q5$	0	0

4.2.2 Definitions and Notations

A *partition* (scheme) is a division of attributes of a relation into vertical fragments in which for any two fragments, the set of attributes of one is non-overlapping with the set of attributes of another. For example, the partition $\{(1,3)(2,4)(5)\}$ defines a collection of fragments in which attributes 1 and 3 are in one fragment, 2 and 4 are in another and 5 is in a separate fragment. The following are used in the derivation of the Partition Evaluator.

n : Total number of attributes in a relation that is being partitioned.

T : Total number of transactions that are under consideration.

q_t : frequency of transaction t for $t = 1, 2, \ldots, T$.

[1] In this chapter, no distinction is made between update and retrieval transactions; it is discussed as one of the extensions in a later chapter

M : Total number of fragments of a partition.

n_i : Number of attributes in fragment i.

n^r_{ikt} : Total number of attributes that are in fragment k
accessed remotely with respect to fragment i by transaction t.

f^i_{tj} : frequency of transaction t accessing attribute j in fragment i
note that f^i_{tj} is either 0 or q_t

A_{ij} : Attribute Vector for attribute j in fragment i.
t-th component of this vector is f^i_{tj}

S_{it} : Set of attributes contained in fragment i that the transaction t
accesses; It is empty if t does not need fragment i.

$|S_{it}|$: number of attributes in fragment i that the transaction t accesses.

R_{itk} : Set of relevant attributes in fragment k accessed remotely
with respect to fragment i by transaction t;
these are attributes not in fragment i but needed by t

$|R_{itk}|$: number of relevant attributes in fragment k accessed remotely
with respect to fragment i by transaction t

The attribute vector A_{11} (assuming A_1 is in partition 1) for the example given above is as follows.

$$A_{11} = \begin{bmatrix} 0 \\ q2 \\ q3 \\ 0 \\ q5 \end{bmatrix}$$

Consider the partitioning scheme (1,3),(2,4),(5), for the attribute usage matrix given above. Fragment 1 is (1,3), fragment 2 is (2,4) and fragment 3 is (5). Assume that transaction T2 (i.e., $t = 2$) is run at the site where fragment 1 is located. Then S_{it} is (1,3) and $|S_{it}|$ is 2. n^r_{i2t} is 2 and n^r_{i3t} is 1. $|R_{it2}|$ is 1 and $|R_{it3}|$ is 1.

Next we explain how each of the two components of our Partition Evaluator is derived.

4.2.3 Irrelevant local attribute access cost

For the first component, we use the square-error criterion as given in [Jain 88] for data clustering. The objective here is to obtain a partition which will minimize the square-error for a fixed number of fragments. This criterion assigns a penalty factor whenever irrelevant attributes are accessed in a particular fragment.

Let us assume that n attributes have been partitioned into M fragments $(P_1, P_2, ...P_M)$ with n_i attributes in each fragment. Thus $\sum_{i=1}^{M} n_i = n$. The mean vector V_i for fragment i is defined as follows.

$$V_i = \frac{1}{n_i} \sum_{j=1}^{n_i} A_{ij} \qquad\qquad 0 < i \leq M \qquad\qquad (4.1)$$

This mean vector represents an average access pattern of the transactions over all attributes of fragment i. For an attribute vector A_{ij}, $(A_{ij} - V_i)$ is called the "difference vector" for attribute j in fragment i. The square-error for the fragment P_i is the sum of the squares of the lengths of the difference vectors of all the attributes in fragment i. It is given by

$$e_i^2 = \sum_{j=1}^{n_i} (A_{ij} - V_i)^T (A_{ij} - V_i) \qquad\qquad 0 < i \leq M \qquad\qquad (4.2)$$

If $A_{ij} = V_i$ then e_i^2 will be zero. This will occur for the trivial case when there is a single attribute in each fragment or for the case when all the attribute in each

fragment are relevant to all the transactions that access that fragment. It is the latter case that we are interested in and to avoid the former case, we will use the second component.

The square-error for the entire partition scheme containing M fragments is given by

$$E_M^2 = \sum_{i=1}^{M} e_i^2 \tag{4.3}$$

Smaller the value of E_M^2, smaller is the cost due to access of irrelevant attributes. E_M^2 however does not reflect the cost that might be incurred by accessing attributes remotely when the fragments may be in different sites. Hence, for distributed database applications, we cannot evaluate partitions on the basis of E_M^2 alone. The behavior of E_M^2 is given in figure 4.2 as curve I for a data set consisting of 10 attributes and eight transactions (more details of this example are discussed in the next chapter).

