LBL-21623
c.2

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

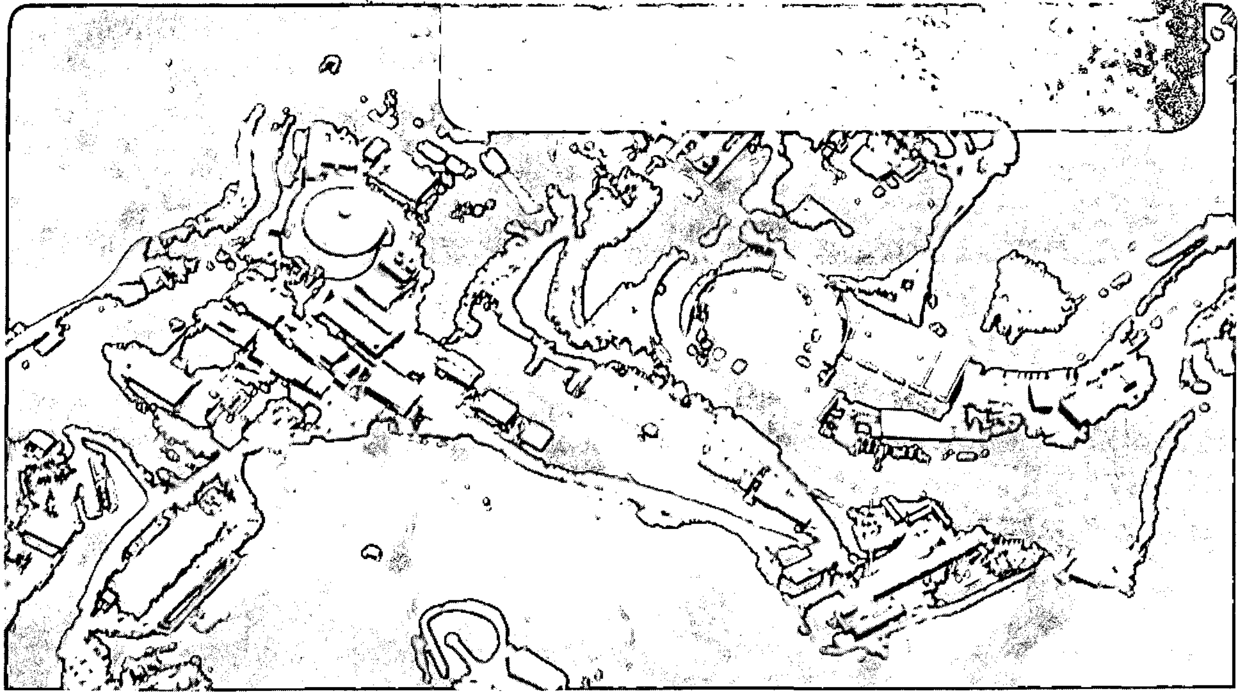
Computing Division

To be presented at the 3rd International Workshop on Statistical and Scientific Database Management, Luxembourg, Grand-duchy of Luxemburg, July 22-24, 1986

SCIENTIFIC AND STATISTICAL DATA MANAGEMENT RESEARCH AT LBL

F. Olken, D. Rotem, A. Shoshani, and H. Wong

June 1986



LBL-21623
c.2

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

LBL-21623

Scientific and Statistical Data Management Research at LBL

F. Olken
D. Rotem
A. Shoshani
H.Wong

Computer Science Research Department
University of California
Lawrence Berkeley Laboratory
Berkeley, California 94720

June, 1986

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098.

Scientific and Statistical Data Management Research at LBL [*]

Frank Olken
Doron Rotem [†]
Arie Shoshani
Harry K.T. Wong

Computer Science Research Deptartment
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

This paper is a review of scientific and statistical data management research at LBL in recent years in the areas of: logical modeling and user interfaces, database operators, and physical organization and access methods. In the area of logical modeling and user interfaces we discuss: SUBJECT, a system for organizing multi-dimensional data, GUIDE, a graphical query system, and logical modeling of temporal data. In database operators we dicuss sampling from relational databases, and transposition of compressed data. In the area of physical DB organization and access methods we discuss: header data compression, rearrangement of data arrays to enhance data compression, batched interpolation search, bit transposed file organization, techniques for controlling overflow from multi-dimensionsal data structures (e.g. grid files), and data structures for temporal data.

I Introduction

This paper is intended as a review of the research in scientific and statistical data management at Lawrence Berkeley Laboratory (LBL) in recent years. In an earlier paper [SW85] we have identified the research issues in scientific and statistical databases (SSDBs). The research areas have been organized into three major groups: logical modeling and user interfaces, database operators, and physical organization and access methods. In this paper we follow this organization. We discuss here in some detail our more recent research results, and some older results which bear on our more recent work. Our other work in SSDBs in also cited for completeness.

In the area of logical modeling and user interfaces we commence with a discussion on modeling, browsing, and querying of multi-dimensional data in the SUBJECT system. We then review a graphical user interface (GUIDE) which employs graphical searching ond browsing over entity-relationship schemas and subject directories. Next, we discuss the logical modeling of temporal data which are commonly found in SSDBs, and operators for the manipulation of such data.

In the area of database operators, we discuss first the implementation of sampling from relational databases, and then algorithms for the transposition of compressed data.

In physical organization and access methods we begin with a review header data compression and the rearrangement of data arrays to enhance data compression efficiency. Next we examine batched interpolation searching. We then review our work on a file structure which is an extreme version of the transposed file organization, called bit transposed files. Then we have a discussion on reorganizing multidimensional data structures (such a multi-paging, or grid files) to minimize page overflows. We conclude this section with a discussion of physical structures

*Issued as LBL technical report LBL-21623. This work was supported by the Director, Office of Energy Research, Office of Basic Research Sciences, Division of Engineering, Mathematical and Geosciences of the U.S. Department of Energy under Contract DE-AC03-76SF00098.

†On leave from Univ. of Waterloo, Canada. Partially supported by Canadian NSERC Grant A3055.

and access methods for temporal data.

We have published several papers which describe and analyze the characteristics of SSDBs. The following two papers explain the motivation of our research program. In [Sho82] we analyzed several statistical applications, reviewed the existing literature, and discussed some known solutions. In [SOW84] we studied in detail ten scientific database applications and we identified common characteristics among them. Other papers discussing various SSDB issues include: [SM82] and [DNSS83].

II Logical Modeling and User Interfaces

The first two sections below represent two different approaches to user interfaces, but both are designed to alleviate the burden from the user of having to remember names, acronyms, formats, and complex syntax rules. The first section describes a system (SUBJECT) which is based on the modeling of multi-dimensional data in SSDBs as logical graphs. The second section discusses our work on a graphical user interface for data exploration (GUIDE). The third section discusses temporal data in SSDBs, and proposes a logical model and operators for such data.

