LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-166

INDEX SELECTION IN A SELF-ADAPTIVE RELATIONAL DATA BASE MANAGEMENT SYSTEM

Arvola Y. Chan

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract no. N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

This blank page was inserted to preserve pagination.

MIT/LCS/TR-166

INDEX SELECTION IN A SELF-ADAPTIVE RELATIONAL DATA BASE MANAGEMENT SYSTEM

Arvola Y. Chan

September 1976

LABORATORY FOR COMPUTER SCIENCE

(Formerly Project MAC)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE MASSACHUSETTS 02139

ACKNOWLEDGEMENTS

It is my pleasure to acknowledge the continual encouragement, advice and support that I have received from my thesis supervisor, Professor Michael Hammer, during the research reported in this thesis. I would also like to express my sincere thanks to members of the Programming Technology Division of the Laboratory for Computer Science: Dennis McLeod, Sunil Sarin, Bahram Niamir, Bruce Daniels, Christopher Reeve and Timothy Anderson for the many helps, suggestions, criticisms and comments that they have provided in the course of this work and in the preparation of this document. Special thanks are due to Bahram Niamir for the derivation of the closed form solution for the tuple accessing cost function included in Appendix I.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661.

This report reproduces a thesis of the same title submitted to the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, in partial fulfillment of the degree of Master of Science, August 1976.

INDEX SELECTION IN A SELF-ADAPTIVE RELATIONAL DATA BASE MANAGEMENT SYSTEM

by

Arvola Y. Chan

Submitted to the Department of Electrical Engineering
and Computer Science on August 9, 1976 in partial
fulfillment of the requirements for the Degree of
Master of Science

ABSTRACT

The development of large integrated data bases that support a variety of applications in an enterprise promises to be one of the most important data processing activities of the next decade. The effective utilization of such data bases depends on the ability of data base management systems to cope with the evolution of data base applications. In this thesis, we attempt to develop a methodology for monitoring the developing pattern of access to a data base and for choosing near-optimal physical data base organizations based on the evidenced mode of use. More specifically, we consider the problem of adaptively selecting the set of secondary indices to be maintained in an integrated relational data base. Stress is placed on the acquisition of an accurate usage model and on the precise estimation of data base characteristics, through the use of access monitoring and the application of forecasting and smoothing techniques. The cost model used to evaluate proposed index sets is realistic and flexible enough to incorporate the overhead costs of index maintenance, creation, and storage. A heuristic algorithm is developed for the selection of a near-optimal index set without an exhaustive enumeration of all possibilities.

THESIS SUPERVISOR: Michael Hammer
TITLE: Assistant Professor of Computer Science and Engineering

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The development of large integrated data bases, each serving a wide variety of applications, promises to be one of the most important data processing activities of the next decade. The effective utilization of such data bases is highly dependent on the relationship between their physical organization and the prevailing modes of use to which they are put. In this thesis, we address the problem of optimizing the performance of an integrated data base by automatically adapting its physical organization to changing access requirements.

1. Integrated Data bases

An integrated data base may be defined as a collection of interrelated data stored without harmful or unnecessary redundancy and accessed in a uniform and controlled manner, serving one or more applications in an optimal fashion [Martin75]. It may be viewed as the respository of information needed for performing certain functions in an enterprise. In addition to accesses (continuous retrievals and updates) by application programs for regular control functions, it may be used by interactive users for unanticipated information retrieval for planning purposes.

The profits to be gained from the integration of previously related but highly duplicated data bases are manifold [Martin75, Chamberlin76]. The elimination of unnecessary redundancy leads to reduced storage and updating costs. More importantly, the consistency of information stored in the data base is enhanced, since the possibility of having different copies of the data in different stages of updating is removed. Furthermore, the improved

coherence of the data will significantly increase the usability of the data base. By providing users with the capability of extracting any information that is logically contained in the data base, the generation of extensive printed reports on a scheduled base for manual analysis can often be avoided.

In order for these data bases to be truly effective, the data management systems which support them will have to manifest two important characteristics: data independence and non-procedural access. By data independence we mean that users and their application programs are shielded from knowledge of the actual physical organizations used to represent their data, concentrating on a logical view of the data. This makes the data base easy to use and avoids the need for application programs to change when the data base's physical structure is reorganized. Non-procedural access also makes the data base easy to use; this means the provision of access languages which allow the specification of desired data in terms of properties it possesses rather than in terms of the search algorithm used to locate it in the data base.

2. Relational Data Model

The relational model [Codd70] of data has been proposed as a means of achieving the above goals of data-independence and non-procedural access. The user of a relational data base is provided with a simple and uniform view of the data, a logical view which is completely independent of the actual storage structures used to represent their data. The simplicity of this logical data structure lends itself to access by means of easy-to-use languages, which provide associative referencing (content addressing) of the data base contents.

Specifically, a relational data base consists of a collection of relations - a relation is a named two-dimensional table, which has a fixed number of (named) columns and an arbitrary number of (unnamed) rows (called tuples). Each tuple (t_1, t_2, ..., t_m) represents an entry in the relation; t_i, the ith component of a tuple, is a member of D_i, the domain associated with the ith column. (Henceforth, we will use the terms column and domains interchangably.) The relation EMP depicted in Figure 1 has four columns; for each tuple of the relation, the corresponding columnar values represent the name, age, sex, and salary of the particular employee. Figure 1 represents a snapshot of the relation at a particular point in time; relational data base languages provide users the ability to selectively retrieve or modify individual tuples, as well as insert and delete tuples.

EMP:

TUPLE	NAME	AGE	SEX	SALARY
1	Smith	30	M	16000
2	James	25	M	12000
3	Black	28	F	14000
4	Brown	35	M	20000
5	Jones	20	F	10000
6	White	40	F	16000
7	Gray	35	M	15000
8	Green	20	F	10000
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-

Figure 1

A Sample Relation

The table of Figure 1 is purely the user's logical view of the data base; there are no stipulations as to how this data would actually be stored on the computer.

In order to find the names of all male employees making more than $15,000, the user might express a query [Astrahan75] as

```
SELECT  NAME
FROM    EMP
WHERE   SEX = 'M'
AND     SAL > 15000.
```

The query language processor would translate this specification of the desired information into searches on the data base that utilize the precise storage structures and auxiliary access mechanisms used to store the data in order to locate the desired tuples and retrieve the specified column values.

3. Relational Data Base Implementation

Because of the distance of the user's view of a relational data base (and of his queries against it) from the realities of the data base's physical organization, more responsibility is placed on a relational data base system than on a conventional system. This responsibility takes two forms: choosing the physical representation for a relation; and optimizing the execution of queries against a relation, making optimally efficient use of the available access structures. Relational data base systems must possess "intelligence" in order to make decisions in these areas, which have heretofore been the province of human decision-makers.

We believe that the selection of good storage structures is the primary issue in relational data base implementation, since the efficiency that can be achieved by a query optimizer is strictly delimited by the available storage structures. Furthermore, the efficient utilization of

a data base is highly dependent on the optimal matching of its physical organization to its access requirements, as well as to other of its characteristics (such as the distribution of values in it). (For example, certain data base organizations are suitable for low update - high retrieval situations, while others yield optimal performance in opposite circumstances.) Hence, the usage pattern of a data base should be ascertained and utilized in choosing its physical organization. In addition, when viewed as the repository of all information used in managing an enterprise, an integrated data base can no longer be considered as a static entity. Instead, it is continually changing in size, and its access requirements gradually alter as applications evolve and users develop familiarity with the system. Accordingly, the tuning of a data base's physical organization to fit its usage pattern must be an ongoing process.

In current relational data base systems, the data base administrator (DBA) may make recommendations to the system about desirable auxiliary access structures, but his judgements are based largely on intuition and on a limited amount of communication with some individual users. For large integrated data bases, a more systematic means for acquiring information about data base usage, and a more algorithmic way of evaluating the costs of alternative configurations, will be essential. A minimal capability of a data base management system should be the incorporation of monitoring mechanisms that collect usage statistics while performing query processing. A more sophisticated system would sense a change in access requirements, evaluate the cost/benefits of various reorganization strategies, and choose an optimal structure to be recommended to the DBA; eventually, such a system might itself perform the necessary tuning.

4. Architecture of a Prototype Self-Adaptive DBMS

The work to be reported in this thesis is part of an ongoing research effort to develop a self-adaptive data base management system. The intent of this development is twofold: to develop the techniques and methodology for the construction of such systems, and to do performance analysis of these techniques so as to assess their costs and payoffs.

The operation of the initial version of the prototype system is envisioned as follows. The specifications of data base interactions, by both interactive users and application programmers, will be expressed in a non-procedural language; these are first translated into an a high level procedural system level interface language, which is then interpreted by the system modules. The language processor has available to it a model of the current state of the data base, which contains, among other things, a description of the current physical organization of the data base, and estimations of the characteristics of the data base's current contents. Using this information, the language processor can choose the best strategy for processing each data base operation in the current environment. Statistics gathering mechanisms are embedded within the system modules that interpret the object code of the language processor, and record data concerning the execution of every data base transaction. The statistical information gathered for a run is deposited in a collection area and summarized from time to time. When the reorganization component of the system is invoked (which will be at fixed intervals of time), the statistical information collected over the preceding interval is combined with statistics from previous interval and used to obtain a forecast of the access requirements for the upcoming interval; in addition, a projected assessment of various characteristics of the data in the data base is made. A near-optimal physical organization for the data base is then determined heuristically; optimality means

with respect to total cost for the upcoming interval, taking into account the storage and maintenance cost of any auxiliary access structures. This cost is compared with the projected cost for the existing organization. Reorganization will be performed only if its payoff is great enough to cover an appropriate fraction of its cost as well as that of application program retranslation.

5. Thesis Objectives

The principal goal of this thesis is to develop the techniques and methodology for the construction of self-adaptive data base management systems. At its heart, this is a problem in pattern recognition, statistical forecasting and artficial intelligence: first, to extract from a mass of statistics relating to data base performance a succinct pattern which characterizes its mode of use; second, to apply forecasting techniques developed in management science in the detection of shifts in usage pattern and the projection of upcoming access requirements; third, since an exhaustive consideration of all possible structures is computationally infeasible, to develop efficient heuristics that can use the projected usage pattern to synthesize a near-optimal structure.

The continuous monitoring of accesses to a data base opens up many possibilities for its reorganization. Rather than providing a comprehensive study on reorganization possibilities in a data base management system, we have limited the scope of our initial investigation to a well-defined aspect of data base reorganization, so as to obtain some concrete results. We have chosen as the vehicle for this study the problem of index selection in a relational data base. A secondary index (sometimes referred to as an inversion) is a well-known software structure which can improve the performance of

accesses to a relation (file) [Bleir67, Date75, Martin75]. For each domain (field) of the relation that is indexed, a table is maintained, which for each value of the domain in question contains pointers to all those tuples (records) whose contents in the designated domain is the specified value. Clearly, the presence of a secondary index for a particular domain can improve the execution of many queries that reference that domain; on the other hand, manitenance of such an index has costs that slow down the performance of data base updates, insertions, and deletions. Roughly speaking, a domain that is referenced frequently relative to its modification is a good candidate for index maintenance. The choice of which (if any) domains to index must be done with care; a good choice can significantly improve the performance of the system, while a bad selection can seriously degrade it. The goal of our system is to make a good choice of those domains for which to maintain secondary indices, based on how the data base is actually used.

6. Approach

There have been a number of previous studies on the index selection problem [Lum71, King74, Stonebraker74, Cardenas75, Held75b, Farley75, Scholnick75]. However, we feel that the results that have been obtained are not directly applicable to a complete or general data base environment. Some of these have been formal analyses which have made many simplifying assumptions in order to obtain analytic solutions; others have been system designs that are incomplete or unrealistic in many ways. Our thrust here is to relax many of the simplifying assumptions made in previous studies and to develop more complete and accurate models of costs and accesses. In addition, we will stress the importance of the acquistion of accurate parameters to the cost model, an area which is of special significance in a dynamic environment where access requirements are continually changing, but which

has hardly been addressed in previous works. Four basic components of our investigations can be identified:

(1) the development of accurate cost models for the processing of data base transactions under different indexing organization (i.e. when different sets of domains are indexed);

(2) the identification of the set of usage parameters that succinctly characterize the data base usage and which can be inexpensively acquired during the processing of data base transactions;

(3) the application of appropriate forecasting techniques to detect and respond to shifts in access patterns and data characteristics;

(4) the design of heuristic computation procedures that exploit the structure of the index selection problem in the synthesis of a near-optimal data base organization (i.e. choosing a near optimal index set) at a reasonable cost.

7. Organization

The rest of this thesis is organized as follows. Chapter Two summarizes the data base environment which we shall utilize: the data model, the transaction model, the storage and index organizations, and the various assumptions we have made. In Chapter Three, we present our cost analysis for various basic operations in the data base and describe the objective cost function that we will attempt to minimize. Then in Chapter Four, we explain

how parameters needed by our cost model are acquired through statistics gathering and application of forecasting techniques. In chapter Five, we argue for the need of heuristics for the solution of the index selection problem and describe the heuristics we have devised. Finally, Chapter Six includes summary, conclusions and suggestions for future research.

CHAPTER 2
DATA BASE ORGANIZATION

In this chapter, we describe the data base environment we have assumed in our research. Our discussions will be based on a rather general model of the data base, one which can readily be extended to characterize a large variety of existing systems. We will describe the storage and access structures in the data base, the kinds of transactions that may be conducted against it, and the way transactions are processed. In addition, we will contrast our assumptions with those employed in previous studies which we feel to be incomplete or unrealistic.

1. File Model

As we have said, we operate in the environment of a relational data base. The totality of formatted data in the data base consists therefore of one or more relations. However, we address here the reduced problem of selecting indices for a data base made of a single relation. (We expect that extensions can be made to the general multi-relation case.) Even though insertion and deletion of tuples are permitted in our transaction model, we will assume that the cardinality (number of tuples) of the relation remains relatively unchanged between two consecutive points at which index selection is considered (i.e. the rate of change in size of the data base per review interval is small).