From this graph, we can see that the minimum value for E_M^2 is certainly achieved for n partitions in an n attribute system (The minimum value may be reached even for less than n partitions depending on the AUM). We would like to have a number of fragments which is typically much less than n and still having the least E_M^2 value. In some data clustering techniques, the number of data clusters is minimized using an index called Davies-Bouldwin (DB) index [13] which is a measure of the spread between centers of the clusters. For a typical data set with small standard deviation (less than 0.1), this index reaches a global minimum (highest spread) for a partition size that falls between the extremes. From the curve I of figure 4.2, we can see that the standard deviation of the data set given by the attribute usage matrix is greater than 0.1, which does not meet the condition for using the DB index. For

this reason, we seek another quantity which will reflect the remote access cost. In the next chapter, we will discuss the development of a quantity for remote attribute access cost.

4.2.4 Relevant Remote Attribute Access Cost

Now we will include the second component which would compute a penalty factor that computes the function shown earlier in figure 4.1. Given a set of partitions, for each transaction running on a partition compute the ratio of the number of remote attributes to be accessed to the total number of attributes in each of the remote partitions. This is summed over all the partitions and over all transactions giving the following equation. The second term is given by

$$E_R^2 = \sum_{t=1}^{T} \Delta_{i=1}^{M} \sum_{k \neq i} \left[q_t^2 * |R_{itk}| \frac{|R_{itk}|}{n_{itk}^r} \right] \tag{4.4}$$

Here Δ[2] is an operator that is either an average, minimum or maximum over all i. These different choices of the operator give rise to average, optimistic and pessimistic estimates of the remote access cost. If specific information is available regarding transaction execution strategies, then we can determine for each transaction t, the remote fragments accessed by the transaction and the remote access cost can be refined accordingly. In our experimental investigation, we use the optimistic estimate for illustration.

We will show in the next chapter how we obtained the above form for the second term. Our Partition Evaluator (PE) function is given by

$$PE = E_M^2 + E_R^2 \tag{4.5}$$

[2] Δ is introduced to keep the formula general and applicable to a wide class of problems; also, a desired new operator can be substituted for Δ without having to change the objective function.

4.2.5 Compatibility of the Two Components in PE

As the second component has frequency in quadratic form, we need to make sure that both the components are compatible in terms of the units they produce. Specifically, we need to see whether the frequency appears in the same way in the first component. In order to do that, it is instructive to look closely at the first term. In particular, we will rewrite E_M^2 differently so as to identify the contributions due to each transaction to the square error term of each fragment in the partition. Thus

$$E_M^2 = \sum_{i=1}^{M} \sum_{j=1}^{n_i} (A_{ij} - V_i)^T (A_{ij} - V_i) \tag{4.6}$$

Now the mean Vector V_i for fragment i, can be defined as follows.

$$V_i = \begin{bmatrix} \frac{|S_{i1}| * q_1}{n_i} \\ \frac{|S_{i2}| * q_2}{n_i} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \frac{|S_{it}| * q_t}{n_i} \end{bmatrix}$$

The attribute vector A_{ij} is,

$$A_{ij} = \begin{bmatrix} f_{1j}^i \\ f_{2j}^i \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ f_{tj}^i \end{bmatrix}$$

Now,

$$E_M^2 = \sum_{i=1}^{M} \sum_{j=1}^{n_i} \left[f_{1j}^i - \frac{|S_{i1}| * q_1}{n_i}, \ldots, f_{tj}^i - \frac{|S_{it}| * q_t}{n_i} \right] \begin{bmatrix} f_{1j}^i - \frac{|S_{i1}| * q_1}{n_i} \\ f_{2j}^i - \frac{|S_{i2}| * q_2}{n_i} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ f_{tj}^i - \frac{|S_{it}| * q_t}{n_i} \end{bmatrix} \tag{4.7}$$

The above formula can be rewritten as follows so as to allow us to identify the different components that make up the irrelevant local attribute access term.

$$E_M^2 = \sum_{i=1}^{M} \sum_{j=1}^{n_i} \sum_{t=1}^{T} \left[\delta_{jt} * q_t^2 \left(1 - \frac{|S_{it}|}{n_i} \right)^2 + (1 - \delta_{jt}) \left(q_t * \frac{|S_{it}|}{n_i} \right)^2 \right] \tag{4.8}$$

where,

δ_{jt} = 1 if the attribute j is accessed by the transaction t

= 0 if the attribute j is not accessed by the transaction t.

The first term $q_t^2 \left(1 - \frac{|S_{it}|}{n_i}\right)^2$ represents relevant attribute accesses and the second term represents irrelevant attribute accesses. Even if we have a 0 in A_{ij}, we still have the mean squared quantity $\left(q_t * \frac{|S_{it}|}{n_i}\right)^2$.