In addition to the topics discussed in this section, we have published several other papers that deal with logical aspects of SSDBs. They include: the modeling of summary data [Joh81b,Joh81a], spatial data analysis [Mer82], metadata management [McC82], a semantic model for SSDBs [Kre82], and micro/macro statistical DB management [Won84].

1 SUBJECT: modeling multi-dimensional data

1.1 Motivation

One of the motivations for the SUBJECT system is the modeling of multi-dimensional structures that commonly exist in SSDBs. While in business applications the concept of an "entity" (e.g. employee, department, bank account) is common, it is more convenient in SSDBs to think about "cases," which are instances of an experiment, a simulation, or a survey. While cases can be given unique identification numbers (so that they can be thought of as instances of entities), they are commonly characterized by a set of parameters or categories. For example, trade data can be identified in terms of the exporting country, importing country, commodity, year, and month. Similarly, a corrosion experiment can be described in terms of the temperature, acidity, salinity, length of exposure, and the material used. In all such examples there exists naturally a multi-dimensional space for which measured data are collected.

The difficulty of dealing with multi-dimensional spaces is further compounded by the fact that each dimension can itself have a complex (usually hierarchical) structure. For example, a trade commodity can be broken into categories of food, energy, clothing, etc. Each category can be further broken into sub-categories, such as energy into the sub-categories of oil, coal, and gas. This sub-categorization of a single dimension can continue into many levels, and sometimes overlaps may exist. Representing such complexity requires special facilities at the logical and user interface level.

Another goal of the SUBJECT system is to provide a simple user interface that can be used by novice users with very little training. We wanted to avoid imposing on the user the burden of learning a syntax for expressing queries, and having to remember names, acronyms and formats of data elements before specifying a query. Also, a novice user should be able to browse through descriptive information about the databases (meta-data), select a database to explore, and continue on to express queries using a single uniform interface. At the same time, experienced users should be provided with efficient ways (shortcuts) to accessing the data. In addition, we wanted to avoid the problem of inconsistency of names given to elements that are used in several files by providing some means of sharing the logical description of such elements.

1.2 Approach and solution

The main idea in the SUBJECT system is the use of an directed acyclic graph (DAG) with special node types to represent the logical structure of both the data and the meta-data. The graph structure uses two types of nodes that are called *cluster* nodes and *cross product* nodes. Cluster nodes are used to represent the concept of a collection of items. Thus, a cluster node may represent a "state", whose subordinate nodes are a collection of cities. Similarly, cluster nodes can represent a collection of data values (e.g. years), or a collection of files. Cross product nodes are used to represent multi-dimensional structures, usually parameters or categories of the data. For example, to model oil production by state by year by oil type, we would use a cross product node to represent the combination of state by year by oil type.

In Figure 1 we illustrate the use of cross product nodes (marked with an X) and cluster nodes (marked

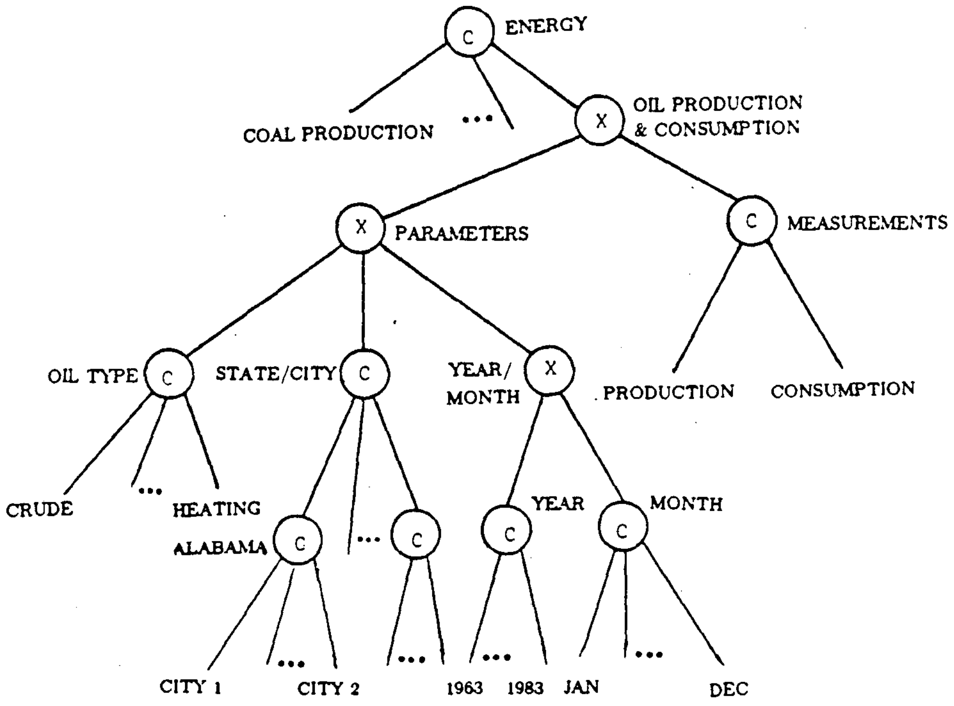


Figure 1: An example of a SUBJECT graph

with a C). Starting from the top, "energy" is a (meta-data) cluster node of the files relating to energy data. "Oil production and consumption" is a cross product node representing a file made of several parameters and measurements. The multi-dimensionality of the parameters are represented as a cross product node of "oil type", "state/city", and "year/month". Note that "state/city" is a cluster of clusters, while "year/month" is a cross product of clusters. In general, if a combination of elements are needed to make up an item in the next level, a cross product node is used; if a single element in sufficient to represent an item in the next level, a cluster node is used. The system uses this semantic information in order to perform the aggregation of items correctly (when the user issues a query to extract summaries of the database). However the user need not be aware of node types when browsing through the graph.

A novice user can enter the SUBJECT system at the root node of the directed graph to find the subject categories that exist in the system. By selecting a subject category, the user is provided with more detailed descriptions. The user can continue to browse the meta-data in order to become familiar with the databases available in the system, and eventually select a data file. At this point, the user is allowed to explore the attributes and parameters of the file, and express a query by selecting the desired nodes of the graph. This is where SUBJECT differs from conventional approaches, where users are required to access the meta-data first, remember names and formats of attributes, and then express a query using the syntax of the query language.

An alternative to the browsing capability is provided for experienced users, where they can search for the data file using keys. Thus, they may quickly locate a desired data file, and proceed to express query conditions by moving around the directed graph of that file. The system is also designed to provide documentation associated with nodes in the graph.