2. Access Model

Previous studies on index selection have often assumed rather unrealistic access models:

both King [King74] and Schkolnick [Scholnick75] have assumed that the cost of accessing an arbitrary subset of all the tuples in a relation is directly proportional to the size of the set. This will be true only if all tuples are equally accessible, as in the case when they all reside in primary memory, or equally inaccessible, as in the case when each is independently stored on secondary storage. For data bases of reasonable size and reasonable tuple length, neither assumption will hold.

In this study, we will assume that the totality of the data base (both the stored representation of the relation and the set of secondary indices that are maintained) resides on direct access secondary storage devices like disks [Rothnie74, Blasgen76]. Physical storage space on such devices is partitioned into fixed size blocks called pages. The page is the unit of memory allocation and the unit of transfer between main memory and secondary storage. The accessing cost of a page is assumed to be independent of the sequence of page accesses. Furthermore, we will assume that the system is I/O bound, so that page accessing cost dominates all other internal processing costs. Hence, the processing cost for a data base transaction is measured solely in terms of the number of pages that have to be accessed in its processing.

3. Tuple Organization

We will assume that tuples are of fixed length (i.e. each occupies the same amount of physical storage space) so that each page has a capacity for a fixed number of tuples. To retrieve all the tuples in the relation, a scan of all the pages on which the tuples reside can be performed. (Henceforth, we will refer to these pages as the segment on which the relation is stored.) The cost of this sequential scan is just p pages, where p is the number of

pages in the segment. However, in many instances, only a small subset of the tuples will actually be required for processing; hence, it is desirable to provide additional access paths to enable access to just those tuples that are needed. In other situations, all tuples may be required, but in a specific sort order. If the required ordering is different from the one in which tuples are physically stored, then sorting will be required; for typical sizes of data bases, an external sort would be in order, and would entail going over the data in several passes. Hence, it is desirable to physically cluster together tuples that are needed together. Held and Stonebraker [Held75b] have investigated a variety of organizations for storing tuples of a relation on pages of a direct access file, and have made a broad categorization of keyed structures versus non-keyed structures. A key structure is one in which a domain (or a combination of domains) is used to determine where in secondary storage the tuple should be stored. The advantage of a keyed structure is that tuples that are often needed together can be clustered together physically. However, any modification to a tuple in the keyed domain(s) will require the tuple to be relocated. Hence, all index entries that point to this tuple will have to be modified also. A non-keyed structure is one in which the tuples are stored using some criteria that is independent of the value of the tuple. The advantage of a non-keyed structure is that it enables auxiliary access structures like indices to be maintained more economically.

For the purpose of this thesis, we will assume that the tuples in the relation are organized as a non-keyed structure. We will assume that they are stored sequentially on the pages of the segment without any preferred ordering. (For example, they might be stored according to their chronological order of insertion into the data base.) Since the cost of a sequential scan is dependent on the number of pages in the segment, it is essential that the storage utilization in the tuple space be maximized, so as to minimize the cost of segment scans. We

will assume that all empty spaces resulting from the deletion of tuples will be reused for newly inserted tuples, before a new page is allocated for the segment. (This can readily be done by keeping a linked list of the empty spaces in the segment. The linked list can be stored in the empty spaces in the segment itself. Only a pointer to the head of the list need to be maintained separately for the purpose of storage allocation in the tuple space.) Even with the above assumption, poor storage utilization can still result from a long sequence of tuple insertions followed by a long sequence of tuple deletions. On the other hand, garbage collection in the tuple space would have to be accompanied by the modification of all those index entries for tuples that are relocated. To simplify our discussions here, we will finesse the need to garbage collect in the tuple space by assuming that there are no clustered deletions of tuples from the same page, and that the general trend is for the data base to grow in size. (Note that we could readily include garbage collection overhead in our cost model by monitoring the average number of tuples that are relocated per review interval, in addition to the actual number of insertions, deletions and modifications, towards the estimation of index maintenance cost.)

4. Index Organization

We assume that each tuple in the relation has associated with it a unique tuple identifier (TID), a logical address which enables the tuple to be located with a single page access. An index on a column of a relation is then a mapping from values in the column to TIDs of tuples with those values. Conceptually, an index may be viewed as a binary relation consisting of pairs whose first component is a value from the column and whose second component is the TID of a tuple with that value. Figure 2 shows an index on the column salary for the EMP relation depicted in Figure 1. (This sequential organization is actually

assumed in [King74].) However, as Cardenas [Cardenas75] has pointed out, the organization of the index is itself an important problem in the enhancement of system performance. We will therefore assume that the index is organized in such a way that all those TIDs associated with the same column value are easily accessible. Specifically, we will adopt the VSAM-like tree organization as used in Blasgen's study [Wagner73, Blasgen76]. Figure 3 shows how the index shown in Figure 2 will actually be stored. It is a balanced tree whose nodes are index pages. Leaf pages contain pairs whose first component is a column value and whose second component is a sorted list of the TIDs of those tuples with that column value. The pairs in each leaf-page are sorted on the value of their first component. Higher level pages contains pairs consisting of the identifier of a lower page and the high key value on it. These pairs are also sorted by the values of their first components. The tree is kept balanced on insertion or deletion in a way that is similar to the maintenance of B-trees [Bayer72], with the splitting and merging of pages as necessary.

SALARY	TID
16000	1
12000	2
14000	3
20000	4
10000	5
16000	6
15000	7
10000	8
-	-
-	-
-	-
-	-

Figure 2
Conceptual Organization of an Index

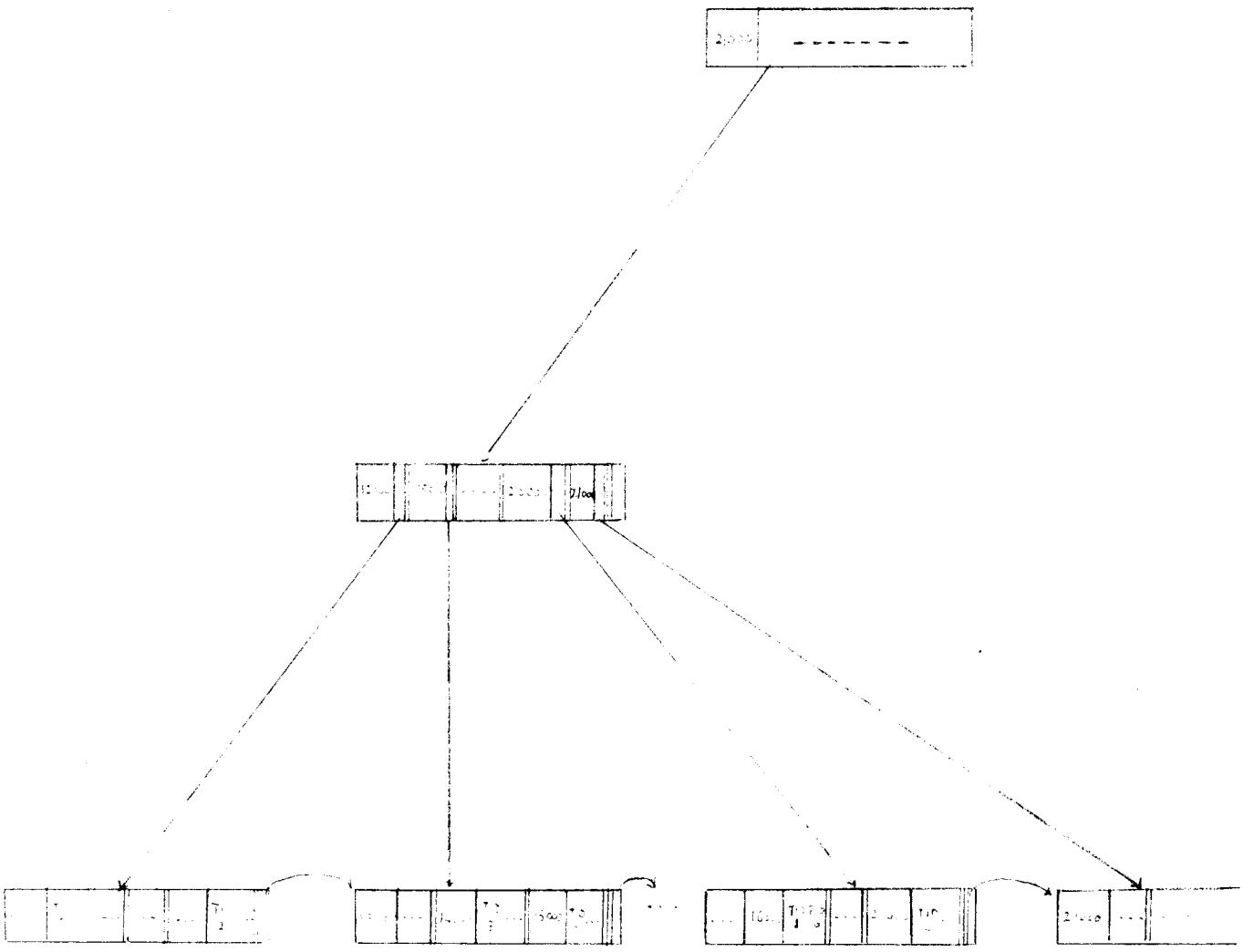

Figure 3

Physical Organization of an Index

5. Transaction Model and Processing

We will consider four types of transactions that may be conducted against the data base: the retrieval, insertion, updating and deletion of tuples. An update or a delete operation is often specified in two components: a selection component which determines the set of tuples to be processed, and an action component (which in the case of an update, determines how each tuple is to be processed). As will be discussed below, the use of an index (or indices) to identify the set of tuples that satisfy (or potentially satisfy) a selection component entails a number of steps, after which we can no longer assume that any part of an index still reside in primary memory. Hence, we can assume that the maintenance to the indices due to an update or delete is independent of the selection component of the transaction (i.e., the maintenance cost of an index due to a tuple deletion or modification is the same regardless of whether the tuple is identified through the use of that index or through a sequential scan). Therefore, we will assume that the data base transactions specified in the source language get translated by the language processor into sequences of queries, updates, inserts, and deletes, as described below.

(1) Query - this can result either from a retrieval specification in the source language or from the data selection component of an update or delete specification as discussed above. It enables those tuples to be retrieved or acted upon to be specified in terms of the properties they possess. In relational access languages that are currently being developed, powerful and general data selection predicates are allowed [Codd71, Boyce74, Astrahan75, Held75a, Czarnik75]. However, in order to be able to evaluate the tradeoffs of a particular index, we shall limit ourselves to the consideration of only those data selection predicates for whose processing the utility of indices can readily be

determined. We will therefore allow only the following predicate types:

(a) a predicate consisting of a single equality condition or a conjunction of two or more equality conditions;

(b) a predicate consisting of a disjunction of two or more equality conditions.

(By an equality condition, we mean a predicate of the form A=k, where A is some domain name, and k is a constant or program variable.)

Henceforth, we will refer to the process of identifying the set of tuples that satisfy (qualify for) the selection predicate associated with a query as the resolving of the query. (A retrieval specification in the source language may in addition to the selection of tuples, specify what fields in the selected tuples are to be output or further processed. However, the time to perform these operations is independent of which indices are maintained, and so will be ignored in our disucssion here.) For a query arising from a delete or update specification in the source language, we will assume that each qualified tuple is returned accompanied by its TID, thus allowing it to be identified in subsequent delete or update operations.

An index (or a set of indices) can be used to totally or partially resolve a query. (A query is said to be totally resolved when the exact set of tuples that satisfy the associated selection predicate is identified, and it is said to be partially resolved if a superset (but one which is smaller than the entire set of tuples in the relation) of those tuples that satisfy the associated selection predicate is identified. This partially

qualified set of tuples will have to be brought into main memory and each tuple must be examined to determine if it satisfies the full predicate.) Given a data selection predicate and an existing set of indices, there are a number of possible strategies for obtaining the set of selected tuples. Depending on the nature of the predicate and the selectivities of the domains involved, it may be most profitable to use none, all, or a subset of the applicable indices. In previous studies, it is assumed that indices are used whenever they are available. However, as we will see from our cost analysis in the next chapter, there may be situations in which it would be most economical to use less than the full set of applicable indices in resolving a query. For simplicity, we will assume that the query processor uses the following decision criterion: it will evaluate the expected cost of processing the query using the full set of applicable indices (i.e. existing indices on those columns which are specified in the query) and will use all of them only if this expected cost is less that of a sequential scan; otherwise a sequential scan will be utilized. (Note that a disjunctive query will be resolved using indices only if indices on all of the domains referenced in the query are available; a tuple that does not satisfy any of the predicates resolved through indices can still potentially satisfy those predicates on domains for which no indices exist, and hence some qualified tuples can only be identified through a sequential scan of the entire segment.)

We will assume that a query is processed using indices as follows:

(a) For each domain specified in the query and for which an index exists, a list containing the TIDs of all those tuples that satisfy the equality condition on the column in question is obtained;

(b) Depending on whether the selection predicate is a conjunction or disjunction, an intersection or union list of all those lists obtained in step (a) is formed. This restricted list contains the TIDs of all those tuples that satisfy the conjunction or disjunction of those equality predicates involving domains for which indices exist;

(c) This restricted set of tuples is brought to main memory for further processing. (In the case of a conjunctive query that has only been partially resolved, i.e., the restricted set of tuples only satisfy the conjunction of those equality predicates involving domains which are indexed, each of the restricted tuples is checked against the equality conditions involving the non-indexed domains and then discarded or retained accordingly. (This is sometimes known as the removal of false-drops [Schkolnick75].)

(In the process of obtaining the TID list for the restricted tuples that have to be accessed, it is possible that some of the TID lists involved are so long that they cannot completely reside in primary memory. Therefore, we will assume that the list manipulation phase is combined with the tuple access phase: i.e., we will assume that the individual TID lists are in the same sort order, so that the union or intersection process can be carried out in a single pass over all of the participating lists [Welch76]. By utilizing portions of the resulting TID list as soon as it is available, extra page accessing can be avoided.)

(2) Insert - this inserts a single tuple into the relation. It is specified by supplying a value for each of fields in the tuple to be inserted, and results in the tuple's insertion into the main file, together with the necessary maintenance to the existing set of indices.