Therefore,

$$E_M^2 = \sum_{i=1}^{M} \sum_{t=1}^{T} \left[|S_{it}| * q_t^2 \left(1 - \frac{|S_{it}|}{n_i} \right)^2 + (n_i - |S_{it}|) \left(q_t * \frac{|S_{it}|}{n_i} \right)^2 \right] \tag{4.9}$$

where

$\sum_{j=1}^{n_i} \delta_{jt}$ = $|S_{it}|$ and

$\sum_{j=1}^{n_i} (1 - \delta_{jt}) = n_i - |S_{it}|$

Hence,

$$E_M^2 = \sum_{i=1}^{M} \sum_{t=1}^{T} \left[q_t^2 \left| S_{it} \right| \left(1 - \frac{\left| S_{it} \right|}{n_i} \right)^2 + q_t^2 (n_i - \left| S_{it} \right|) \left(\frac{\left| S_{it} \right|}{n_i} \right)^2 \right] \tag{4.10}$$

If $n_i = |S_{it}|$, then $E_M^2 = 0$. This implies that the transaction t accesses all attributes in fragment i whenever it accesses the fragment i. We can still reduce the above equation as follows.

$$E_M^2 = \sum_{i=1}^{M} \sum_{t=1}^{T} \left[q_t^2 * \left| S_{it} \right| \left(1 + \frac{\left| S_{it} \right|^2}{n_i^2} - 2 * \frac{\left| S_{it} \right|}{n_i} \right) + q_t^2 * \left| S_{it} \right| (n_i - \left| S_{it} \right|) \left(\frac{\left| S_{it} \right|}{n_i^2} \right) \right] \tag{4.11}$$

Simplifying the equation above we get,

$$E_M^2 = \sum_{i=1}^{M} \sum_{t=1}^{T} \left[q_t^2 * \left| S_{it} \right| \left(1 - \frac{\left| S_{it} \right|}{n_i} \right) \right] \tag{4.12}$$

The equation above is the same as equation as 6, but in a much simpler form. We can clearly see from this equation the contribution to E_M^2 by the irrelevant attributes; $\frac{(1-|S_{it}|)}{n_i}$ is the fraction of irrelevant attributes in fragment i as far as transaction t is concerned. As it was mentioned earlier, E_M^2 is the cost factor only for local transaction processing.

Now we can see that the E_R^2 term shown earlier, is very much similar in form to the E_M^2 term. Also the necessity for having a squared frequency term in E_R^2 is made clear. Hence the PE is given by

$$PE = E_M^2 + E_R^2 \tag{4.13}$$

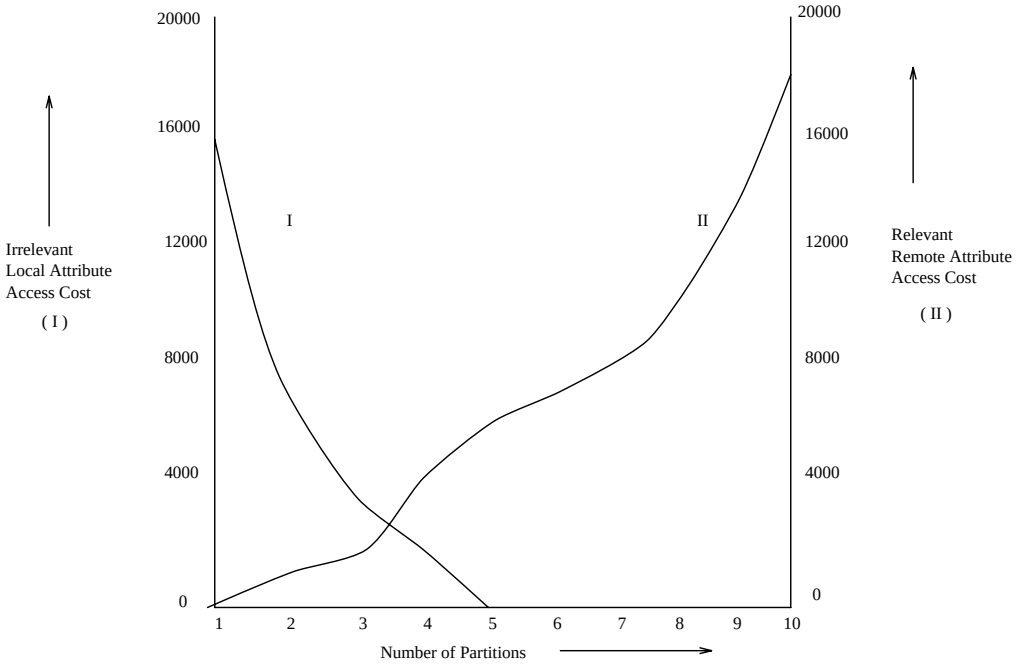

Figure 4.2. Behavior of PE components for an example

Curve II of Figure 4.2 illustrates the behavior of E_R^2 for the same example of 10 attributes and 8 transactions.