Another important concept of the graph representation is that of "node sharing", which permits more than one arc to point to the same node, thus forming directed acyclic graphs. Node sharing allows attribute domains to be shared between different file, providing several advantages: eliminating duplication of data values; achieving consistency of naming, where items that are the same, but reside in different files, are forced to have the same name; and allowing the specification of join domains between files, permitting multiple physical files or fractions of these files to be viewed jointly as a single logical entity.

A detailed description of the SUBJECT system and the advantages it provides are given in [CS81].

To summarize, the most important advantages are: modeling meta-data and data in the same graph structure, supporting both novice and experienced users, eliminating the need to remember names, values and formats of data elements as a prerequisite to query specification, providing a mechanism for avoiding duplication of data and using multiple names across data files, and expressing queries without the need to learn a query language.

2 GUIDE: A Graphical User Interface

2.1 Introduction and Motivation

The main motivation of our work comes from experiences in using query languages of commercial Database Management Systems (DBMSs) and Statistical Packages. Even people with a computer science background often have difficulty using the so-called "high level user friendly languages." Non-expert users may not have the patience, ability, or desire to learn and use these languages correctly. By non-expert users (as opposed to casual users), we mean non-computer science professionals such as social analysts, statisticians or accountants who have to deal with data regularly. The problem becomes much worse in an environment with very large databases that have very large and complex database definitions (schemas). Large statistical databases such as the Census database and energy database are examples of such an environment.

We believe that the following factors are the major reasons for the difficulty in using and understanding query languages.

- The user has to remember too many things.
- Semantically poor data models.
- No feedback during the query process.
- Lack of levels of detail in schemas.
- Lack of meta-data browsing facility.
- Limited and difficult-to-use aggregation facility.
- Lack of integrated, easy-to-use data display.

2.2 Approach

Our goal is to put together a set of facilities into an integrated system that address some of the above mentioned problems.

First, there are facilities in the system that remove the memory burden from the user. The facilities provide menus, examples, illustrations, and help messages at any stage of query formulation.

Second, a version of the Entity-Relationship (E-R) model is used to represent relationships between entities explicitly. This capability helps the user view the semantics of the DB schema. For experimental purposes, the graphics user facility interfaces to a subset of the query language CABLE [Sho78] which is implemented on top of the database system DATATRIEVE, a DEC VAX/VMS product.

Third, we chose to use a graphics user interface for the following reasons:

1. The E-R model schema can be displayed as a network of objects, each object representing an entity or relationship type. This gives the user an overall view of the schema at all times.
2. Queries can be expressed as a traversal along the network of entities. Colors can be used to indicate the paths of the queries, and hence, pictorially indicate the scope and meaning of the queries.
3. Parts of the schema can be selectively made visible or invisible and thus provide the basis to the implementation of multiple levels of detail to aid in the understanding and use of the schema.

Fourth, the user can build the query in a piecemeal fashion and have intermediate results of partial queries available at all times.

Fifth, to handle the problem of meta-data and the large number of the entities and their attributes, two kinds of directories are provided. The first is called a "hierarchical subject directory" (similar to that in SUBJECT which can be used to organize the entities into logical groups hierarchically. The user is guided by the system through this directory to locate the relevant part of the schema for which queries can be expressed. This is also a useful facility to browse and explore the subject matters of the database. The second kind of directories are called "hierarchical attribute directories," and are used to organize attributes of entities (or relationships) into groups similar to the subject directory. Each entity or relationship type has an attribute directory. Both kinds of directories are implemented as menus.

Sixth, a facility is made available to "rank" objects according to their "relevancy" to a particular group of users. The entity and relationship types are ordered and classified into groups according to the users' interests and the frequency of access in queries. Different groups of users may have different classifications. The first group of objects (with rank 1) is included the most important objects or focal points of the schema. The second group (with rank 2) of objects provides (together with objects from the first group) the next, more detailed, level of the schema. As the rank of groups goes higher, we see more details. "Focus" can be specified in the schema so that a selected object will be placed in the center of the screen. Also, there are commands to move the picture around the screen, to zoom-in and zoom-out on the selected part of the picture. The idea is to present the right level of detail and the right part of the schema.

Seventh, an interactive aggregation facility is available in which the user is allowed to select those attributes from any entities as "parameter" attributes of the aggregation (called category attributes). Also, the user can select a set of "measured" attributes from any entities (called summary attributes), and for each such attribute, a simple descriptive statistics function such as sum, average, etc. can be applied.

Eighth, a graphical data display facility is made available for the display of the aggregation result. Data display formats such as tabular, pie, bar, and plot are implemented. The important property is that the interface is completely menu-driven, and integrated with the rest of the GUIDE system in that it can be invoked from any point in GUIDE without having to get off GUIDE itself.

All facilities are offered to the user through graphics menus.

2.3 Results

An implementation of a prototype system based on the approach mentioned before was completed. There are five stages of query formulation in GUIDE: schema definition, schema exploration, query expression, aggregation, and graphics output display.

Data definition stage The Data Base Administrator (DBA) provides information about the schema during the definition stage. In addition to information on entities, relationships, and their attributes, there are examples and explanations of these objects. The graphical layout of the schema is fed into the system in this stage. Facilities are provided to the DBA to do the following:

- specify a graphical layout of the schema;
- build a hierarchical subject directory for the schema objects;
- build a hierarchical attribute directory for each entity and relationship type;
- specify the "importance ranking" of entity and relationship in the schema. Every object is given a rank (currently from 1 to 5).

Schema exploration stage During this stage, the user can use the hierarchical subject directory to reach the most relevant part of the schema. From there the schema can be graphically examined at several levels of abstraction and only objects above a specified importance ranking are made visible. The user can also graphically edit the schema so that irrelevant objects can be removed from the screen. With the relevant part of the schema selected and displayed at the desired level of detail the user is now ready to express queries.

Query expression stage In the query stage, the user can build up a query in a piecemeal fashion. The database retrieval results of a partial query can be shown if so desired. Examples and explanations on any object on the screen can be requested. Graphically, the user will be traversing a network of objects. The query is a path selected by the user and shown in different colors for each partial query. The user is also encouraged to experiment with different conditions on the schema objects by adding or subtracting conditions and the result of these experimentations are available at any time. Piecemeal formulation of a query is an important facility. It is achieved through the formulation of "local queries." The user can concentrate on several parts of the schema being shown on the screen and can formulate a "local query" on each part so that each local query is completely independent of another. The idea is to allow the user to have a focused vision of small parts of the schema so that local results can be obtained and understood without having to compose a complex query covering a large schema space at the same time. The user can then link local queries to form a complex query. This complex query can then be treated as a local query when the user expresses additional local queries. All the local queries can be linked to form yet another more complex query, etc. This process continues until the final query is formulated. The retrieval results of each local query are always available for display. The result of linking several local queries is also displayable at any time.