(3) Delete - this deletes a single tuple from the relation. It is specified by supplying the TID of the tuple to be deleted, together with values in different fields of the tuple, and results in the deletion of the tuple from the stored representation of the relation, and the necessary maintenance to the existing set of indices to reflect this deletion.

(4) Update - this involves a single tuple in the relation. It is specified by three components: the TID of the tuple to be updated, its old component values, and its new component values. It causes the tuple to be updated, and the indices on the affected domains modified accordingly.

6. Query Distribution

In earlier index selection studies, simplifying assumptions on query distributions are often adopted. In [King74], it is assumed that selection predicates only consist of of single equality conditions. Hence, it is sufficient to summarize the statistics on query distribution by the probabilities of each domain being specified in a selection predicate. In [Scholnick75], the restriction to the consideration of single-domain queries is relaxed, but with the imposition of the new assumption that domain occurrence probabilities in queries are independent. Hence, the model that is used there is unable to account for the positive or negative correlation among domain occurrences in queries; such correlation is common in the usage of real data bases. (For example, in queries on the EMP relation (Figure 1), age and salary might often be specified together while name is more likely to be specified alone.) In this study, we will do away with any such simplifying assumption by observing the occurrence frequencies of those queries that actually occur.

7. Domain Value Distribution

In earlier work, it is often assumed that the set of distinct values in a domain is evenly distributed among tuples in the relation, and that all domain values are equally likely to be specified in the constant part of a selection predicate. Consequently, the average fraction of tuples that can be expected to satisfy an equality condition on a domain is assumed to be the reciprocal of the number of values in the domain. However, in a real data base, it is often the case that the distribution of domain values among tuples and in query specifications are skewed; i.e. some values are used more often than others. We would like to take advantage of our continuous monitoring facility to detect such situations. We will therefore define a new measure for the resolving power of a domain index. We define the average selectivity of a domain as the average fraction of tuples under consideration that have historically satisfied an equality condition involving that domain.

Since we allow the specification of multiple domains in queries, it is necessary to have a measure for the joint resolving power of two or more domain indices. For this purpose, we will assume that the specification of values from different domains in a query are uncorrelated (i.e., given that a query specifies two columns A and B, the probability of a particular key value in column B being specified is independent of the value in column A that is specified). Hence, the joint conjunctive selectivity of a set of domains D, each with average selectivity AS_i is

(2.1) $$\Pi_{i \in D} AS_i$$

(The interpretation of this expression is that the expected fraction of tuples that satisfy a

number of predicates simultaneously is equal to the product of the individual expected fractions that satisfy the predicates.) Similarly, the joint disjunctive selectivity of a set of domains D, each with average selectivity AS_i is

(2.2) $$1 - \Pi_{i \epsilon D} (1 - AS_i)$$

(The interpretation of this is that the expected fraction of tuples that satisfy a disjunction of equality conditions is the complement of the fraction expected not to satisfy any of the equality conditions in the disjunction.)

8. Objective of Index Selection

We assume that index selection will be reconsidered at fixed intervals and that usage statistics are collected during the processing of each transaction in the data base and summarized at the end of each interval. The objective of the selection process is to minimize the total system cost for the upcoming interval. This total cost includes retrieval processing; index creation, maintenance and storage; and application program retranslation. In contrast with previous studies, we have chosen to minimize this total cost, rather than using a probabilistic model of data base transactions and attempting to minimize only the expected cost of an average transaction. Our information on the absolute level of activities in the data base (in addition to their relative levels) allows us to amortize such cost as index creation, index storage and application program retranslation over the data base transactions, rather than completely omitting them from the cost model.

CHAPTER 3
COST ANALYSIS

One of the most important tasks in the analysis and enhancement of performance of any system is the determination of the set of parameters that have a significant effect on performance and the formulation of the cost model relating system performance to these parameters. Since we operate in a dynamic environment, we have to resort to the continuous monitoring of data base transactions to obtain the parameters in our system. As we shall see, most of these parameters can be directly measured. However, there are others that are not directly observable, in which case we have to relate them to statistics that can be readily obtained through statistics gathering. In this chapter, we will analyze the cost of various basic operations in the data base system and then discuss the objective cost function that our index selection procedure will endeavour to minimize.

As we have seen, the processing of a query using indices involves the retrieval of the relevant TID lists from the indices, the manipulation of these lists to obtain an intersection or union list, and the accessing of the restricted set of tuples as identified by the resultant list. As in previous studies, we assume that the manipulation of the TID lists is done in main memory, and is therefore negligible according to our cost criterion of page accesses. Similarly, any need to remove "false-drops" from the restricted set of tuples is done in primary memory at a negligible cost. Hence, the processing cost of a query using indices can be assumed to be made up of two components: the cost of using the relevant indices and the cost of accessing the restricted set of tuples. As regards to modifications to the data base (update, insert and delete), maintenance of the existing indices in addition to modifications of the stored representation of the relation must be made. Since the latter cost is incurred

regardless of what domains are indexed, we will ignore it from further consideration, and concentrate only on the maintenance cost of the indices in determining the costs of data base modification.

1. Tuple Access

In order to compare the utility of different sets of indices towards the processing of a query, we need to have an estimate of the sizes of the sets of tuples that must be accessed in order to resolve the query, given the availability of each of the index sets. We have earlier defined the average selectivity of a domain as a measure for the resolving power of an index on that domain. Using the selectivity of the domains specified in a query which are indexed, we can estimate the number of tuples that have to be examined to evaluate all the predicates. Since our cost criterion is the number of pages that have to be accessed, we have to translate this expected number of tuple accesses to an expected number of page accesses. We feel most previous index selection studies have been inaccurate in their choice of cost model for the accessing of such a restricted set of tuples. In [Scholnick75] linear relationship between the number of tuples to access and the accessing cost is assumed. This is equivalent to saying that each tuple specified in the resulting TID list will incur one page access. In [Held75b], a piecewise linear relationship is assumed: if the relation is stored as p pages of t tuples each, and r tuples are to be accessed, then the number of page accesses is assumed to be min(r, p). In a paged memory environment in which tuples are blocked together on pages, neither of the above schemes accurately model the tuple accessing process (since the restricted set of tuples can be accessed in the order of their TIDs so that tuples from the same page will incur only a single page access). A more realistic scheme to estimate the accessing cost for r tuples is to assume that they are equally likely to be any r tuples in the

segment, and to use the expected number of page accesses for r randomly selected tuples in the relation as an estimate for the tuple accessing cost for a query whose resolution under the availability of a particular set of indices is expected to require the retrieval of r tuples. This expected number has been considered in a number of previous studies [Rothnie72, Schmid75, Yue75]. However, the formulations that have been derived are often computationally infeasible or inaccurate. Based on a Markov model approximation to the underlying process of accessing r randomly selected tuples, Rothnie [Rothnie72] has obtained a lower and upper bound on the expected number of pages that have to be touched. Schmid and Bernstein [Schmid75], using a combinatorial analysis, have derived an exact formulation that involves a complicated recurrence relation whose computation for moderate values of the parameters becomes very costly and inaccurate because of the significant round-off errors encountered. The following formulation, due to Yue and Wong [Yue75], is by far the most satisfactory.

Let n = number of tuples in segment

t = number of tuples per page

p = number of pages in segment

$f(r)$ = expected number of page access for r randomly selected tuples, then

(3.1) $$f(0) = 0$$

(3.2) $$f(r+1) = \frac{t(p-1)-i}{pt-i} f(i) + \frac{pt}{pt-i}$$

The value of f for an arbitrary value of r can be computed from the recurrence relation with relatively little round-off error. However, this computation involves r multiplications and r divisions and is therefore quite expensive to carry out. We (in conjunction with

Bahram Niamir of the MIT Laboratory for Computer Science) have obtained a closed form solution of the above difference equation which can be computed more efficiently.

$$f(r) = \frac{n}{t}\left(1 - \frac{\binom{n-r}{t}}{\binom{n}{t}}\right) \qquad (3.3)$$

A detailed derivation of this formulation is included in Appendix 1. The above formulation also admits of a simple interpretation. Consider an arbitrary page in the segment; the probability that it does not contain any of the r desired tuples is equal to the number of ways of choosing t tuples from n - r tuples, divided by the number of ways of choosing t tuples from n tuples. Hence, the expression within the parenthesis gives the probability that this page contains one or more of the r desired tuples. Thus, multiplying this expression by the total number of pages in the segment gives the number of pages expected to contain one or more of the desired tuples, i.e., the expected number of page accesses.

For a fixed value of p (say 1000), and for a typical value of t (say 50), the shape of the function f(r) is shown in curve #3 of Figure 4. It is instructive to note that for values of r close to, but less that p, the value of f(r) is roughly 0.6p, which is substiantially different from the value given by a linear cost function. (Curve #1 indicates a linear cost function [Scholnick75] while curve #2 indicates a piecewise linear cost function [Held75b].)

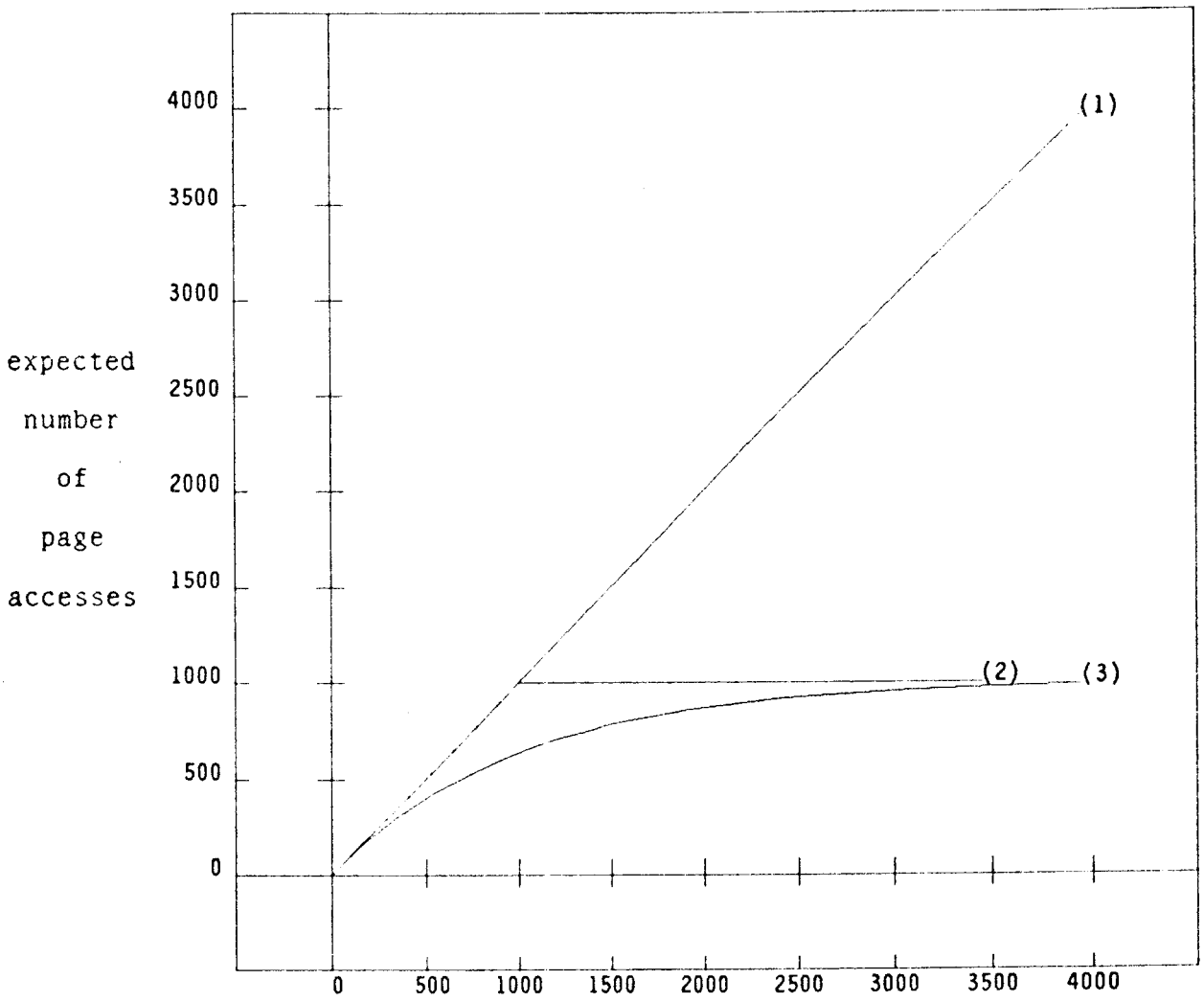

Figure 4

Tuple Accss Cost Functions

2. Index Creation

In order to determine if reorganization is worthwhile, we need to have an estimate for its costs and its payoffs. Two major components of the reorganization cost (due to a change in indexing policy) are:

(1) cost of retranslating existing application programs;

(2) cost of creating the new indices.

The former can be estimated from the previous translation costs of the individual application programs. (In many systems, the data manipulation language is interpreted in which case no retranslation cost is incurred as a result of physical reorganization.) The latter, in general, depends both on the current size of the relation and on the number of distinct key values in each of the domains. we assume that an index is created as follows.

(1) For each tuple in the relation, a pair consisting of the value of the tuple for the indexed domain, and of the tuple identifier is formed.

(2) These pairs are sorted, with the domain value as the major sort key, and the tuple identifier as the minor sort key. (Typically, this will involve an external sort consisting of a sorting and a merging phase.)

(3) A data structure (see Figure 3) that facilitates the accessing of the list of tuple identifiers for tuples associated with any value in the domain is constructed from the

sorted pairs of domain values and tuple id.