CHAPTER 5
ANALYSIS OF THE PARTITION EVALUATOR

The final form of the Partition Evaluator is given in equation 12. In order to analyze and test the behavior of the PE, an exhaustive enumeration program was written. The input to the program is an attribute usage matrix. The following input was used to test the Partition Evaluator. For each fragment (from 1 to 10), the Partition Evaluator was computed.

Input: **Attribute Usage matrix**

$Trans.\backslash Attrs.$	1	2	3	4	5	6	7	8	9	10
$T1$	25	0	0	0	25	0	25	0	0	0
$T2$	0	50	50	0	0	0	0	50	50	0
$T3$	0	0	0	25	0	25	0	0	0	25
$T4$	0	35	0	0	0	0	35	35	0	0
$T5$	25	25	25	0	25	0	25	25	25	0
$T6$	25	0	0	0	25	0	0	0	0	0
$T7$	0	0	25	0	0	0	0	0	25	0
$T8$	0	0	15	15	0	15	0	0	15	15

An exhaustive enumeration program was written in C++ to produce all the possible combinations of attributes with number of fragments varying from 1 to 10. The Partition Evaluator was applied to each of these combinations. We assume that if a transaction is to be run on a fragment, and that fragment does not contain even a single attribute accessed by that transaction, then that transaction is not run on that fragment. Each of the transactions is run on each fragment and the minimum, maximum and the average value of the Partition Evaluator is calculated. It took about forty minutes to run this Partition Evaluator program on a Sun 4 machine. Total number of fragments evaluated was 115975. The optimal values (minimum)

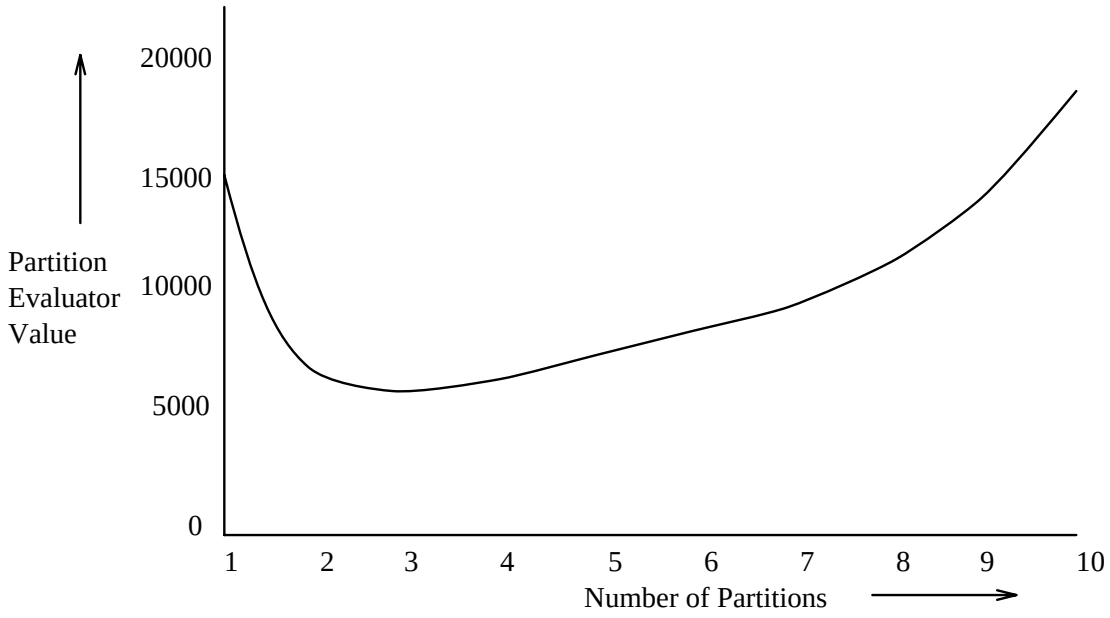

Figure 5.1. Behavior of partition evaluator for an example

along with the partitioning scheme for each partition size is given below. The results are plotted in figure 5.1.