Aggregation stage During this stage, the user can select graphically the category and summary attributes from entities for aggregation. Facilities are available for the user to examine any entity included in a previous query by displaying the available attributes within that entity, to find out the description on the entity and its attributes, and select aggregate functions for the summary attributes.

Graphics data display stage In this stage, the aggregation result can be displayed in various forms such as graphs, bar charts, pie charts, etc., with a set of optional graphics enhancements. A graphics package has been interfaced to GUIDE for graphical displays.

2.4 Summary

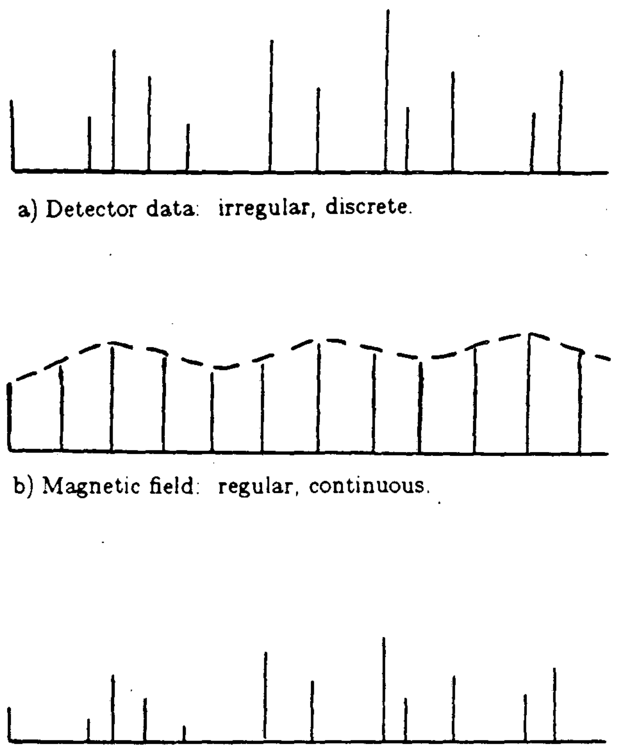
A prototype of GUIDE became operational in summer of 1983. A real database of Current Population Survey from the Bureau of Labor Statistics was installed. From the experience of using the system, we learned that the concepts motivated in GUIDE are justified, and the approach we took is a reasonable first step. But we also learned that the software and hardware environment in which GUIDE was built are not really powerful enough to provide the kind of interaction speed needed for a powerful interface such as GUIDE. With the advent of powerful and inexpensive graphics hardware and much better graphics software, a more efficient implementation of GUIDE can be obtained. This work was reported in [WK82].

3 Modeling Temporal Data

3.1 Problem description

Our interest in managing temporal data stems from the needs of SSDBs, where physical experiments, measurements, simulations, and collected statistics are often in the time domain. Such applications are inherently time dependent and the ability to manage data in the time domain is essential. The time aspect may not be as important in other applications (such as business databases) since such applications can often be satisfied with the most updated information only. Historical data would typically be archived, but not available on-line.

Currently, there are no commercial systems that explicitly model or support temporal data. Even in the case that historical information is kept in commercial applications, they are not typically accessed across the time dimension; rather one slice of the data for a certain point in time is accessed. Recently, there is renewed interest and new research in modeling and managing time for business applications. Perhaps one of the reasons is that we can now afford to store on line historical information since memory


a) Detector data: irregular, discrete.

b) Magnetic field: regular, continuous.

c) Corrected detector data.

Figure 2: Examples of Time Sequences

and magnetic disk storage costs are rapidly decreasing, as well as the advent of optical disks.

Temporal data have special semantic features. There may be many interpretations to the time domain that could be confusing unless precise models and operations in that domain exist. To illustrate such semantic features we contrast below two sequences (in the time domain) of measurements taken in a typical high energy physics experiment: the sequence of detector measurements and the sequence of magnetic field measurements. In this experiment, high energy particles are collided and the paths of the resulting sub-particles are recorded by detectors. Each detector "fires" as the sub-particle goes by it. The sub-particles are subject to a magnetic field which causes them to curve according to their electric charge. The magnetic field tends to drift, and therefore it is measured at regular intervals. These magnetic field measurements are later used (during analysis) to correct the detector measurements.

There are several other aspects to such an experiment, but for simplicity we consider here only the measurements of the detectors and the magnetic field. The typical pattern of these measurements is shown in parts a) and b) of Figure 2.

Note the differences between these two sequences of measurements. The first difference is that in the detector data the sequence of measurements appear in *irregular* positions over the time domain, while in the magnetic field data the measurements exist at *regular* intervals. This difference is important when we use both sequences to generate the corrected detector data shown in part c) of Figure 2. For each detector measurement we need to find the corresponding magnetic field value. Since magnetic field measurements do not always exist for the times that detector measurements are taken, the corresponding magnetic field values have to be interpolated from existing ones. This leads us to the second difference. In the detector data, measurements have the semantic interpretation of being *discrete*; that is values exist only at the points of measurements. In contrast, the magnetic field sequence is interpreted as a *continuous* sequence in that values exist for any time point (and are interpolated if necessary.)

The current practice of dealing with applications involving temporal data such as the example above, is to develop special purpose programs for each application. Our goal is to provide data management tools that deal with temporal data as well defined data elements that can be manipulated using system supported operators. Such tools should be useful across applications, and greatly simplify the development of applications involving temporal data. Furthermore, applications that use such tools could be modified or adapted if experimental conditions change without having to rewrite new application programs. In general, our purpose is to model the various types of temporal data, so that their semantics are clear to the user. It is also necessary to provide query facilities that permit users to specify conditions in the time dimension, as well as correlate temporal data sequences that may be varying at different rates.

3.2 Approach and solution

Our approach to modeling temporal data is to introduce a temporal data element called a *time sequence* (TS) which can assume different properties. Structurally, a TS consists only of the series of (time,value) pairs. The interpretation of the time sequences is determined by their properties. Thus, in the example above, both the detector data and the magnetic field data are conceptually represented as a sequence of (time,value) pairs, but the detector data is a "regular" TS of type "discrete", and the magnetic field data is an "irregular" TS of type "continuous" which has an interpolation routine associated with it. The result is that both of these time sequences can be treated uniformly from a syntactic point of view. Thus, operations over and between time sequences can be specified even though they have different semantic properties. In the example above, the value of the magnetic field that corresponds to a given detector value could then be retrieved or calculated by a system that supports time sequences. The system would know to use the interpolation routine (if needed) to produce the correct value, because the magnetic field TS is of type "continuous".