Let n = number of tuples in relation

p = number of pages in segment

w = number of words per page

b = number of pages available for internal buffering

ln = number of words in the representation of a key value in the domain

Step one involves the scanning of the segment and the formation of the pairs of key value and TID. For practical sizes of the data base, these pairs have to be written back to secondary memory for temporary storage. This can be combined with the initial internal sorting phase of step two with a cost of

(3. 4) $$p + \lceil n(ln+1)/w \rceil$$

where p is the cost of scanning tuples in the segment and $\lceil n(ln+1)/w \rceil$ is the number of pages needed for the writing out of the n pairs (each of length ln+1) of domain value and TID into the initial sorted subfiles. Let

(3. 5) $$p' = \lceil n(ln+1)/w \rceil$$

Then we will assume that at the end of step one, s - 1 subfiles of length b and one of length b' are formed where

(3. 6) $$s = \lceil p'/b \rceil$$

$$(3.7) \qquad b' = \begin{cases} b & \text{if } \mathrm{mod}(p', b) = 0 \\ \mathrm{mod}(p', b) & \text{otherwise} \end{cases}$$

The cost of merging these s subfiles is derived in Appendix 2 and is given by

$$(3.8) \qquad C_{merge}(s - 1, b')$$

However, we note that in the last pass of the merging process, instead of writing out the $b * (s - 1) + b'$ pages for the single sorted file, we will build the VSAM-like tree for the index. Hence, the cost of the second phase of the index creation procedure is

$$(3.9) \qquad C_{merge}(s - 1, b') - (b * (s - 1) + b')$$

Finally, the cost of the third phase consists of writing out the leaf and node pages of the index tree and can be estimated as follows. (We will assume that pages in an index are not filled to capacity at creation time, so as to facilitate subsequent modifications.)

Let u_n = initial fraction of utilization in a node page

u_l = initial fraction of utilization in a leaf page

v = number of distinct key values in the indexed domain

c = number of key pointer pairs a node page can contain

k = initial number of key pointer pairs with which a node page is filled ($=u_n * c$)

then the number of leaf pages lf is

(3.10) $$lf = (v * ln + n)/(u_1 * w)$$

The height h of the index tree is

(3.11) $$h = \lceil \log_k lf \rceil$$

(where the leaf nodes are at height 0), and the number of node pages is

(3.12) $$\begin{aligned} & \lceil lf/k \rceil + \lceil \lceil lf/k \rceil /k \rceil + \lceil \lceil \lceil lf/k \rceil /k \rceil /k \rceil + \ldots\ldots \\ & \cong lf/k + lf/k^2 + lf/k^3 + \ldots\ldots \\ & = lf/(k - 1) \end{aligned}$$

From the above analysis, we also have a rough estimate for the storage requirement of an index on the domain in question, which is

(3.13) $$lf + lf/(k - 1)$$

The above analysis has been motivated by the need to estimate the costs of index creation and storage. However, it depends very much on the number of distinct keys in the indexed domain, for which we can only have a rough estimate. Consequently, it will be difficult to come up with a close estimate of the index creation and storage costs.

3. Index Accessing and Maintenance

(The average cost of using an index as well as the total maintenance cost of an index within

an interval can be directly measured. The purpose of the analysis below is only for the purpose of estimating these parameters for those domains which are not indexed.)

The use of an index to obtain the list of tuple identifiers with a particular value in the indexed domain involves starting from the root of the tree, and following a path through the node pages which leads to the leaf page containing the desired TID list. Let h be the height of the index (where the root of the tree is at height 0, and the leaf pages are at height h), then the cost of using the index to obtain an average TID list can be estimated as

(3.14) $$h + \lceil lf/v \rceil$$

Similarly, the cost of modifying a TID list in a leaf of the index (as a result of a tuple insertion or deletion) when no overflow or underflow is incurred, is

(3.15) $$h + 2 \lceil lf/v \rceil$$

(The maintenance to an index due to the update of a tuple in the indexed domain can be assumed to be the sum of the maintenance due to a delete and an insert.) The cost of index maintenance due to the splitting and merging (or garbage collection) in the index is more difficult to parameterize, since it depends on the actual sequence of tuple insertions, deletions and modifications. This component has often been completely ignored in previous studies. Here, we can add to the above cost an average overhead cost per modification, a parameter which can be obtained by monitoring the actual maintenance of an index. For those domains that have not been previously index, the normalized average overhead among those indices that have been maintained can be used as a very rough estimate.

4. Total System Cost

With full knowledge of the upcoming requirements, the total system cost for the next interval under a particular indexing policy is computed as follows. For each query type, we can determine, using the selectivities of the domains that occur both in the query and in the proposed index set, how many tuples will need to be scanned to resolve this query type, with the full use of the indices. Our non-linear cost function translates this into an expected number of page accesses. To this is added the expected number of page accesses that are involved in accessing the indices themselves. This then gives us the total processing cost for this query type, if the proposed indices are used. We then know if the query processor would, given the proposed index set, use them to resolve this query type or would process it by means of a sequential scan. In any event, we thus know the projected cost of processing this query type in the presence of the proposed set of indices. We multiply this cost by the expected frequency of this query type, and repeat the process for all the query types. This gives the projected total query processing cost. Adding to this the projected indexing costs (creation (if applicable), maintenance (due to tuple insertion, deletion and modification) and storage) and the application program retranslation costs (nil for the index set which is identical to the one that is maintained in the previous interval) yields the total system cost for the next interval.

Let CC_i = expected creation cost of index on domain i (if not already exist)

MC_i = expected maintenance cost of index on domain i

SC_i = expectedstorage cost of index on domain i

AC_i = expected cost of obtaining a TID list using an index on domain i

AS_i = average selectivity of domain i

Q = projected set of queries in upcoming interval

F_q = occurrence frequency of query q where $q \in Q$

n = average number tuples in relation

p = average number of pages in stored representation of relation

D_p = set of domains indexed in the previous interval

D_q = set of domains specified in query q

T_q = type of query q (0 if conjunctive, 1 if disjunctive)

$RC(D)$ = application program retranslation cost, 0 if $D = D_p$

$I_q(D)$ = 1 if $D_q \subset D$, 0 otherwise

$C_q(D)$ = cost of processing query q with the index set D

$$= (1-T_q)*\min(p,((\Sigma_{i \in D \cap D_q} AC_i)+f((\Pi_{i \in D \cap D_q} AS_i)*n))) +$$
$$T_q*((1-I_q)*p+I_q*\min(p,((\Sigma_{i \in D_q} AC_i)+f((1-(\Pi_{i \in D_q}(1-S_i)))*n))))$$

The objective of the index selection procedure is then to select the index set D which minimizes the following expression:

(3.16) $$\Sigma_{i \in D} (CC_i + MC_i + SC_i) + \Sigma_{q \in Q} F_q * C_q(D) + RC(D)$$

CHAPTER 4
PARAMETER ACQUISITION

A fundamental problem in an adaptive system operating in a dynamic and uncertain environment is the exploitation of new information in reducing the uncertainty of the system. In our context, this involves the utilization of observed data on how access requirements and data characteristics change over time in the estimation of exogenous (uncontrollable) parameters essential for predicting the performance of different indexing organizations for the planning horizon. In the following sections, we will describe the statistics that are to be collected during transaction processing. We will then explain our choice of forecasting technique and how the various parameters in the system are to be forecasted.

1. Statistics Gathering

Statistics are collected during the procesing of each data base transaction for two purposes:

(1) as a direct measurement of certain system parameters in the current time interval;

(2) to be used in the indirect estimation of certain system parameters, parameters that cannot be measured directly or whose direct measurement would entail excessive overhead.

The statistics to be gathered for the purpose of index selection fall into four general classes.

(1) Index Maintenance Statistics - This has several components. First of all, there is the total maintenance cost of each active index in the current interval. For domains that are not indexed, we need to obtain an estimate of the cost that might have been expended had an index been maintained on each of these domains. For this purpose, we record the total number of tuples that are deleted from and inserted into the relation in the current interval, and the number of updates to each domain in the tuples. In addition, we will break down the maintenance cost of each active index into two parts: the cost of basic maintenance to a leaf in the index tree, and the more difficult-to-parameterize costs of node splitting and merging necessary for maintaining the index as a balanced tree. This will allow us to calulate the normalized node splitting and merging overhead per insertion or deletion (an update can be counted as a delete and an insert) in the active indices, which will be used in estimating the cost of maintaining an index on a domain which is not indexed in the current interval.

(2) Query Type Statistics - The type of a query is determined by the set of domains it utilizes and by whether it is a conjunction or disjunction of equality predicates. We record the occurrence of each query (this can be encoded as a bit pattern) and then summarize the occurrence frequencies of each query type from time to time.

(3) Domain Selectivity Statistics - For each domain, we maintain its average selectivity over all uses of the domain in equality conditions in the current interval. This is accomplished by recording the number of times the domain occurs in equality conditions and the sum of the selectivities of the domain in each of these predicates. If an index for the domain is used to resolve the particular equality condition, then the precise selectivity of the domain for this query can be calculated as the fraction of

tuples in the relation with the domain value in question. If the equality condition is resolved through a sequential scan, the selectivity of the involved domain has to be calculated in a reduced tuple space and then extrapolated to the entire tuple space. This is necessary for two reasons: in the first place, the scan may be of a reduced set of tuples identified through the use of an index (or indices); secondly, we assume that the query resolver is efficient in that it will avoid the unnecessary checking of tuples against equality predicates (i.e., avoiding testing subsequent predicates in a conjunction once one has evaluated to false, or in a disjunction once one has evaluated to true). The estimation of selectivity can be done as follows:

(a) Suppose the equality condition appears in a conjunction of conditions of the form

$$C_1 \wedge C_2 \wedge \cdots\cdots \wedge C_n$$

where each of the C_i is an equality condition involving domain D_i. (We assume that the ordering of the equality conditions above reflects the order of conditions against which a tuple is checked.) Let N_0 be the total number of tuples scanned, and let N_1, N_2, ..., N_n be the number of tuples that satisfy C_1, $C_1 \wedge C_2$, ..., $C_1 \wedge C_2 \wedge \cdots\cdots \wedge C_n$ respectively. (Note that these numbers are readily available). The selectivity of domain D_i for this query is then approximated as

$$S_i = N_i / N_{i-1} \qquad (4.1)$$

(c) Suppose the equality condition appears in a disjunction of conditions of the form

$$C_1 \vee C_2 \vee \cdots\cdots \vee C_n$$

where each of the C_i is an equality condition involving domain D_i. Let N_0 be the total number of tuples scanned and let $N_1, N_2, \cdots, N_n$ be the number of tuples that satisfy C_1, $\sim C_1 \wedge C_2$, $\cdots$, $\sim C_1 \wedge \sim C_2 \wedge \cdots \wedge C_n$ respectively. (Again, these numbers are readily available). The selectivity of domain D_i for this query is then approximated as

$$S_i = N_i / (N_0 - \Sigma_{1 \le j < i} N_j) \qquad (4.2)$$

(4) Index Accessing Statistics - For each active index, we record the number of times it is used for resolving equality conditions and the total cost expended in such uses. This allows us to obtain the average cost of using the index. As for a domain that is not indexed, we can estimate the number of distinct values in each as the reciprocal of its observed selectivity and use the procedure described in the previous chapter for estimating its average accessing cost.

The foregoing statistics comprise our model of the usage pattern of the data base. The frequency count of the query types, together with the index maintenance statistics constitute the record of transactions with the data base. By recording the types of the queries that actually occur, we detect any correlations (positive or negative) that may exist between the occurrences of different domains in a query (it may happen that some combinations of domains are frequently used together, while others rarely are). Thus we avoid making the strong (and often inaccurate) assumption that the simultaneous occurrences of domains in a query are mutually independent events. (Previous studies have made this assumption, and

so have recorded access history merely as the frequency of each domain's occurrence in queries.) We observe that our measure of domain selectivity serves as a succinct yet precise indication of how a domain is actually used in queries. By averaging the selectivities of the actual occurrences of a domain, we take into consideration both skewness in the distribution of domain values over the tuples as well as non-uniform use of domain values in queries. This measurement of selectivity is more accurate than its conventional estimate as the reciprocal of the number of distinct values in the domain. Finally, we note that all of the foregoing statistics can be collected and maintained with very little overhead, either in execution time or in storage requirements. All of the required information can be easily obtained during query or transaction processing, and requires little space for its recording.

2. Application of Forecasting Techniques

As we have said, at each reorganization point, we forecast a number of characteristics of the system for the interval up to the next reorganizational point. Specifically, we predict the following:

(1) the average size of the relation (number of tuples and number of pages);

(2) the average selectivity of each domain;

(3) the expected cost of maintaining an index for each domain;

(4) the expected storage requirement of an index for each domain;

(5) the expected cost of each use of an index in obtaining the TIDs of those tuples that satisfy an equality condition involving the indexed domain;

(6) the number of occurrences of each query type.

We could do these projections solely on the basis of statistics collected during the latest time period, or we could combine together the statistics collected over all previous periods. However, neither alone would be satisfactory for the purpose of a stable and yet responsive adaptive system. In the former case, the system would be overly vulnerable to chance fluctuations, whereas in the latter case, it would be too insensitive to real changes. A more satisfactory approach would be to take into consideration the pattern of change in each of these parameters in earlier time intervals in arriving at predictions for their values in the upcoming interval. A broad spectrum of techniques is available for the analysis and forecasting of time series. However, because of the potentially large number of parameters in our cost model, we have to restrict ourselves to those forecasting techniques that are efficient in terms of computation and storage requirement. Specifically, we consider here the technique of exponential smoothing for our forecasting procedure because of its simplicity of computation, its minimal storage requirement, its adjustibility for responsiveness and its generalizability to account for trends. In the following discussion, we will refer to the t th observation of a time series (i.e., values of a parameter over successive periods of time) as x(t) and the next forecast based on observations up to x(t) as $\hat{x}(t)$.