Number Of Fragments	*Min PE Value*	*Partition Scheme*
1	15085	(12345678910)
2	8457	(1456710)(2389)
3	5820	(157)(2389)(4610)
4	6024	(15)(2389)(4610)(7)
5	6874	(15)(2389)(46)(7)(10)
6	7724	(15)(2389)(4)(6)(7)(10)
7	8976	(1)(2389)(4)(5)(6)(7)(10)
8	11692	(1)(289)(3)(4)(5)(6)(7)(10)
9	14000	(1)(28)(3)(4)(5)(6)(7)(9)(10)
10	18350	(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)

As we can see from the above table the optimum value (third row above) is found to be in a partition of three fragments. The Partition Evaluator values above are the minimum of the values where the second component of the PE is calculated using optimistic (minimum value) estimate. For this particular example, other algorithms such as Ra's [Nava 89], Zahn's [Zahn 71] and Binary Vertical Partitioning [Nava 84] identify the above mentioned partition set (ie 3 fragments) as their optimum. In Zahn's approach, once the maximum spanning tree is obtained two different conditions can be used to determine the partitions [Jain 88]. For this example, when these two conditions are applied to Zahn's algorithm, they produce two different partitioning schemes. One of them is the same as the optimal partitioning scheme obtained as above and the other one is not. Hence, one may want to use another criterion (such as the PE) to guide the selection of the most appropriate condition when faced with a number of alternatives. Since producing any partitioning scheme using the Bond

Energy algorithm is subjective in nature, this Partition Evaluator can be used to guide the partitioning once the Bond Energy Algorithm is applied. Ra's and Binary Vertical Partitioning algorithms were applied to another example with twenty attributes and fifteen transactions [Nava 84]. The results of these two algorithms were different. Ra's algorithm produces five fragments and Binary Vertical partitioning algorithm produced four fragments. The Partition Evaluator was applied to both the results and it was found that indeed the four fragment result is better than the five fragment result. This example highlights the usefulness of the Partition Evaluator to evaluate the results of the different partitioning algorithms.

The following discussion explains how one of these figures is arrived at.

Number Of Fragments	*Min PE Value*	*Partition Scheme*
3	5820	(157)(2389)(4610)

Let us call (1 5 7) as fragment I, (2 3 8 9) as fragment II and (4 6 10) as fragment III. The first step is to calculate the square error for the given input.

Mean for each fragment is:

Fragment I	*Fragment II*	*Fragment III*
25	0	0
0	50	0
0	0	25
12	18	0
25	25	0
17	0	0
0	13	0
0	8	15

1) The square error

$$E_M^2 = \sum_{i=1}^{M} \sum_{j=1}^{n_i} (X_{ij} - V_i)^T (X_{ij} - V_i)$$

$E_M^2 = 1234 \quad + \quad 2078 \quad + \quad 0 = 3312$

2) Cost due to relevant remote attribute accesses is:

	Value	Minimum of the values
T1 run @ Fragment I:	0	0
T1 run @ Fragment II:	cannot be run	
T1 run @ Fragment III:	cannot be run	
T2 run @ Fragment I:	cannot be run	
T2 run @ Fragment II:	0	0
T2 run @ Fragment III:	cannot be run	
T3 run @ Fragment I:	cannot be run	
T3 run @ Fragment II:	cannot be run	
T3 run @ Fragment III:	0	0
T4 run @ Fragment I:	$35^2 * 2 * 2/4 = 1225$	
T4 run @ Fragment II:	$35^2 * 1 * 1/3 = 408$	408
T4 run @ Fragment III:	cannot be run	
T5 run @ Fragment I:	$25^2 * 4 * 4/4 = 2500$	
T5 run @ Fragment II:	$25^2 * 3 * 3/3 = 1875$	1875
T5 run @ Fragment III:	cannot be run	
T6 run @ Fragment I:	0	0
T6 run @ Fragment II:	cannot be run	
T6 run @ Fragment III:	cannot be run	
T7 run @ Fragment I:	cannot be run	

T7 run @ Fragment II:	0	0
T7 run @ Fragment III:	cannot be run	
T8 run @ Fragment I:	cannot be run	
T8 run @ Fragment II:	$15^2 * 3 * 3/3 = 675$	
T8 run @ Fragment III:	$15^2 * 2 * 2/4 = 225$	225

Total cost of accessing relevant remote attributes is 408 + 1875 + 225 = 2508

Therefore,

$$\text{PE} = 3312 + 2508 = 5820.$$

The above example shows that the Partition Evaluator can be used to evaluate the results of any partitioning scheme.

CHAPTER 6
EXTENSIONS AND IMPLEMENTATION

So far, we have discussed the formulation of an objective function (Partition Evaluator) assuming that attribute usage matrix is the only information available to the designer during the partitioning phase of the design process. In this section we first discuss briefly how we can use the objective function in the presence of additional information and then discuss the status of the implementation effort.