In a paper describing this work [SK86] we have identified four properties that should be associated with time sequences. These properties are sufficient to capture the semantics of temporal data found in scientific applications that we have observed. These properties specify the "regularity" of the TS, its type (discrete, continuous, step-wise constant, and interval), whether it is "static" (i.e. no new values are expected to be added) or "dynamic" (i.e. data values can be added dynamically), and the time domain description (start time, end time (for the static case), and time units). Distinguishing between regular and irregular TSs is clearly useful to a user as part of describing the behavior (semantics) of TSs. However, there are two other reasons for this distinction. The first is that there exist special statistical analysis methods (called "time series analysis") that can be applied only to regular TSs. The second reason is that regular TSs can be stored and accessed more efficiently that irregular TSs. This aspect is discussed in some detail in the subsection describing temporal data structures (in the Physical Organization section.).

In general, we would like to specify the retrieval or manipulation of any subset of values of a TS in a single operation. For example, we may want to perform some arithmetic over the values of a certain detector's TS, and generate a new TS as a result. In addition, we may want to apply the same operation to a set of TSs, such as a subset of desired detectors. We refer to a collection of TSs as a *time sequence array* (TSA), where rows correspond to individual TSs, and the columns represent points in time. We have identified classes of operations that can be applied to TSAs. As is discussed in more detail in [SK86], these operators are designed to specify restrictions on TSAs in both dimensions (i.e. making selections in the time domain as well as selecting the desired TSs), and to specify operations over one or more TSAs to produce a new TSA. Thus, for example, they would permit the selection of a subset of detectors and a certain time range, and specify some operation to be applied to the set of time sequences of these detectors using a few concise commands.

This work is still in progress, and the precise syntax of operators over TSAs will be developed in the future. However, the framework of defining the precise semantics of TSs and operators over TSAs seem fundamental to clear and concise definition, retrieval and manipulation of temporal data.

III DB Operators

In this section of the paper we briefly review our work on sampling from relational databases and on transposition algorithms for compressed data.

In another paper, we discuss the question of which operators should be included in a SSDB management system [Olk83].

1 Sampling

1.1 Introduction

This section is concerned with the question of how to efficiently extract random samples of relational queries from a relational data management system. Our goal is to obtain the samples without first computing the entire query result which is to be sampled. This work was reported in [OR86b].

Why sample?

Random sampling is used on thos‿ occasions when processing the entire dataset is not necessary and is considered too expensive in terms of reponse time or resource usage. The savings generated by sampling may be due to reductions in the cost (in reponse time or resources, CPU and I/O time) in retrieving the data from the DBMS. Retrieval costs are significant when dealing with large statistical or scientific databases.

In addition, savings may result from reductions in the cost of subsequent "post processing" of the sample. Such "post processing" of the sample may involve expensive statistical computations, or further physical examination of the real world entities described by the sample. Examples of the latter include physical inspection and/or testing of components for quality control, physical audits of financial records, and medical examinations of sampled patients for epidemiological studies.

Clearly for sampling to be useful, the application must not require the complete answer to the query. Thus random sampling is typically used to support statistical analysis of a dataset, either to estimate parameters of interest or for hypothesis testing. Applications include scientific investigations such as high energy particle physics experiments, quality control, and policy analyses. For example, one might sample a join of welfare recipient records with tax returns or social security records in order to estimate welfare fraud rates.

Why put sampling in DBMS?

Given that one wants to perform sampling, is it worthwhile to put the sampling operator into the DBMS?

We believe that one should put sampling operators into the DBMS for reasons of efficiency. By embedding the sampling within the query evaluation, we can reduce the amount of data which must be retrieved in order to answer sampling queries, and can exploit indices created by the DBMS.

Sampling can also be used in the DBMS to provide estimates of the answers of aggregate queries, in applications where such estimates may be adequate (e.g. policy analysis), and where the cost in time or money to fully evaluate the query may be excessive.

Sampling may also be used to estimate database parameters used by the query optimizer to choose query evaluation plans.

1.2 Sampling from Relations

In the technical report [OR86c] we discuss how to apply classical sampling methods to sampling from entire database relations resident on disk. We examine sampling from variably blocked files, grid files and B^+–tree indexed files.

One basic tactic employed is acceptance/rejection sampling to accommodate the variable numbers of records stored on pages.

We also discuss the use of sequential sampling algorithms for sampling from relations as they are generated.

1.3 Sampling from Relational Operators

In the VLDB paper [OR86b] we show how to sample the output of individual relational operators such as selection, projection, intersection, union, difference, and join. Our sampling algorithms avoid the need to first compute the entire result of the relational operator. These sampling techniques form the basic building blocks for sampling from more complex composite queries. The techniques entail a synthesis of the basic file sampling techniques and algorithms for implementing relational operators. We discuss only simple random sampling.

The fundamental problem with sampling from relational operators is that most of the relational operators (except selection and intersection) modify the inclusion probabilities of records in a non-uniform fashion. Hence it is not generally possible to simply interchange sampling and relational operators.

Our basic strategy for dealing with this problem is the use of acceptance/rejection sampling techniques to adjust the inclusion probabilities so as to produce a uniform random sample.

Typically these methods require information on the cardinality of the sampled domain values in the target relations. This information can be readily obtained only if the target relations are either indexed or hashed.

Given the necessary indices, the sampling algorithms attain computational complexity which are proportional to the sample sizes. However, the constant factor is usually the inverse of the ratio of average cardinality of domain values to maximum cardinality. If this ratio is large, these sampling algorithms may be quite expensive.

Material	Temp.	Acidity	Salinity	Time	Corrosion
Steel	1000	100	1	10	0.7
Steel	1000	100	1	20	0.9
"	"	"	"	.	1.2
"	"	"	"	.	1.5
"	"	"	"	.	1.7
"	"	"	"	100	2.3
"	"	"	2	10	0.8
"	"	"	2	20	1.0
"	"	"	"	.	1.2
"	"	"	"	.	1.5
"	"	"	"	.	1.8
"	"	"	"	100	2.4
"	"	"	.	.	.
"	"	"	.	.	.
"	"	200	.	.	.
"	"	.	.	.	.
"	"	.	.	.	.
copper	"	.	.	.	.
"	.	.	.	.	.
"	.	.	.	.	.