3. Exponential Smoothing

Intuitively, a weighted moving average strikes a reasonable balance between the two

extremes for parameter prediction mentioned earlier. Forecasts derived by weighing past observations exponentially (or geometrically) have been used with some success in operations research and economics [Brown59, Muth60, Winters60, Brown62]. The forecast is based on two sources of evidence, the most recent observation and the forecast made one period before. The exponential smoothing procedure, in its simplest form, is carried out as follows:

$$\hat{x}(1) = x(1) \tag{4.3}$$

$$\hat{x}(k) = \alpha\, x(k) + (1 - \alpha)\, \hat{x}(k - 1) \tag{4.4}$$

where α is called a smoothing constant and takes on values between 0 and 1. A closed form expression for $\hat{x}(k)$ is

$$\hat{x}(k) = x(1)\ (1 - \alpha)^{k-1} + \alpha \sum_{i=0}^{k-2} x(k - i)\ (1 - \alpha)^{i} \tag{4.5}$$

In essence, the new forecast is calculated as a weighted average of all previous observations with the weight decreasing geometrically over successively earlier observations. The compactness of the scheme lies in the fact that only two parameters need to be maintained for each time series: the current observation and the previous estimate. Note that equation (4.4) can be rewritten as follows:

$$\hat{x}(k) = \hat{x}(k - 1) + \alpha\, (x(k) - \hat{x}(k - 1)) \tag{4.6}$$

We see that the new forecast is equal to the sum of the two terms: the old estimate and a correction term that is proportional to the previous forecasting error (difference between

forecast and actual observation). The rate of response to recent changes can be adjusted simply by changing the smoothing constant: the larger the smoothing constant, the more sensitive is the forecast to recent changes and chance fluctuations. Since the weights given to earlier observations sum up to one, no systematic bias is introduced (i.e., the expected value of the forecast is equal to the expected value of the random variable). Hence, this procedure is appropriate only for the forecasting of the expected values of stochastic variables whose sums do not change between successive periods [Denning71]. If there is a long term upward or downward trend in the series, the forecast will always lag behind or lead the actual observation. Since we expect to observe trends in various activities in the data base, it is appropriate to choose a forecasting technique that can accomodate trends.

4. Adaptive Forecasting

This is a variant of the simple exponential smoothing technique that takes trend into consideration. Its form is [Theil64]

$$\hat{x}(t) = \bar{x}(t) + e(t) \tag{4.7}$$

$$\bar{x}(t) = \alpha\, x(t) + (1 - \alpha)\,(\bar{x}(t - 1) + e(t - 1)) \tag{4.8}$$

$$e(t) = \beta\,(\bar{x}(t) - \bar{x}(t - 1)) + (1 - \beta)\, e(t - 1) \tag{4.9}$$

where $\bar{x}(t)$ and $e(t)$ are the trend and the trend change at time t respectively. (Either an additive or a multiplicative trend can be incorporated; the latter through logarithmic transformation of the original series.) To carry out a forecast, we need only the current

observation, the previously computed values for the trend, and the trend change, and the computation is still very simple.

The appropriate choices for the smoothing constants α and β, however, is a non-trivial problem. It is possible to take a completely empirical approach [Winters60]. By maintaining the previous values for the time series, it is possible to compare the forecasts made using different sets of the parameters α and β (One reasonable criterion for comparison may be the standard deviation of forecasting error). Winters [Winters60] has suggested the method of steepest descent [Beckenbach56] for finding the best parameters for a single series. This method, however, consumes sufficient storage space and computing time to make its application to the large number of series in our system feasible. On the other hand, we have no reason to believe that a set of parameters that work best for a particular series will work equally well for other series, so that it might not be too practical to choose the optimum weights for one series and use the same weights for all other series. Theil and Wage [Theil64] have formulated an explicit stochastic model as a basis for the above forecasting method (equations (4.7) through (4.10)). The time series is postulated to be generated by

$$x(t) = \xi(t) + \mu(t) \tag{4.10}$$

$$\xi(t) = \xi(t-1) + \psi(t) \tag{4.11}$$

where $\xi(t)$ is the mean of $x(t)$ and $\psi(t)$ is the trend change from period $t-1$ to period t. (We can interpret $\bar{x}(t)$ and $e(t)$ of equations (4.8) and (4.9) as estimators of $\xi(t)$ and $\psi(t)$ respectively.) The trend change is postulated to be generated by

(4.12) $\eta(t) = \eta(t-1) + \nu(t)$

where $\mu(t)$ and $\nu(t)$ are time series with zero mean, constant variance ($\sigma^2(\mu)$ and $\sigma^2(\nu)$ respectively) and zero covariance of all kinds.

For this underlying model, Theil and Wage [Theil60] have found the optimal weights α and β to be used. Let

(4.13) $g^2 = \sigma^2(\nu)/\sigma^2(\mu)$

(4.14) $h^2 = -g^2/8 + g(1 + g^2/16)^{1/2}/2$

Then the optimal weights for α and β are

(4.15) $\alpha = 2h/(1 + h)$

(4.16) $\beta = h$

The mean square error of the forecasts is dependent on the choices for α and β which in turn are dependent on the estimate for the variance ratio g^2 (ratio between the estimates for $\sigma^2(\nu)$ and $\sigma^2(\mu)$) A sensitivity analysis of the consequences of error in estimating the variances ratio (g^2) in [Theil60] has shown that a 50% error results in less that 1.5% increase in the mean square prediction error. We can therefore start with a rough estimate of g^2, determine the appropriate values for α and β (or equivalently, we can start with an arbitrary choice for α and β), and adapt these coefficients to updated estimates of the variance ratio.

Our application of the above technique in the forecasting of a time series in our system can be summarized as follows.

At initialization, let

(4.17) $\bar{X}(0) = x(0)$

(4.18) $e(0) = 0$

At $t = 1$,

(4.19) $\bar{X}(1) = \alpha\, x(1) + (1 - \alpha)\, \bar{X}(0)$

(4.20) $e(1) = \beta\, (\bar{X}(1) - \bar{X}(0))$

At $t = 2$,

(4.21) $\bar{X}(2) = \alpha\, x(2) + (1 - \alpha)\, (\bar{X}(1) + e(1))$

(4.22) $e(2) = \beta\, (\bar{X}(2) - \bar{X}(1)) + (1 - \beta)\, e(1)$

(4.23) $\sigma^2(\mu, 2) = (\bar{X}(1) + e(1) - x(2))^2$

(4.24) $\sigma^2(\nu, 2) = (e(2) - e(1))^2$

($\sigma^2(\mu, t)$ and $\sigma^2(\nu, t)$ are estimates of the variance $\sigma^2(\mu)$ and $\sigma^2(\nu)$ at time t)

For $t > 2$,

$$(4.25) \quad \bar{x}(t) = \alpha\, x(t) + (1 - \alpha)\, (\bar{x}(t - 1) + e(t - 1))$$

$$(4.26) \quad e(t) = \beta\, (\bar{x}(t) - \bar{x}(t - 1)) + (1 - \beta)\, e(t - 1))$$

$$(4.27) \quad \sigma^2(\mu, t) = ((t - 3)\, \sigma^2(\mu, t - 1) + (x(t) - \bar{x}(t - 1) - e(t - 1))^2)/(t - 2)$$

$$(4.28) \quad \sigma^2(\nu, t) = ((t - 3)\, \sigma^2(\beta, t - 1) + (e(t) - e(t - 1))^2)/(t - 2)$$

We begin by using arbitrary values for α and β. As new estimate for the variance ratio g^2 becomes available (from the ratio of $\sigma^2(\nu, t)$ to $\sigma^2(\mu, t)$) , we can adapt the values for α and β. (Note that the amount of information that has to be passed on from one interval to the next is still quite small, and the computation needed to choose the appropriate weights is minimal.)

5. Parameter Forecasting

Using the foregoing techniques, we can summarize our forecasting procedures for upcoming parameters as follows:

(1) average size of the relation - we can use the current size of the relation as our current

observation, forecast the size of the relation at the end of the upcoming interval, and use the average of the two as the average size of the relation over the upcoming interval.

(2) maintenance cost of each domain index - if the domain is indexed in the previous interval, then its actual maintenance cost can be used as the latest observation; otherwise we can use the estimated cost as described earlier as the latest evidence.

(3) number of occurrences of each query type - if an observed query type has no previous forecast, then we will use the observed frequency as the next forecast and treat this as a new series to be forecasted.

(4) average selectivity of each domain, storage requirement and average accessing cost of each domain index - we note that our estimates for the current values of these parameters in the case of non-indexed domains are rather crude, hence we will reinitialize the forecasting procedure for each "newly" indexed domain, i.e., if a domain is indexed in the most recent interval but not in the one before, then we will use the most recent observation as the sole evidence in the forecasting of these parameters.

CHAPTER 5
INDEX SELECTION

A straightforward approach to the index selection problem would be to evaluate the projected total system cost for each possible index set (using equation (3.16)), and then select that set of domains which gives the smallest cost. With m domains in the relation, there are 2^m possible choices of index sets. For small m (say less than 10), this enumerative approach may be feasible for finding the optimal combination of domains to be indexed. However, because of the exponential rate of increase of the number of possible index sets with the number of domains, the search space becomes prohibitively large very rapidly. (With 30 domains, there are more than 10^9 index sets to be considered. The cost of exhaustively exploring the search space may no longer commensurate with the profits that can be gained.) Yet, it is not uncommon to find single-relation data bases with tens of domains. Therefore, it is appropriate to look for ways whereby the search space of potential index sets can be systematically reduced. One possible approach is to look for properties of the cost function that will allow it to be minimized without exhaustive enumeration, such as through a depth-first search, as exemplified in Schkolnick's index selection study [Schkolnick75]. However, these properties depend upon unrealistic assumptions that domain occurrences are uncorrelated and that the tuple acess cost function is linear; and even so, the associated upper bound of $2^{m^{0.5 \log m}}$ index sets to be tested is not enough of a reduction to enable the inexpensive selection of the optimal index set for a relation with a moderate number of domains.

When we remove the above two assumptions, the computation needed to evaluate the utility of a proposed index set becomes dependent on the number of distinct query types

forecasted. (All told, there are $(2^m - 1)$ possible conjunctive query types (which specify 1 or more domains) and $(2^m - m - 1)$ disjunctive query types (which specify 2 or more domains for a total of $(2^{m+1} - m - 2)$ possible query types.) Except in cases when only a few of the large number of potential query types actually occur, the evaluation of the cost-effectiveness of a particular potential index set is quite expensive. Hence, we have a strong incentive for systematically reducing the search space for the optimal index set. Yet, because of our lack of simplifying assumptions, the hope of finding an algorithmic way to explore a reduced search space of practical size and still finding the optimum is dim. Therefore, it is appropriate to draw on the experience of artificial intelligence researchers working in areas where formal mathematical structures are computationally impractical, and use heuristic methods [Feigenbaum63, Meier69] that significantly prune down the search space and that work towards obtaining a near-optimal solution.

1. Index Selection Heuristics

In this section, we examine the structure of the index selection problem and describe a number of ways in which the index selection cost can be reduced.

(1) Not all queries can use indices profitably. The expected set of tuples that satisfy a query may be so large (i.e. the qualified tuples are likely to reside on close to p pages) that no set of indices could possibly be useful in its processing. Since the cost of computing the utility of a proposed set of indices is dependent on the total number of queries under consideration, those queries that cannot profitably make use of indices should be removed from the projected query set whose processing cost is to be minimized. This can be done by computing the processing cost of each query given

the availability of indices on all domains that are used in the query. If this is more expensive than a sequential scan, then the query should be removed from the projected set of queries.

(2) Some domains can be eliminated from the initial candidate set by virtue of their low occurrence frequencies in queries. This effectively reduces m, the initial number of domains in the candidate set. Using the forecasted frequency of each query type, and the projected selectivity of each domain in the relation, we can compute an upper bound on the number of page accesses that the use of an index on a particular domain can save in the processing of the forecasted set of queries. If this upper bound is less than the projected cost of maintaining an index on the domain, then this domain can safely be excluded from the initial candidate set, i.e., the domain is so unselective or is used in retrievals so infrequently relative to its being updated, that it cannot possibly be profitable to index it.

The upper bound on the utility of an index for an arbitrary domain i is computed as follows. Let q be a conjunctive query type that involves domain i, and let S_q be the joint selectivity of all the domains of q. Then the tuples that satisfy q are expected to reside on $f(S_q n)$ pages, where n is the total number of tuples in the relation and f is our non-linear function for expected page accesses. So an upper bound on the benefit that an index on i could possibly bring to the evaluation of q would be to reduce the number of pages to be accessed from p to $f(S_q n)$. (A similar formula holds as well for the maximal reduction where q is a disjunctive query, but with S_q there representing the joint disjunctive selectivity of the domains used in q.) In the case of a conjunctive query, an additional upper can be computed which in some cases may be tighter than

the one just mentioned. Note that an index on domain i with selectivity S_i reduces the number of tuples that have to be examined to resolve a conjunctive query involving domain i by a factor of S_i. However, because of the convexity of our tuple access cost function, a reduction by S_i in the number of tuples to access leads to less than a reduction by S_i in page accesses. Hence the maximal incremental saving (in terms of page accesses) cannot exceed $p*(1 - S_i)$. Thus the upper bound on the utility of an index for i is:

$$(5.1)\quad \Sigma_{q \in Q_i} F_q * ((1-T_q) * \min(p*(1-S_i), p*(1-f((\Pi_{j \in D_q} S_j)*n)) + T_q * p*(1-f((1-\Pi_{j \in D_q}(1-S_j))*n))$$

where Q_i = set of forecasted query types that use domain i

F_q = projected number of occurrences of q

D_q = set of domains referenced in q

T_q = 0 if q is conjunctive and 1 if q is disjunctive

n = total number of tuples in the relation

f = non-linear tuple access cost function for the relation

(3) Some domains could be known to be included in the optimal index set by virtue of their high occurrence frequencies in queries. For each domain, we can compute a lower bound on the savings in query processing its indexing can bring. If the latter is less than the expected maintenance cost, then the domain must be included in the optimal index set. In cases where a domain is used together with others in a query, it is very difficult to assess the lower bound on the utility of the index. Therefore, we will compute the lower bound for a domain based only on those queries in which the

domain occur alone.

(4) A near optimum choice of the index set can be made incrementally. This heuristic permits analysis of the problem as a stepwise minimization, each time adding to the index set that domain which will bring the best improvement to the cost function.