Dependencies: If a universal relation or a relation which is not in the Boyce-Codd normal form is used as input, several constraints need to be considered when a partition is generated. The cost of maintaining functional and multi-valued dependencies is reduced if all the attributes participating in a dependency are either in the same fragment or at least allocated to the same node. The first case can be easily accommodated without changing the objective function by defining a *dependency checker* which evaluates whether a partition should be considered for cost computation. In other words, during the design process any fragment that violates any of the dependencies would be rejected even though that fragment might give the minimum value for the partition evaluator; such partitioning schemes are deemed not admissible. For the second case, the information is passed to the allocation phase.

Fixed partition size: In many cases, the number of sites are known a priori and the objective function need to minimize the cost for a given number of fragments. This, again, does not require any changes as one can choose the minimum cost of all the alternatives for the number of fragments specified.

Update vs. retrieval transactions: Although no distinction was made between update and retrieval transaction in the earlier discussion, it is quite straightforward to distinguish between them. Typically, retrieve only transaction access data items only once where as there is an additional overhead of writing back for updated data items on to secondary storage (sometimes more if the data item has already been flushed to secondary storage). A weight factor can be used to distinguish update type from retrieval type transactions. We can increase the frequency (for example double it) of update type transactions to reflect their processing cost. In our simulation, we can take into account the type of queries by increasing the frequency for update type queries in the attribute usage matrix.

Local vs. remote processing cost : In our formulation of PE, we have assumed a unit cost for processing local as well as remote data once the remote data reaches the local node. We also assumed a uniform transmission cost between any two sites in the network. However, realistically, there is difference between accessing local data possibly using access methods and remote data for which there are no access methods (unless dynamically created at the local site). Also, the transmission costs between any two nodes is not likely to be the same.

We can include another factor to reflect the ratio of local processing cost to the processing of data coming from remote sites. This factor can be included giving different weights to the two component terms in equation 13. If W_1 is the factor for processing local data and W_2 is the factor for processing remote data, it can be accommodated by modifying equation 13 as follows:

$$PE = E_M^2 + (W_2/W_1)E_R^2 \tag{6.1}$$

Note, however, that a better way would be to integrate the factors into the formula.

Non-uniform transmission cost between nodes can be taken into account by modifying equation 4 to include a multiplicative factor TR_{ik} (which reflects the actual transmission cost/unit data between sites i and k) inside the inner summation.

Replication: The Partition Evaluator can be extended with minimal changes to accommodate replication of fragments. In this case, we assume that attributes are remotely accessed only if they do not exist locally. Thus in computing the second term, only those attributes that exist in a fragment remotely accessed with respect to the site of another fragment (call it "local") but do not exist in the local fragment are taken into account. However, the form of the Partition Evaluator remains the same. Although replication provides availability, the cost of maintaining consistency of replicated data need to be considered. The above suggestion takes only the cost of retrieval into account and not the cost of update propagation.

6.1 Implementation Status

A distributed database design testbed is being developed at the Database Systems R & D Center, Univ. of Florida, Gainesville. The testbed includes several different vertical partitioning algorithms and modules that compare and evaluate the results of these algorithms. The algorithms that were developed as part of this effort are as follows:

Bond Energy Algorithm: Input: Attribute Affinity Matrix (AAM). Output: Clustered Affinity Matrix (CAM). Now it is left to the subjective evaluation of the designer to split the clustered affinity matrix. In order to determine the

split point quantitatively the CAM is given as the input to the Binary Vertical Partitioning Algorithm.

Binary Vertical Partitioning Algorithm: Input: Clustered Affinity Matrix. Output: This algorithm determines the split point along the diagonal of the matrix. The split point splits the attributes into two partitions only. In order to obtain more partitions the the partitioned matrix is given as the input again until no improvement is possible. Thus it is possible to obtain n-ary partitions.

Minyoung Ra's Graphical Algorithm: Input: Attribute Affinity Matrix. Output: This algorithm produces a fixed number of partitions which is determined by the algorithm. Thus it is difficult to have control over the number of partitions to be generated.

Exhaustive Enumeration Algorithm: Input: Attribute Usage Matrix (AUM). Output: This algorithm exhaustively enumerates all possible combinations of the attributes. Hence we can easily choose the number of partitions in the partition scheme. However we are limited by the number of attributes. We have run this algorithm for attribute usage matrices with upto ten attributes. This algorithm is to be modified to incorporate heuristics to reduce the search space. Then it is possible to work with attribute usage matrix with increased number of attributes.

These algorithms are explained in chapter 2 in detail. The figure 6.1 gives an overall outline of the design testbed.

More algorithms can be easily "hooked" on to this design testbed prototype.