Table 1: Multi-factor parametric experiment

2 Transposition

2.1 Motivation

The most common operations over summary databases (besides searching) are transposition and aggregation. The former requires a re-ordering of the category attributes for the purpose of presentation and analysis. An example is to transpose the database in Table 1 so that temperature and acidity appear after material, salinity and time. Transposition operations are required to obtain the popular file structure called transposed file. Transposed files are the most efficient file structure for many SSDB applications. The motivation of transposed files is that the typical access of SSDBs is to fetch a long sequence of individual records and extract a small number of attributes. By storing the records as a collection of contiguous attribute columns, i.e., all of the data for a field (attribute) are stored together, only those attribute columns which are needed for a query need be retrieved.

Aggregation operations are used to "collapse" away some category attributes to obtain a more concise database to facilitate more efficient analysis. An example of aggregation is a request such as: "what is the average corrosion level of steel by temperature, acidity, and salinity?" Since the dimension time is ignored in the request, the corrosion values are aggregated on the time dimension. The answer to the above request is obtained by averaging the corrosion level values over all time values for each combination of the other category attributes.

Efficient methods of performing transposition and aggregation are the keys to efficient SSDB system support of data analysis. Since many large summary SSDBs are typically compressed, efficient transposition and aggregation methods directly over compressed data without first decompressing are important.

Note that transposition and aggregation operations are closely related. An aggregation operation on attribute A can be realized by first transposing A from its original position to the right of the rightmost category attribute in the database, then the corresponding summary attribute values are aggregated (typically by a simple arithmetic operations such as sum, weighted average, etc.). For example, collapsing the temperature dimension from the database involves transposing the attribute to the right of the time attribute, then the corrosion values are aggregated.

Our approach to transposition and aggregation is to design new algorithms that can operate directly on compressed data without first decompressing them. In [WL86b] several algorithms are developed and described in detail. Most commercial statistical DBMSs decompress the data first.

2.2 Results

Below the main idea and the applicability of each algorithm will be briefly highlighted. A detailed description can be found in [WL86b].

The first algorithm is a "general" algorithm in the sense that it can be used in all situations. First, the physical database is read, and for each data item, a "tag" is computed and stored with the data item on disk. A tag is the logical sequence number for the data item in the transposed space. The second step involves sorting the tag and data item pairs in ascending order of the tags. After the sorting is done, the tags associated with the data item are discarded. As the tags are stripped, the necessary headers for the data items are generated and these headers and the data items represent the result of the transposition.

The second algorithm performs the operation in main memory in one pass. This is feasible in the event when the transposed subspace is small enough to fit into main memory. The main idea of the algorithm involves scanning the physical database once, and employing the reverse array linearization to find the proper slot for each data item in the memory buffer. A compression algorithm then runs over the data in memory and the result is stored in compressed form on disk.

For the case that the transposed subspace is too large to fit in main memory, a third algorithm can be used. The algorithm takes advantage of the situation when there are a small number of large fragments of transposed subspace that are already in the right position. The algorithm involves the merging of these fragments, and compressing of the result. This algorithm is used instead of the first algorithm if the number of fragments is small.

A fourth algorithm takes advantage of the situation when the cross-product of the cardinalities of the transposed attributes are relatively small and they can be moved as a group. In this situation, N buffers are used to store the temporary result of transposition where N is equal to the product of cardinalities of the transposed attributes. This algorithm is slower but not as memory intensive as the second algorithm. But when applicable, it offers better performance than the first and third algorithms.

These transposition algorithms have been analyzed and implemented. They operate directly on compressed data without the need to first decompress them. The methods proposed are applicable to databases that are compressed using the general method of run-length encoding. A decision procedure is also given to select the most efficient algorithm based on the transposition request, available memory, as well as the database parameters. Formulas have been developed which identify the required memory space,

All four algorithms are implemented using C. The algorithms have the same order of I/O performance as that of other published algorithms for transposing uncompressed data. Since aggregation operations can be developed on top of transposition operations, the result of this work can be applied directly to efficient aggregation algorithms on compressed data.

In conclusion, direct manipulation of compressed data can yield great efficiency. Algorithms need to be developed and analyzed for other operators on compressed data. Transposition is just one (and important) such operation. We are now researching other operators which can be applied to compressed data.

IV Physical Organization and Access Methods

Much of our work has been done in physical organization and access methods (including compression), because SSDBs exhibit unique data and query characteristics. The large size of SSDBs and the inadequacies of commercial data structures has compelled attention to specialized methods of compressing, organizing and accessing SSDBs.

In addition to the topics discussed in detail below we have published several papers on: database machine design for SSDBs [Haw82], main memory database techniques [DKO*84], and a survey of physical database support techniques for SSDBs [Olk86].

1 Header Compression

1.1 Problem

The problem addressed by header compression is the design of data compression methods for statistical data which permit fast random access to the compressed data.

Many statistical databases include large tables of summary statistics (e.g., contingency tables, census tables, etc.). Such tables often contain many missing and/or zero values. Furthermore, the counts in such tables are often highly skewed in distribution, with many small counts (which could be stored in a few bytes) and fewer large counts (requiring several bytes). Such tables are often stored by sorting them in lexicographic order of the keys. This leads to clustering of data values, especially of nulls and zeros.

Data compression offers savings in disk space requirements. Furthermore, for large accesses of contiguous blocks of data often found in SSDB queries, data compression can yield significant savings in disk transfer time. While data compression is normally thought to increase the CPU time needed to access data (because of the required decompression) directly processing the compressed data (e.g., for transposition and aggregation operations) can yield significant savings in CPU time. This topic is discussed elsewhere in this paper.

1.2 Approach

The header compression technique is composed of four basic ideas:

1. run length compression of strings of nulls,
2. segregating the run length information from the compressed data,
3. organizing the cumulative run lengths as a B-tree index,
4. representing cumulative run lengths via partial sums trees.

1.3 Run length encoding

Run length encoding is a classic ad hoc compression technique, which is especially well suited to the compression of statistical data which often displays clustering of null(zero) and non-null elements into lengthy runs.

The basic idea is to replace consecutive sequences (runs) of null (zero) elements with a count of the run length. Runs of nonzero elements must also be prefaced with a count (run length).

1.4 Segregation

In applications where sequential decoding of the compressed data is sufficient (e.g., communications and tape based applications) the run length information is interleaved with the compressed (nonzero) data.

However, this implies that the time to access some element is proportional to its location in the compressed dataset. If, instead, we segregate the run lengths from the compressed data, then we find an element in time proportional to its location in the list of runs. If the runs are long, this can be an appreciable savings.

1.5 B-tree Index

Once we have segregated the run lengths, we can accumulate them to form an index. A simple array of cumulative run lengths could be searched via binary searching, in time $O(\log_2(n))$. A B-tree index is clearly faster, offering access times of $O(\log_f(n))$, where f is the average node fanout.