There have been two previous suggestions regarding the incremental selection of domains to be indexed. Farley and Schuster [Farley75] suggest that the incremental selection process can be terminated once no single domain in the non-indexed set can be chosen that will yield incremental cost/benefits. This is insufficient for our choice of query and tuple access models; there are two reasons why it may be necessary to consider the incremental savings brought by adding two or more indices together to the index set candidate. First, it may happen that for a query involving a conjunction of conditions, the selectivity of any one domain may not be sufficient to reduce the number of pages to be accessed to significantly less than the total number of pages in the relation, whereas the joint selectivity of two or more domains might. (Recall that the reduction must be significant in order to cover the index accessing cost.) Secondly, a disjunction of conditions can be resolved via indices only if all of the domains involved in the disjunction are indexed. An alternative strategy has been suggested by Held [Held75b], who, at any stage of the incremental index selection procedure, considers the incremental savings of each of the possible subset of domains in the candidate set with less than or equal to some fixed number of domains in it. This, of course, may be very inefficient. We have taken an intermediate approach. We consider the adjoining of multiple domains to the index set only if no single domain that will yield positive incremental savings can be found.

(5) Only a small subset of all possible candidate domains need be considered in determining the next domain or set of domains to be adjoined to the index set at each stage. We can rank the domains with respect to the maximal savings each can bring and then consider only the top ranking M domains, and combinations of them, for detailed incremental savings calculation. (Alternatively, we can take the maintenance cost of each into consideration and divide the maximal savings of each index by its maintenance cost before doing the ranking.) Furthermore, a bound M' (M'≤M) can be imposed on the number of domains that will be considered together.

(6) An upper bound can be put on the total number of cost evaluations (i.e., the total number of index sets considered) to be performed in the entire selection procedure. Also, an upper bound can be put on the maximum size of an index set that will be considered. The incremental selection procedure will be terminated when either of these bounds is exceeded.

2. Index Selection Procedure

To illustrate the above heuristics, we present the details of our index selection procedure. Our procedure can be divided into three phases.

Phase 1 (Initialization) - During this phase, a tentative index set is chosen to include all those clearly profitable domains, and a ranking of the domains that might be profitable to adjoin to the tentative index set is computed. This involves the following steps:

(a) Remove from the projected set of queries all those that cannot profitably make use of

indices.

(b) For each domain, compute a lower and upper bound on the savings an index on the domain can bring.

(c) Partition the set of domains D in the relation into three disjoint subsets: D_t - the tentatively chosen index set, D_c - the candidate set, and D_n - the non-profitable set. Initialize D_t with those domains whose maintenance costs are less than the corresponding minimal savings they can bring, D_n with those domains whose maintenance cost exceeds the corresponding maximal savings they can bring, and D_c with $D - D_t - D_n$.

(d) Rank the domains in D_c with respect to their estimated utility.

Phase 2 (Incremental Selection) - The tentative index set is enlarged by the adjoinment of domain(s) to it incrementally.

(a) Consider in turn the incremental savings gained by indexing each of the M top ranking domains in the candidate set (i.e., for each of these domains d, compute the cost associated with $D_t + d$, and compare it to the cost associated with D_t). Adjoin to D_t that one which will give the best improvement to the cost function. If one cannot be found, then consider larger-sized combinations (up to M') of these M domains. Consider combinations of the next larger size only if it is not profitable to adjoin any of the combinations of the current size.

(b) Remove the domain(s) from D_c as they are adjoined to D_t. Resume considering individual domains for further adjoinment after an adjoinment to D_t.

(c) Terminate the incremental selection if no subset (of size less than or equal to M') of the M top ranking domains in the candidate set can be chosen such that its adjoinment to the index set will improve the index set's cost function; or if the upper bound on the total number of cost evaluations is reached.

Phase 3 (Bump-shift [Kuehn63]) - Domains that have been adjoined to the tentative index set early in the incremental selection phase may turn out to be uneconomical as the result of later addition of other domains to the set, and thus should be removed from the index set. Since the probability of the need for the simultaneous removal of more than one domain is quite small, we will only consider the removal of individual domains. (The necessity to remove two domains from D_t implies that some of those queries whose processing costs are significantly improved by the initial adjoinment of these domains to D_t, become less dependent on them as other indices become available. The fact that it is not profitable to remove one of them alone implies that there are some queries which depend on both of them, and whose processing costs are improved, in the presence of both indices, by more than the maintenance cost of either one, but less than the maintenance cost of both. Such a combination of circumstances is rare enough for us to ignore it.)

(a) For each domain d tentatively assigned to the index set, subtract the total cost for D_t from that for D_t - d. Remove from D_t that d for which the above difference is largest.

(b) Repeat the process until no domains remain in D_t whose removal would improve its

cost function.

In order to assure that we have a real local optimum, we may go back to the incremental selection phase after some domains have been removed from the tentative index set. To guarantee that the process terminates, we would put a domain d into D_n if it is removed from D_t by the bump-shift phase.

3. Performance of Index Selection Heuristics

We have discussed a number of ways in which the index selection problem may be simplified. The initialization phase of our heuristic selection procedure leads to a reduction in the search space for the optimal index set and a reduction in the total number of queries that have to be considered under any proposed indexing policy, without jeopardizing the possibility of finding the real optimum index set. On the other hand, when we make use of the heuristics of stepwise minimization and of considering only the top ranking domains for incremental selection, we have opted for a good solution at reasonable cost rather than the optimal solution at any cost. There are several reasons why the stepwise minization procedure should be good.

(a) It resembles the methods that might be employed by an intelligent human being in solving the index selection problem. For any tentatively chosen index set, we know for sure that the cost of maintaining those indices is less than the savings that they bring. Furthermore, the total system cost is monotonically decreasing as successive domains are added to (during incremental selection) and removed from (during bump-shift) the tentative index set.

(b) It has been successfully applied in problem areas of a similar nature. In [Kuehn63], stepwise minimization was applied to the problem of choosing the sites for warehouses from a number of potential sites which minimize a particular cost function. The problem is in many respects similar to the index selection, especially in the fact that for each potential warehouse site, there is the possibility of having or not having a warehouse at that site, just as for each domain, we have the possibility of having or not having an index on that domain.

(c) It actually finds the optimal index set under certain circumstances. It is provably optimal if only single domain queries and/or disjunctive queries are present in the projected query set. In such a case, it is impossible for the heuristic algorithm to choose an index set that is a subset of the optimum, since it considers adjoining combinations when necessary; also, it is impossible for the heuristic algorithm to include in its choice a domain that is not in the real optimum index set. (The fact that a domain has been adjoined to the heuristically chosen index set means that there is a set of queries which depend on the availability of the index in question in order to be resolved using indices, and that the savings from processing these queries using indices more than pay for the maintenance cost of that index.)

In the presence of both conjunctive and disjunctive queries, it is possible that the heuristically chosen index set can depart significantly from the optimal index set. However, we can argue that the probability of this occurring is quite small, and even if it this does occur, the total system cost under the heuristically chosen set of indices may not be too different from that under the optimal index set.

Let D_{opt} and D_{heur} be the optimum index set and the set chosen by the heuristic index selection procedure respectively; then we have the following circumstances in which D_{heur} will be non-optimal.

(a) D_{opt} strictly includes D_{heur} - This is highly unlikely, since we do consider the adjoinment of multiple domains (up to a certain bound) to the tentatively chosen index set if no simpler adjoinment is profitable.

(b) D_{heur} strictly includes D_{opt} - Because of our bump-shift procedure, we know that D_{heur} must include two or more domains that are not in D_{opt}, and as discussed in the previous section (on the bump-shift procedure), this is very unlikely.

(c) There are domains in D_{opt} which are not in D_{heur} and vice versa - This is probably going to be the most common. The fact that domains which are in $D_{opt} - D_{heur}$ are not adjoined to D_{heur} implies either that they need to be simultaneously indexed to be useful and their total number exceeds the bound on the number of domains that the heuristic index selection procedure will consider for simultaneous adjoinment, a possibility which is quite remote; or that indices on them are no longer useful in the presence of domains in $D_{heur} - D_{opt}$, in which case the total system cost for D_{heur} may not be too far away from that for D_{opt}.

We have performed a limited amount of experimentation with the above heuristic algorithm, applying it to a number of access histories, and comparing its results to those obtained by an exhaustive consideration of all possible index sets. In the cases that we have tested, the heuristic algorithm has almost always found the optimal index set at a small

fraction of the cost of the exhaustive procedure. Most of the increments to the tentative index set consist of single domains, so that the total number of index sets considered only increased only linearly, instead of exponentially, with the total number of domains in the relation. Moreover, the bump-shift phase seldom yielded an improved choice over that given by the incremental selection, which in many cases was already identical to the choice given by an exhaustive search and therefore optimal.

4. On Further Reducing the Cost of Index Selection

The main thrust of the heuristic index selection algorithm described in the previous section was towards reducing the search space for potential index sets by making the selection procedure an incremental one. However, in addition to the need for cutting down the index set search space, there is also the need to minimize the cost of assessing the cost/benefits of each individual index set. By making forecasts of query type occurrence frequencies based on past observation, we have thus far avoided the strong assumption that individual domain occurrence probabilities in a query are independent. In consequence, however, our scheme requires that in considering each possible increment to the index set, we evaluate the costs of processing each of the projected query types that involves any of the domains in the increment. The number of possible query types is an exponential function of the number of domains in the relation; so the number of query types that actually occur is also likely to increase quite rapidly with the number of domains. There may be as many as $2^m - 2^{m-k}$ conjunctive query types that will require individual computation (for new processing cost), where k is the size of the increment under consideration; these are those queries that use at least one of the domains in the proposed increment. One possible simplification is to group queries together and to characterize the group in terms of a small number of statistical

properties. Instead of finding the savings in the processing of each of the queries that are affected by a proposed increment, we can compute the savings for the group as a whole, which can be done more efficiently. In the following sections, we will examine one query grouping scheme that has been suggested previously and suggest extensions to it.

In [Schkolnick75] (who considers only conjunctive queries), all queries are put into a single group which is described by the query occurrence probabilities of each domain, (i.e., the fraction of queries in which the domain is used). Furthermore, these probabilities are assumed to be independent. For example, for a relation with three domains a, b and c, each with occurrence probability P_a, P_b, P_c respectively, the probability of having a query that involves just the domains a and b is assumed to be

(5. 2) $$P_a * P_b * (1 - P_c)$$

since P_a and P_b are the occurrence probabilities of domains a and b, and $1 - P_c$ is the probability of domain c's non-occurrence. For a proposed index set D, the total query processing cost can then be computed as follows.

Let
- N_t = total number of tuples
- N_Q = total number of queries
- Q = set of all possible queries
- AS_i = average selectivity of domain i
- AC_i = average access cost for index on domain i
- P_q = occurrence probability of query q, $q \in Q$
- D_q = domains specified in query q, $q \in Q$

F_q = cost of accessing the set of tuples to resolve q using index set D

= cost of accessing $(\Pi_{i \in D_q \cap D} AS_i) * N_t$ tuples

C_q = processing cost of query q with index set D

= $(\Sigma_{i \in D_q \cap D} AC_i) + F_q$

then the total query processing cost is

(5. 3) $$N_Q * \Sigma_{q \in Q} P_q * C_q$$

With m domains in the relation, there are 2^m possible queries. However, in evaluating the utility of an index set of size s, only 2^s distinguishable sub-groups of queries need to be considered. (A distinguishable sub-group of queries consists of all those queries with the same expected processing cost under a given set of indices. Given an index set D, two queries fall into the same sub-group if they use the same set of domains in D, since their processing will involve the use of the same set of indices, resulting in the accessing of sets of tuples of the same expected size.) Consider the above 9-domain relation and the index set which includes only domain a, then the possible queries in the group can be divided into two sub-groups, those that specify domain a and those don't. In general, it is necessary to evaluate the processing cost of each of these distinguishable sub-groups individually before an expected processing cost for an average query can be computed. However, by assuming that the tuple access cost function is linear, a further simplication results in the following total query processing cost

(5. 4) $$N_q * ((\Sigma_{i \in D} P_i) * AC_i + \Pi_{i \in D} (1 - P_i + P_i * AS_i))$$

The above formula admits of the following simple interpretation: for an average query, with probability P_i domain i in the index set D is specified, in which case the fraction of tuples that have to be examined is reduced by AS_i; and with probability $1 - P_i$ domain i is not specified in which case an index on domain i does not lead to any reduction in the number of tuples to be examined.

We thus see that Schkolnick's scheme leads to a very simple computation for the evaluation of the utility of an index set. However, the simplifying assumptions that lead to this computational simplicity are not altogether realistic.

5. Query Clustering

We feel that the idea of grouping queries is fundamentally sound, since it significantly reduces the number of query types that have to be considered at each step of the incremental index selection procedure. On the other hand, grouping can lead to loss in correlation information. For example, again consider the above 3-domain relation: it may happen that domains a and b never appear together in queries, whereas the independence assumption will lead us to assume that a query that specifies only domain a and domain b does occur with probability $P_a * P_b * (1 - P_c)$. In order to preserve the correlation information, we should only group similar queries together. Hence, the division of the queries into more than one group may be necessary. Since some correlation information is inevitably lost when queries are grouped together and it often happens that some queries occur quite frequently while others only rarely, we may want to consider the most frequently occurring queries individually, in the process of incremental indexing utility calculation, while grouping the less frequent ones into one or more groups.

To incorporate the above scheme, the evaluation of the utility of any proposed increment to the tentative index set can be modified as follows. The incremental savings afforded by the increment to each of the frequent (non-grouped) queries is computed as before. As for each of the query groups, we compute the improvement to each distinguishable sub-group of queries that is affected by the increment. The improvement to the group is then computed as a product of the total number of query occurences in the group and the average improvement to a query in the group. The latter is obtained from the sum of the improvements to each of the distinguishable subgroups, weighed by their individual occurrence probability with respect to the group.

The clustering scheme we suggest for the less frequent queries is of the "nearest-centroid" type [Belford75]. (This involves the definition of a metric or a measure for the distance between queries and groups of queries. The centroid of a group may be looked upon as an average (or representative) query in the group.) Since the cost evaluation process at each step of the incremental index selection procedure is dependent on the number of query groups we have, we may a priori determine the number of groups (say G) into which the less frequent queries are to be divided. A possible clustering strategy is as follows. We rank the less frequent queries in terms of their occurrence frequencies, and start off with groups that are singletons of the G top ranking queries. The remaining queries are considered sequentially; each is added to the group with the nearest centroid, after which the centroid for the affected group is recomputed.