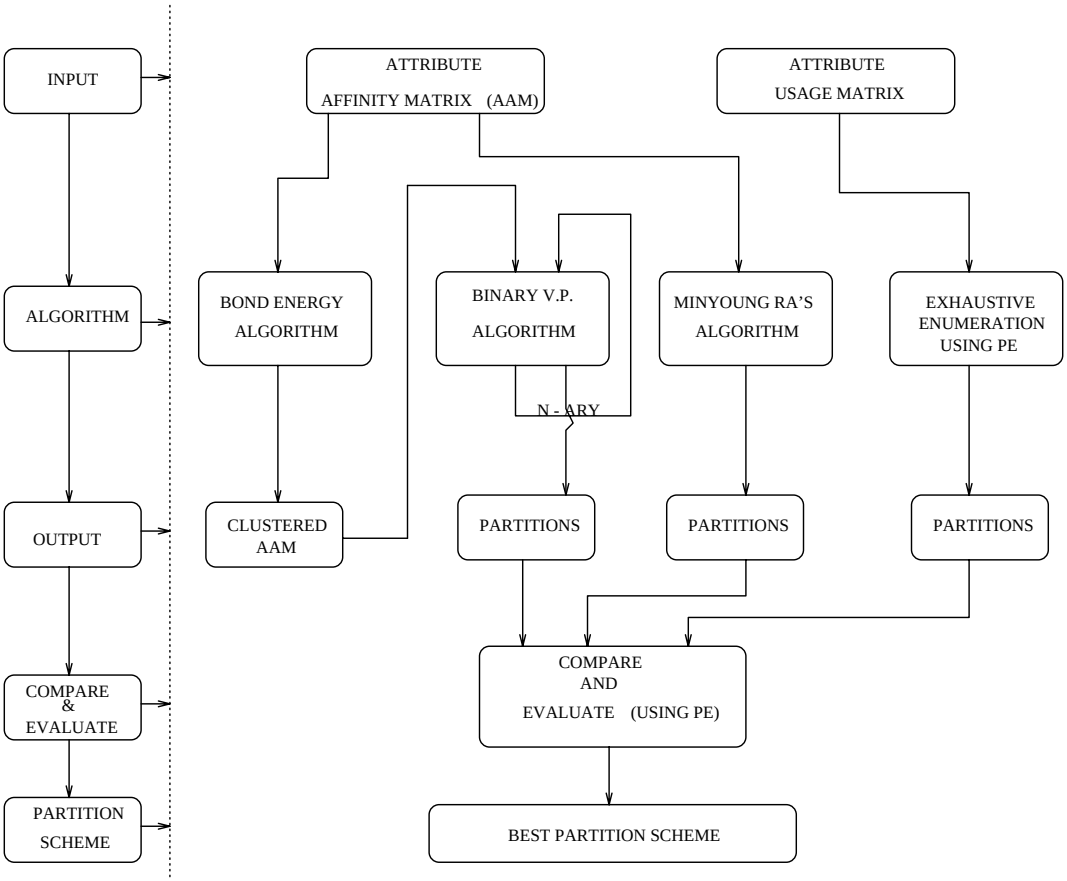

Figure 6.1. Distributed Database Design testbed prototype

CHAPTER 7
Summary and Future Work

In this paper, we have presented a general approach to the vertical partitioning problem. Our study brought out the problems associated with the use of attribute affinity matrix currently used in almost all of the earlier data partitioning algorithms. We started with the objective function used in clustering methods and extended it to suit the requirements of database design. As a result we have brought together work in two isolated areas and established a correspondence between them. Using the objective function derived in this paper and the approach proposed, one can evaluate any vertical partitioning algorithm that uses the same input as our model. We wanted to identify and express an objective function quantitatively before embarking on the development of algorithms. It will now be easier to develop algorithms (heuristics-based or otherwise) exploiting the properties of the function being optimized. Our Partition Evaluator satisfies the need for a common criteria or objective function and can be used to compare and evaluate the extant vertical partitioning algorithms. Finally, the PE developed in this paper has the flexibility to incorporate additional design information such as type of queries (retrieval/updates), allocation information about the partitions, transmission cost and the transaction usage pattern at any particular site.

We are currently developing heuristic algorithms for the objective function derived in this paper. We are in the process of incorporating other input to the database design process, such as query types, constraints on allocation, transmission cost, transaction usage pattern at any particular site, into the objective function derived in this paper. We plan on comparing different algorithms in more detail using the

partition evaluator. We hope to integrate the the Partition Evaluator into a database design testbed which will help designers to choose the right algorithm for database initial design as well as for redesign.

REFERENCES

[1] A. Alsberg. Space and Time Savings through Large Database Compression and Dynamic Restructuring. *Proceedings of the IEEE*, Vol 63, No. 8, August 1975.

[2] M. Babad. A record and file partitioning model. *Commun. ACM 20*, 1(Jan 1977).