The use of interpolation search of cumulative runs to achieve access times of $O(\log\log n)$ is described elsewhere in this paper.

1.6 Partial sum trees

Storing cumulative sums of run lengths presents two problems. One problem is that the sums become large and require several bytes to store, whereas run lengths usually require only one or two bytes. The second problem is that updating them requires time linear is the location of the update (in terms of runs).

Both problems can be ameliorated by representing the cumulative counts (run lengths) via partial sum trees, a well known technique of encoding rank information in trees. Basically, the idea is to store in each internal node of a binary tree the sum of the counts of its leaves. In our case the leaves contain the lengths of the individual runs. By adding and subtracting the counts stored in the internal nodes as we traverse the tree during a search, we can compute the cumulative count for any leaf.

It is easy to adapt the partial sum trees to B-trees, we merely promote the partial sum of a subtree up one level in the B-tree (so that additional disk accesses are not needed when searching within a B-tree node).

The result of all this is that we can attain logarithmic access and updating to the compressed data. Constructing the B-tree (from the bottom up) can be done in time linear with the number of runs.

A detailed description of this work is to be found in [ES80,EOS81]. The second paper also discusses a generalization which permits encodings of variable length numbers.

2 Rearrangement of data to achieve efficient compression

2.1 Overview

Data compression provides several useful benefits for large databases. The most obvious advantage is the reduction in storage costs. Such costs comprise a large portion of the total cost for large archival databases (e.g., scientific, statistical, temporal) for which the average activity rate is often low. Data compression reduces the amount of I/O time required for long sequential data transfers which are commonplace in scientific and statistical database (SSDB) applications. In distributed DBMSs compression can yield significant reductions in communications costs of inter-site transfers of large files. As described in an earlier section of this paper, it is possible to perform database operations (e.g., transposition) directly on compressed data, thereby potentially reducing CPU time requirements.

2.2 Approach

It is often possible to improve the amount of data compression by rearranging the data. This approach was taken in many other applications. A simple example consists of sorting a file, so as to bring together data with similar prefixes, thus improving the possibilities for prefix compression (also called front compression). Another example consists of using special space filling curve rasterization techniques for image data to improve data compression. Of course it is necessary to store a specification of how the data was rearranged.

Our work arose from problems of choosing the storage structures for large sparse multi-dimensional arrays in scientific and statistical databases. Typically such arrays are defined over the cross product of certain category attributes (i.e., attributes defined over discrete categories such as material, corrosive agent, fabrication technique). The concatenation of the category attributes constitutes a key for the data values in the array.

The category attributes need not be stored in the database. Instead array linearization is used to implicitly store the category attributes. The values of the category attributes for each data value can be calculated from its location in the linearized array, assuming a specific ordering for the array linearization. Only the measured attribute (e.g., sub-particle count) need be stored in the database. These arrays are often sparse, with many zero counts. Examples include reliability data, spectra, mortality and census databases. Such databases are frequently retrieved but infrequently modified (often append only). Hence the effort required to reorder and compress the data can be amortized over many retrievals.

The compression method we discuss here is run length encoding. It has long been a popular method of data compression. The basic technique consists of replacing runs (consecutive sequences) of identical values (usually zeros or nulls) by a repetition count and a single copy of the repeated value. A variety of methods have been proposed to encode the counts. In all of these methods the size of the compressed data is proportional to the number of nonzero elements plus the number of runs. Hence, we use the number of runs as our measure of the efficiency with which a particular dataset is encoded.

2.3 Results

Many data compression methods are unsuitable for database applications because they require sequential decoding. However, in previous work (described in the previous section on header compression), it was shown that run length encoding can be combined with a B-tree index so as to provide random access to the data in time $O(\log(n))$, where n is the number of runs of consecutive zero (or nonzero) elements. Again the size of the B-tree index is proportional to the number of runs. There are two possible directions to proceed, we can either model the data deterministically or probabilistically. We use a deterministic model if all of the data is in hand. We have proved that finding an arrangement with less than a specified number of runs is NP-complete for data array having two or more dimensions (attributes). The proof uses a reduction to a variant of the travelling salesman problem. We also showed that the same heuristic algorithms which are used for the travelling salesman problem can be applied in our case. One such heuristic runs in polynomial time and achieves a solution which is at most 50 percent more costly than the optimum.

Alternatively, we can use a concise probabilistic model of the data which predicts the location of the nonzero data elements. We can then search for the average optimal category ordering for the probabilistic model. This approach is more tractable and can be used when an estimate of the probability distribution of values in the array can be obtained. Such estimates may be calculated by sampling a given array or by knowledge of the scientific process which generates the data. Our main result for this model is that the optimal way in which to arrange the categories is a Double Pipe Organ arrangement on each dimension of the multi-dimensional file. An example of this arrangement is shown in Figure 3. This arrangement is a generalization of the well-known Pipe Organ arrangement which was proved to be optimal for minimizing disk arm movement in sequential files. We also devised efficient algorithms that achieve this optimal rearrangement, as reported in [OR86a].

This work raises some new problems about other possible rearrangements that can be made to a file to enhance its compression. The problem of ordering the attributes in the array linearization function is still open and we plan to try to find an exact solution or a good heuristic to solve it. We plan to apply the idea of rearrangement to other types of file structures. For example many types of bit-maps can be rearranged as the order of the attribute values is flexible. In some cases only partial rearrangement is

	Alloy A	Alloy B	Alloy C	Alloy D	Alloy E	Alloy F
Part 1	0	1	0	1	0	1
Part 2	1	1	1	1	1	1
Part 3	0	1	0	1	1	0
Part 4	0	1	0	1	0	0

Table 2: Part Failures: Naive Ordering of Columns

	Alloy A	Alloy C	Alloy E	Alloy B	Alloy D	Alloy F
Part 1	0	0	0	1	1	1
Part 2	1	1	1	1	1	1
Part 3	0	0	1	1	1	0
Part 4	0	0	0	1	1	0

Table 3: Part Failures: Better Ordering of Columns

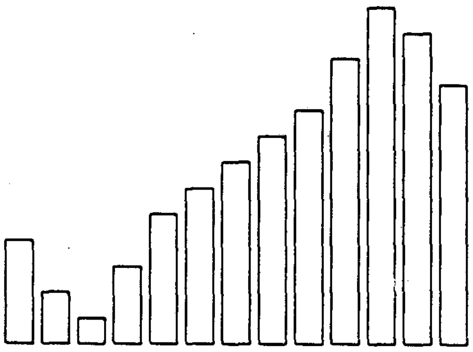


Figure 3: A Double Pipe Organ for 13 elements. Bars are proportional to probabilities of nonzero elements.

allowed or only one dimension may be permuted as order must be preserved on other dimensions. We expect to show that our methods are applicable to a wide range of file designs as well as different compression methods. For example we plan to look at whether Huffman encoding of a file may be improved by some preliminary rearrangement of the attributes.