For each query group, we maintain its centroid and the total number of query occurrences in the group. We represent a query by means of a binary vector which indicates the domains that are used in the query and the centroid of a group of queries by means of a vector that

indicates the occurrence probability of the individual domains with respect to the group.

Let V_g = vector representation of a query group g

V_q = vector representation of a query q

F_g = total number of query occurrences in g

F_q = total number of occurrences of q

The distance between q and g can be computed as

(5. 5) $$||V_q - V_g|| = \Sigma_k |V_{qk} - V_{gk}|$$

When q is added to g, the centroid of the group is recomputed as

(5. 6) $$V_g \leftarrow (F_q * V_q + F_g * F_g)/(F_q + F_g)$$

and the total number of query occurrences in the group is updated as

(5. 7) $$F_g \leftarrow F_q + F_g$$

In order to evaluate the utility of a proposed index set with respect to a given group of queries, we need to have a scheme for the assignment of occurrence probability to each possible query in the group. One possibility is to use the independence assumption discussed previously. However, this results in the assignment of a non-zero probability to the query that specifies none of the domains, which is inadequate since we never include the query that specifies no domain in our grouping scheme. Therefore, we need to have a

scheme for the normalization of probability assignments. In addition, we might want to take into consideration the complexity (number of domains specified) of the component queries of the group. For example, if all of the component queries in the group involve say two domains, then we should discount single-domain or more-than-two-domain queries in the probability assignments. In view of the above two considerations, we can keep track of the number of query occurrences for each complexity in the process of adding queries to a group, and use the following normalization scheme.

Let N_Q = total number of query occurrences in the group

k = total number of domains with non-zero occurrence probability

P_i = occurrence probability of the i^{th} domain

NC_i = number of query occurrences with complexity i

The conditional occurrence probability of a query q, which uses domains in D_q, given that the query is of complexity C_q, can be computed as the product of the occurrence probabilities of domains in D_q, normalized by the sum of products of probabilities of all non-zero-occurrence-probability domains in the group, taken C_q at a time. The above normalization factor for queries of complexity i can be shown to be the coefficient of x^i in the following expansion [Liu68]:

$$\Pi_{1 \le i \le k} \ (1 + P_i x)$$

Hence, the unconditional proability of having a query q which uses the set of domains in D_q and of complexity C_q can be computed as

(5.8) $$(NC_{C_q}/N_Q) * ((_{i \epsilon D_q} P_i)/NF_{C_q})$$

Note that the number of distinguishable sub-groups in a query group with respect to an index set (and hence the cost of indexing utility evaluation) depends on the number of domains with non-zero occurrence probability (with respect to the the group) that the index set contains. (The adjoinment of domains with zero occurrence probability in the group to the index set will not affect the processing of the group.) Therefore, an alternative to the above strategy of a priori deciding the number of groups to have is to limit the number of domains with non-zero occurrence probability in each group. In attempting to add a query to one of the existing groups, we can take into consideration both its distance from the group and the number of domains with non-zero occurrence probability in the resulting group, and create a new group if necessary.

CHAPTER 6
SUMMARY AND FUTURE RESEARCH

The research reported in this thesis has been motivated by the need for intelligent data base management systems to support large integrated data bases. We have proposed a methodology for the incorporation of optimization and self-organization capabilities into data base management system. Specifically, we suggest the following approach:

(1) the development of an accurate cost model that closely reflects the data base environment and data base system operation (this cost model is to be used both by the query processor for selecting the most economic access path for a given query and by the reorganization component of the system for the selection of a near-optimal physical data base organization for the observed access pattern);

(2) the monitoring of accesses to the data base that allows the system to build up an accurate model of the contents of the data base and the way that the data base is used;

(3) the application of forecasting techniques to detect and respond to changes in access requirements and data characteristics;

(4) the design of computationally feasible heuristics that select a near-optimal physical organization at a reasonable cost.

We have applied the foregoing steps to the index selection problem and have achieved a design for the incorporation of an adaptive index selection capability into a dynamic, single-

relation data base environment. In the following sections, we will summarize the novel aspects of our approach and suggest possible extensions to it.

1. Comparison with Previous Work

Our experimental and heuristic approach to the index selection problem is different in many respects from previous studies by Stonebraker [Stonebraker74], King [King74], Schkolnick [Schkolnick75], Farley [Farley75], and Held [Held75b]. These other studies have either been formal analyses, which have made many simplifying assumptions in order to obtain an analytic solution, or else system designs that have been incomplete or unrealistic in various ways.

Our work attempts to go farther than these by utilizing more complete and accurate models of cost and access, and by emphasizing important aspects of realistic data base environments. Our model of tuple access is realistic in the sense that we take into consideration the blocking effect of tuples on secondary storage devices. Our cost models for data base access and maintenance account for such real overheads as the expense of index accessing and the cost of maintaining the index as a balanced tree. Our approach of minimizing the total processing cost for the upcoming interval, rather than the expected cost for a single data base transaction, is flexible enough to account for the overhead costs of index creation, index storage, and application program retranslation.

We have stressed the importance of accurate usage model acquisition and data characteristic estimation in a dynamic environment where access requirements are continually changing. Our scheme endeavours to obtain a precise model of data base usage by recording actual

query patterns, thereby avoiding the strong and often inaccurate assumption that domain specifications in queries are uncorrelated. We also take into consideration the facts that values of a domain may not be equally used in queries and that they may not be evenly distributed among tuples of the relation, by monitoring the actual selectivities of the domain values that are used in queries. On the other hand, we have also made sure that our schemes for gathering statistics during the processing of data base transactions have as little effect on system performance as possible.

We believe it necessary to apply forecasting techniques to past observations and predict future access requirements and characteristics, in order to capture and respond to the dynamic and changing nature of data base usage. In the selection of applicable forecasting techniques, we have stressed the importance of minimal storage requirements, simplicity in computation, responsiveness and adaptability.

Finally, the size of actual data bases is reflected in our concern for efficient heuristics to speed up the index selection process. Our scheme for the grouping of queries allows us to reduce the index selection cost and yet preserve the influence of domain correlation on the selection procedure.

2. Directions for Future Research

There are numerous optimization opportunities in a complex data base environment. In this thesis, we have addressed the optimization issues related to the choice of indices to be maintained and the strategy for using these indices in query processing. By way of conclusions, we suggest several directions in which our work can be extended.

(1) There are many separate issues that need resolution in the selection of physical organization for a general integrated data base, including method of placement of records on secondary storage, primary access mechanism, auxiliary access aids, clustering parameters etc. Within a single-data base environment, an organizational issue that might be profitable to consider in conjunction with the selection of indices is the division of the stored representation of the relation into a number of subfiles, each consisting of subtuples containing only a subset of the fields in the relation. The purpose of such an organization is to limit the amount of irrelevant information that is accessed, when the qualification and output parts of a query involve only a small number of domains in the relation. Previous studies [Kennedy72, Stocker73, Hoffer75] have considered this file partitioning problem in the absence of auxiliary access aids. An adaptive strategy towards the simultaneous selection of indices and file partitions might be fruitful.

(2) Even though our investigations into index selection are in many respects more comprehensive than previous studies, we have considered only the environment of a single-relation data base accessed through a restricted interface with limited capabilities for the selection of data. To fully realize the flexibility of a relational data base, it is necessary to consider a multi-relation environment together with a high-level non-procedural language interface that permits queries with arbitrary interconnection between relations in the qualification part and high level operations on the qualified data. In such an environment, it is necessary to consider the utility of indices for more complicated operations (such as restriction, projection, division, join, etc. [Codd70, Palermo72, Smith75, Rothnie75, Pecherer75, Wong76]) and to select indices for all the relations in the data base as a whole. The recording of detailed access history will be

necessary for optimal index selection in this environment, and the use of heuristics should be fruitful in cutting down the search space and for selecting richer index structures (such as combined indices).

(3) We have proposed that an intelligent data management system should build up a model of the contents of the data base and the way that it is used. Such information can be used for the evaluation of costs of alternative access paths for the processing of queries. In addition to individual query optimization and global choice of optimal physical organization, a query cost estimator can find yet another application in large integrated data bases. It is all too easy for a naive data base user to ask a simple-to-phrase query that will take a great deal of computational resource and time to answer. Frequently, the value of this information to the requestor will not be commensurate with the resources expended to obtain it. If a cost estimator is available at the user interface, a user can obtain an estimate of the cost of answering his query and then decide to pay the price and have it answered, or to cancel the query. More work on the development of cost models for complex query processing, and schemes for the acquisition of the necessary parameters, in order to provide such a facility.

(4) We have applied forecasting techniques to the prediction of upcoming access requirements and data characteristics. In a truly adaptive system, higher level adaptive mechanisms will also be necessary. Levin [Levin75] has suggested the following hierarchy of adaptive mechanisms to be employed in an uncertain environment:

(a) a forecasting mechanism that performs prediction of various parameters in the system based on past observations;

(b) a parameter adaptive mechanism that for a given forecasting technique chooses the best values for the basic parameters of the technique.

(c) a meta-adaptation mechanism that automatically switches from one forecasting technique to another based on their individual performance.

The adpative forecasting procedure we have described actually encompasses the first two mechanisms. To incorporate the meta-adaptive mechanism for a particular time series involves keeping around the entire series (or at least the most recent portion) and comparing the amount of forecasting error that would have been resulted from the application of each of the forecasting technique under consideration. The large number of parameters that we utilize preclude the application of any meta-adaptation mechanism to each of them. On the other hand, a selective application of such a mechanism to parameters to which the cost function is most sensitive may be appropriate.

We have assumed that reorganization is to be considered at fixed intervals, the length of which are to be determined by the data base administrator. Since the overhead of index selection is incurred at each reorganization point, it would be desirable to have the system automatically adjust the intervals between reorganization points to suit the rate of change in access pattern and the degradation of system performance. More fundamentally, an intelliggent adaptive system must assure that the payoff of the adaptive mechanisms is commensurate with its overhead costs, and "switch if off" when the usage requirements reaches a steady state.

APPENDIX 1

PROOF OF EQUATION (3.3)

Consider m tuples $T_1, T_2, \cdots, T_m$ to be placed into n equally likely slots that are partitioned into p blocks of t slots each (n = pt). Let p(r) be the number of blocks that contain $T_1, T_2, \cdots, T_r$. Define

(A1.1) $$p(0) = 0$$

(A1.2) $$d(r) = p(r+1) - p(r)$$

then p(r) and d(r) are random variables, and d(r) takes on values 0 or 1. Let f(r) be the expected value of p(r), then we have the following recurrence relation:

(A1.3) $$f(0) = 0$$

(A1.4)
$$\begin{aligned} f(r+1) - f(r) &= E(d(r)) \\ &= \text{Prob}\,[d(r)=1] \\ &= \sum_k \text{Prob}\,[d(r)=1 \mid p(r)=k]\ \text{Prob}\,[p(r)=k] \\ &= \sum_{k=0}^{k=p} \frac{n-k\,t}{n-r}\ \text{Prob}\,[p(r)=k] \\ &= \sum_{k=0}^{k=p} \left(\frac{n}{n-r} - \frac{k\,t}{n-r}\right) \text{Prob}\,[p(r)=k] \\ &= \frac{n}{n-r} - \frac{t}{n-r}\, f(r) \end{aligned}$$

(A1.5) $$f(r+1) = \frac{n-r-t}{n-r}\, f(r) + \frac{n}{n-r}$$

A closed form solution of the above difference equation can be obtained as follows. Let

(A1. 6) $$s = n - r$$

(A1. 7) $r = n - s$, then

(A1. 8) $$\sum_{s=0}^{\infty} s\, f(n-s+1)\, x^s$$

$$= \sum_{s=0}^{\infty} (s - t)\, f(n-s)\, x^s + \sum_{s=0}^{\infty} n\, x^s$$

With some manipulations of equation (A1.8) we have

(A1. 9) $$\sum_{s=0}^{\infty} (s - 1 + 1)\, f(n-(s-1))\, x^s$$

$$= \sum_{s=1}^{\infty} (s - 1 + 1)\, f(n-(s-1))\, x^s$$

$$= \sum_{s=0}^{\infty} s\, f(n-s)\, x^s - \sum_{s=0}^{\infty} t\, f(n-s)\, x^s + \sum_{s=0}^{\infty} n\, x^s$$

Considering the second equality sign in equation (A1.9), we get

(A1. 10) $$x^2 \sum_{s=1}^{\infty} (s - 1)\, f(n-(s-1))\, x^{s-2} + x \sum_{s=1}^{\infty} f(n-(s-1))\, x^{s-1}$$

$$+ x \sum_{s=1}^{\infty} f(n-(s-1))\, x^{s-1}$$

$$= x \sum_{s=0}^{\infty} s\, f(n-s)\, x^{s-1} - t \sum_{s=0}^{\infty} f(n-s)\, x^s + n \sum_{s=0}^{\infty} x^s$$

Let

$$F(x) = \sum_{s=0}^{\infty} f(n-s)\, x^s \tag{A1.11}$$

$$F'(x) = \sum_{s=0}^{\infty} s\, f(n-s)\, x^{s-1} \tag{A1.12}$$

then from equation (A1.10) we get

$$x^2 F'(x) + x F(x) = x F'(x) - t F(x) + \frac{x}{1-x}, \text{ or} \tag{A1.13}$$

$$F'(x) - \frac{t+x}{x(1-x)} F(x) = -\frac{n}{x(1-x)^2} \tag{A1.14}$$

Equation (A1.14) is a linear first order differential equation, and has the following general solution

$$F(x) = \frac{x^t}{(1-x)^{t+1}} \left(c - \int \frac{(1-x)^{t-1}}{x^{t+1}}\, dx \right) \tag{A1.15}$$

$$= \frac{x^t}{(1-x)^{t+1}} \left(c + \frac{n}{t} \left(\frac{1-x}{x} \right)^t \right)$$

$$= \frac{c\, x^t}{(1 - x)^{t+1}} + \frac{n}{t\,(1 - x)}$$

$$= c\, x^t \sum_{k=0}^{\infty} \binom{t+k}{k} x^k + \frac{n}{t} \sum_{s=0}^{\infty} x^s$$

$$= c \sum_{k=0}^{\infty} \binom{t+k}{k} x^{t+k} + \frac{n}{t} \sum_{s=0}^{\infty} x^s$$

From equations (A1.7) and (A1.8), $f(r) = f(n-s)$ = coefficient of x^s in $F(x)$. Letting

(A1.16) $$s = t + k$$

$$= n - r, \text{ we have}$$

(A1.17) $$F(x) = c \sum_{s=t}^{\infty} \binom{t+k}{t} x^s + \frac{n}{t} \sum_{s=0}^{\infty} x^s$$

(A1.18) $$f(r) = c \binom{n-r}{t} + \frac{n}{t}$$

Using the initial condition $f(0) = 0$, we have

(A1.19) $$c \binom{n}{t} + \frac{n}{t} = 0$$

(A1.20) $$c = -\frac{1}{\binom{n-1}{t-1}}$$

Substituting this for c in equation (A1.19) we have,

$$f(r) = \frac{n}{t} - \frac{\binom{n-r}{t}}{\binom{n-1}{t-1}} \qquad \text{(A1.21)}$$

$$= \frac{n}{t}\left(1 - \frac{\binom{n-r}{t}}{\binom{n}{t}}\right)$$

APPENDIX 2
ANALYSIS OF SORTING COST

The sorting of pairs of domain values and tuple identifiers forms a key step in the creation of an index. For typical file sizes in a data base environment, an external sorting is required. The sort merge technique has been extensively studied [*]. Ignoring internal comparison costs, the cost of a sort merge depends on the number of initial sorted subfiles, the merge factor, and the size of the blocks that are read from and written back into secondary storage. However, as we have assumed that the page is the fixed unit of storage allocation, we will ignore the possibility of improving the disk accessing cost by reading and writing blocks larger than one page each.