[3] H. Benner. On Designing Generalized File Records for Management Information Systems. *Proceedings of the Fall Joint Computer Conference,* 1967.

[4] S. Ceri, S.B. Navathe, G. Weiderhold. Distribution Design of Logical Database Schemas *IEEE trans. on SW engg. Vol SE-9 No.4*, p 487 - 503, July 1983.

[5] S. Ceri, S. Pernici, and G. Weiderhold. Optimization Problems and Solution Methods in the Design of Data distribution. *Information Sciences Vol 14, No. 3*, p 261-272, 1989.

[6] D. Cornell, and P. Yu. A Vertical Partitioning Algorithm for Relational Databases. *Proc. Third International Conference on Data Engineering*, Feb. 1987, pp. 30-35.

[7] H. Day. An Optimal Extracting from a Multiple File Data Storage System: An Application of Integer Programming *Operations Research*, 13, 482-494, 3(May 1956).

[8] P. Dearnley. Model of a Self-organizing Data Management System *Computer Journal* Vol. 17, No. 1, Feb. 1974.

[9] M. Eisner, and D. Severance. Mathematical techniques for efficient record segmentation in large shared databases. *J. ACM 23*, 4(Oct. 1976).

[10] M. Hammer, and B. Niamir. A heuristic approach to attribute partitioning. *In Proceedings ACM SIGMOD Int. Conf. on Management of Data*, (Boston, Mass., 1979), ACM, New York.

[11] J. Hoffer, and D. Severance. The Uses of Cluster Analysis in Physical Database Design *In Proc. 1st International Conference on VLDB*, Framingham, MA, 1975, pp. 69 - 86.

[12] J. Hoffer. An integer programming formulation of computer database design problems. *Inf. Sci.*, 11(July 1976), 29-48.

[13] A. Jain, and R. Dubes. Algorithms for clustering Data. Prentice Hall Advanced Reference Series, Englewood Cliffs, NJ, 1988.

[14] R. Kennedy. The Use of Access Frequencies in DataBase Organization *Ph.D Dissertation, The Wharton School, Univ. of Pennsylvania*, 155 pp, May 1973.

[15] R. Kennedy. A File Partitioning Model *Cal. Tech. Information Science Tech. Report 2*, May 1972.

[16] J. Kittler. A locally sensitive method for cluster analysis. *Pattern Recognition 8*, 22-33.

[17] S. Lu, and K. Fu. A sentence-to-sentence clustering procedure for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics SMC 8*, 381-389.

[18] W. McCormick, P. Schweitzer, and T. White. Problem Decomposition and Data Reorganization by a Clustering technique *Operations Research*, 20 Sep. 1972.

[19] S. March, and D. Severance. The determination of efficient record segmentation and blocking factors for share data files. *ACM Trans. Database Syst. 2*, 3(Sept. 1977).

[20] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design *ACM Transactions on Database Systems*, Vol. 9, No. 4, Dec. 1984.

[21] M. Ra. Data Fragmentation and Allocation Algorithms For Distributed Database Design. *Ph.D Dissertation*, Database R & D Center, University of Florida, Gainesville, 1990.

[22] S. Navathe, and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. ACM SIGMOD, Portland, June 1989.

[23] B. Niamir. Attribute Partitioning in Self-Adaptive Relational Database System. *Master's Thesis*, M.I.T. Lab. for Computer Science, Jan. 1978.

[24] M. Osman. The Partitioning of a DataBase into Supplies Matching User's Queries. *Computer Center, University of Khartoum* Sudan.

[25] M. Schkolnic. A Clustering Algorithm for Hierarchical Structures ACM TODS, Vol. 1, No. 2, pp. 27-44. March 1977.

[26] Y. Seppala. Definition of Extraction Files and Their Optimization by Zero-one programming. *BIT*, 7, 206-215, 3(1967).

[27] E. Shaffer, R. Dubes, and A. Jain. Single-link characteristics of a mode-seeking algorithm. *Pattern Recognition 11*, 65-73.

[28] M. Stocker and A. Dearnley. Self-organizing Data Management Systems *Computer Journal.* 16, 2(May 1973).

[29] N. Tamer, P. Valduriez. Principles of Distributed Database Systems. *Prentice Hall Englewood Cliffs, New Jersey 07362.*

[30] A. Torn. Cluster analysis using seed points and density-determined hyperspheres as an aid to global optimization. *IEEE Transactions on Systems, Man and Cybernetics SMC 7*, 610-616.

[31] C. Yue and K. Wong. On the Optimal Properties of a Partition Algorithm *IBM Research Report,* RC 3797, March 30, 1972.

[32] C. Zahn. Graph-theoretical methods for detecting and describing Gestalt Clusters. *IEEE Transactions on Computers C 20*, 68-86.