Consider the sorting of a file of p pages. Let b be the number of pages in main memory available for internal buffering. As a first step of the sorting process, s sorted subfiles of the original file can be formed using s internal sorts. To optimize the subsequent merging process, s should be minimized by maximizing the size of each of the sorted subfiles. Hence, the size of each sorted subfile should be made equal to the size of the internal buffer, i.e. b pages. The cost of this phase of the sort-merge is 2*p page accesses (since the sorting of each of the subfiles is done internally without incurring extra page accesses). It is possible that the original p pages of the file are only partially occupied, so that the writing out of the sorted subfiles will incur less than p page accesses. Let u be the occupancy factor (or fraction of storage utilization) of the original file; then the cost of forming the subfiles is p * (1 + u) since the total length of the sorted subfiles will only be p*u pages. It is also possible that pu is not a multiple of b, in which case s-1 subfiles of length b, and one with length b' (= p - b * (s - 1)) will be formed.

The merge phase consists of repeatedly merging sorted subfiles until a single one is obtained. Knuth [*] has shown that merge patterns can be represented as trees, and that the merging cost is proportional to the external path length of the corresponding tree. Therefore, sorting cost is minimized by choosing a tree with minimum external path length (sum of the level numbers of all the external nodes), such as a complete z-ary tree where z is as large as allowable by the internal buffer size. Allowing one page for the buffering of tuples from each subfile that participates in a merge, and one page for the output buffer, z will be chosen to be b-1. Given s initial sorted subfiles (of which the first s-1 are of length b, and the last one is of length b'), the algorithm for carrying out the merging according to a complete z-ary tree pattern can be described as follows. First add dummy subfiles (of zero length) as necessary to make s = 1(modulo (m-1)), to the front of the queue of initial subfiles, then combine subfiles according to a first-in-first-out discipline, at any stage merging the z oldest subfiles at the front of the queue into a single file which is placed at the rear. The merging process terminates when a single sorted file is left. The external path length L for a complete z-ary merge tree is [*]:

$$L = qs - \lfloor (z^q - s)/(z - 1) \rfloor \qquad \text{(A2.1)}$$

where $s = \lceil p/b \rceil$

$q = \lceil \log_z s \rceil$

Hence, the paging cost C_1 for the merging phase for the case that pu is a multiple of b is:

$$C_1 = b*L \qquad \text{(A2.2)}$$

If pu is not a multiple of b, there will be s - 1 subfiles of length b, and one with length b' (= p - b * (s - 1)). In this case, the merge-sort cost C_2 is:

(A2. 3) $$C_2 = C_1 - q * (b - b')$$

Hence, the merging cost C_{merge} (s - 1, b') for s - 1 subfiles of length b and one of length b' is

(A2. 4) $$\begin{cases} C_1 & \text{if } b = b' \\ C_2 & \text{if } b \neq b' \end{cases}$$

REFERENCES

[ASTRAHAN75]

Astrahan, M. M., Chamberlin, D. D., "Implementation of a Structured English Query Language", Proceedings of the ACM-SIGMOD International Conference on Management of Data, May, 1975.

[BAYER72]

Bayer, R., McCreight, E., "Organization and Maintenance of Large Ordered Indexes", Acta Informatica, Vol. 1, Fasc. 3, 1972.

[BECKENBACH56]

Beckenbach, E. F., (editor), "Modern Mathematics for the Engineer", McGraw-Hill Inc., New York, 1956.

[BELFORD75]

Belford, G. G., "Dynamic Data Clustering and Partitioning", CAC #162, Centre for Advanced Computation, Research in Network Data Management and Resource Sharing, University of Illinois at Urbana-Champaign, May, 1975.

[BLASGEN76]

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

[BLEIR67]

Bleir, R. E., "Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System (TDMS)", Proceedings of the ACM National Conference, 1967.

[BOYCE74]

Boyce, R. F., Chamberlin, D. D., King, W. F., Hammer, M. M., "Specifying queries as relational expressions: SQUARE", Data Base Management, Proceedings of the IFIP Working Conference, North Holland Publishing Co., Amsterdam, The Netherlands, April, 1974.

[BROWN59]

Brown, R. G., Statistical Forecasting for Inventory Control, McGraw Hill Inc., New York, 1959.

[BROWN62]

Brown, R. G., "Smoothing, Forecasting and Prediction of Discrete Time Series", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1962.

[CARDENAS75]

Cardenas, A. F., "Analysis and Performance of Inverted Data Base Structures", CACM, Vol. 18, No. 5, May, 1975.

[CHAMBERLIN76]

Chamberlin, D. D., "Relational Data-base Management Systems", ACM

Computing Surveys, Vol. 8, No. 1, Mar., 1976.

[CODD70]

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", CACM Vol. 13, No. 6, June, 1970.

[CODD71]

Codd, E. F., "A Data Base Sublanguage founded on the Relational Calculus", Proceedings of the ACM-SIGFIDET Workshop on Data Description, Access, and Control, 1971.

[CZARNIK75]

Czarnik, B., Schuster, S., Tsichritzis, D., "ZETA: A Relational Data Base Management System", Proceedings of the ACM Pacific Regional Conference, April, 1975.

[DATE75]

Date, C. J., "An Introduction to Data Base Systems", Addison-Wesley, Reading, Mass., 1975.

[DENNING71]

Denning, P. J., Eisenstein, "Statistical Methods in Performance Evaluation", Proceedings of the ACM Workshop on System Performance Evaluation, April, 1971.

[FARLEY75]

Farley, J. H. G., Schuster, S. A., "Query Execution and Index Selection for Relational Data Bases", Technical Report CSRG-53, University of Toronto, Mar., 1975.

[FEIGENBAUM63]

Feigenbaum, E. A., Feldman, J., (editors), "Computers and Thought", McGraw-Hill Inc., 1963.

[GOTLIEB75]

Gotlieb, L. R., "Computing Joins of Relations", Proceedings of the ACM SIGMOD Conference, May, 1975.

[HELD75A]

Held, G. D., Stonebraker, M. R., Wong, E., "INGRES: A Relational Data Base System", Proceedings of the AFIPS National Computer Conference, May, 1975.

[HELD75B]

Held, G. D., "Storage Structures for Relational Data Base Management Systems", Memorandum No. ERL-M533, University of California, Berkeley, Aug., 1975.

[HOFFER75]

Hoffer, J. A., Severance, D. G., "The Use of Cluster Analysis in

Physical Data Base Design", Proceedings of the International Conference on Very Large Data Bases, September, 1975.

[KING74]

King, W. F., "On the Selection of Indices for a File", IBM Research RJ 1341, San Jose, Jan., 1974.

[KENNEDY72]

Kennedy, S. R., "A File Partition Model", Information Science Technical Report No. 2, California Institute of Technology, May, 1972.

[KNUTH73]

Knuth, D., "Sorting and Searching", The Art of Computer Programming, Vol. 3, Addison-Wesley, 1973.

[KUEHN63]

Kuehn, A. A., Hamburger, J. M., "A Heuristic Program for Locating Warehouses", Management Science, Vol. 9, No. 4, July, 1963.

[LEVIN75]

Levin, K. D., "Adaptive File Assignment in Distributed Data Bases", internal working paper, the Wharton School, University of Pennsylvania, 1975.

[LIU68]

Liu, C. L., "Introduction to Combinatorial Mathematics", McGraw-Hill Inc., 1968.

[LUM71]

Lum, V. Y., Ling, H., "An Optimization Problem on the Selection of Secondary Keys", Proceedings of the ACM National Conference, 1971.

[MARTIN75]

Martin, J., Computer Data-Base Organization, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1975.

[MEIER69]

Meier, R. C., Newell, W. T., Pazer, H. L., "Simulation in Business and Economics", Prentice Hall Inc., New Jersey, 1969.

[MUTH60]

Muth, J. F., "Optimal Properties of Exponentially Weighted Forecasts", American Statistical Association Journal, June, 1960.

[PECHERER74]

Pecherer, R. M., "Efficient Retrieval in Relational Data Base System", Memorandum No. ERL-M547, University of California, Berkeley, Oct., 1975.

[ROTHNIE72]

Rothnie, J. B., "The Design of a Generalized Data Management System", Ph. D. Dissertation, Department of Civil Engineering, MIT, Sept., 1972.

[ROTHNIE74]

Rothnie, J. B., Lozano, T., "Attribute Based File Organization in a Paged Memory Environment", CACM, Vol. 17, No. 2, Feb., 1974.

[ROTHNIE75]

Rothnie, J. B., "Evaluating Inter-entity Retrieval Expressions in a Relational Data Base Management System", Proceedings of the AFIPS National Computer Conference, Vol. 44, 1975.

[SCHKOLNICK75]

Schkolnick, M., "Secondary Index Optimization", Proceedings of the ACM-SIGMOD International Conference on Management of Data, May, 1975.

[SMITH75]

Smith, J. M., Chang, P., "Optimizing the Performance of a Relational Data Base Interface", CACM, Vol. 18, No. 10, Oct. 1975.

[STOCKER73]

Stocker, P. M., Dearnley, P. A., "Self Organizing Data Management Systems", The Computer Journal, Vol. 16, No. 2, 1973.

[STONEBRAKER74]

Stonebraker, M., "The Choice of Partial Inversions and Combined Indices", International Journal of Computer and Information Sciences, Vol. 3, No. 2, 1974.

[THEIL64]

Theil, H., Wage, S., "Some Observations on Adaptive Forecasting", Management Science, Vol. 10, No. 2, Jan., 1964.

[WAGNER73]

Wagner, R. E., "Index Design Considerations", IBM System Journal, Vol. 4, No. 3, 1973.

[WELCH76]

Welch, J. W., Graham, J. W., "Retrieval Using Ordered Lists in Inverted and Multilist Files", Proceedings of the ACM SIGMOD Conference, June, 1976.

[WINTERS60]

Winters, P. R., "Forecasting Sales by Exponentially Weighted Moving Averages", Management Science, Vol. 60, 1960.

[WONG76]

Wong, E., Youssefi, K., "A Strategy for Query Processing", ACM Transactions on Database Systems, (to appear).

[YUE75]

Yue, P. C., Wong, C. K., "Storage Cost Considerations in Secondary Index Selection", International Journal of Computer and Information Sciences, Vol. 4, No. 4, 1975.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # LCS-TR-166

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 96(103-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form (2) ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP:	(1-96)
UNNUMBERED TITLE PAGE	2-96
SCANCONTROL, COVER, DOD(2), TRGTS(3)	(97-103)

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 11/30/95 Date Returned: 12/7/95

Scanning Agent Signature: Michael W. Cook

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TR-166	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: Index Selection in a Self-adaptive Relational Data Base Management System	5. TYPE OF REPORT & PERIOD COVERED: S.M. Thesis 1975-1976
6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TR-166	
7. AUTHOR(s): Arvola Y. Chan	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0661
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Massachusetts Institute of Technology Laboratory for Computer Science 545 Technology Square; Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency Department of Defense 1400 Wilson Boulevard Arlington, Virginia 22209	12. REPORT DATE: September 1976 13. NUMBER OF PAGES: 98
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Office of Naval Research Department of the Nacy Information Systems Program Arlington, Virginia 22217	15. SECURITY CLASS. *(of this report)*: Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Data base management, secondary indices, inversions, adaptive data base system, global optimization, automatic physical data base reorganization, performance monitoring, heuristics

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The development of large integrated data bases that support a variety of applications in an enterprise promises to be one of the most important data processing activities of the next decade. The effective utilization of such adata bases depends on the ability of data base management systems to cope with the evolution of data base applications. In this thesis, we attempt to develop a methodology for monitoring the developing pattern

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

of access to a data base and for choosing near-optimal physical data base organizations based on the evidenced mode of use. More specifically, we consider the problem of adaptively selecting the set of secondary indices to be maintained in an integrated relational data base. Stress is placed on the acquisition of an accurate usage model and on the precise estimation of data base characteristics, through the use of access monitoring and the application of forecasting and smoothing techniques. The cost model used to evaluate proposed index sets is realistic and flexible enough to incorporate the overhead costs of index maintenance, creation, and storage. A heuristic algorithm is developed for the selection of a near-optimal index set without an exhaustive enumeration of all possibilities.

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**