3 Interpolation Search

3.1 Introduction

Our interest in batched interpolation search comes from three separate search problems in statistical and scientific databases. The first problem involves the searching of data items in a file which has been compressed with the header compression technique [ES80,EOS81]. The second problem is related to the searching of hierarchical relationship implemented in a file structure called hierarchical transposed file [WL86a]. The third problem is the searching of data items in a sparse multi-dimensional data structures. All three of these search problems can be reduced to batched interpolation search over ordered files.

The Interpolation Search Algorithm has received extensive attention. The major result is the $loglog(N)$ [1] (where N is the number of keys in the table) complexity behavior of a single search. However, the effect of batching search queries is not taken into consideration.

The research on interpolation search to date concentrates mainly on main-memory data structure and ignores the secondary memory consideration. We are interested in adding block accesses as well as providing block access approximation expressions to the basic Interpolation Search algorithm. These algorithms are described next.

3.2 Results

An Algorithm for Batched Interpolation Search (BIS)

Let $B = (\alpha_1, \alpha_2, ..., \alpha_k)$ be an ordered collection of search keys to be applied to file X. The idea behind algorithm BIS is that in searching file X for each element α_i in B, one can take advantage of the previous search for element α_{i-1}. Since both B and X are ordered, BIS can start the search for α_i at the place of X where α_{i-1} was found. The savings of batched searching are achieved because the size of file X is monotonically decreasing. The behavior of

[1] "*log*" designates base 2 logarithm.

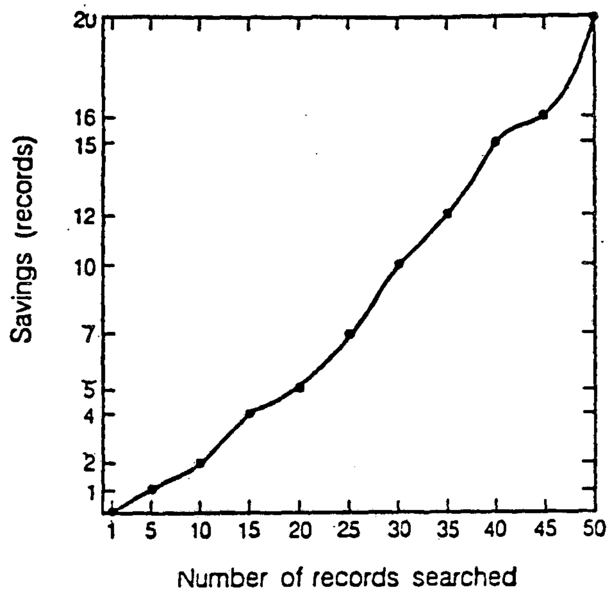


Figure 4: Savings from batch searching.

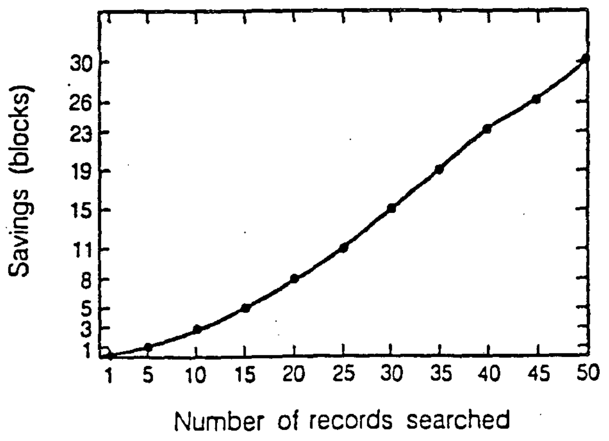


Figure 5: Savings from batch searching with blocking factor of 100.

BIS is still $O(loglog(N))$, but n is reduced by a term proportional to N. The savings gained in practice are discussed in the next section.

To experiment with BIS, we generated 6 sorted files of uniformly distributed random integers between 0 and 2^{31}. 1,000 sets of batched records are also generated with integers uniformly distributed between 0 and 2^{31} with size k for $k = 1$ to 20.

Figure 4 shows the results of executing batched and unbatched interpolation search algorithms on a file of 400,000 uniformly distributed integers. The savings due to the batching of queries over the unbatched interpolation search are roughly 50%.

An Algorithm for Blocked Batched Interpolation Search (BBIS)

In this section, algorithm BIS is modified to take blocking into consideration. The idea is to decide whether we need to bring a new block into memory for each search item in the batch. The block access approximation for BBIS is performed by assuming that the chances of requiring a new block are proportional to the distance between two consecutive search steps. The analysis of BBIS has been experimentally validated.

3.3 Experimental Results

In this experiment, five sorted files of 400,000 integers uniformly distributed between 0 and 2^{31} were generated, each with a different blocking factor. A 1,000 sets of sorted records were also generated with size k for $k = 1$ to 20.

Figure 5 shows the savings of batched block accesses over unbatched block accesses in a file of 400,000 records with blocking factor of 100. Again, there is roughly 50% savings.

3.4 Summary

The basic Interpolation Search algorithm was extended to provide batched searching over blocked and non-blocked database environments. In [LW85] analytic expressions for the behavior of these extensions were developed. All expressions are validated by extensive experiments. In addition, algorithms for batched interpolation search over non-uniformly distributed ordered files are also developed and analysed.

4 Bit Transposed Files

4.1 Introduction and Motivation

Scientific and Statistical Databases (SSDBs) exhibit many specialized data usage and characteristics. Despite the development of many advanced access methods, the dominant file structure for very large SSDBs is still the simple sequential file. The major reason is a "mismatch" between conventional access methods such as inverted files, B-trees, hashing, etc. and the characteristics of SSDBs. First, since the cardinality of SSDBs attributes is typically small, most access methods simply partition the database into a small number of still very large files, with prohibitively expensive overhead for the pointers, structures, tables, etc., with only limited selective power added. Second, since SSDBs are largely static, the expensive overhead associated with the dynamic facilities of most access methods is not justified. Third, the values of SSDBs attributes tend to cluster, and current access methods often do not take advantage of this opportunity for compression. Fourth, the access to SSDBs is typically long "sweep" i.e., a long sequence of individual records is fetched and a small number of attributes extracted. This kind of range access is not supported well by most access methods.

The search for an appropriate file structure begins with the fourth point mentioned above, which is the motivation for the well-known attribute transposed files. Conventional files store the data as a collection of contiguous records, i.e., all the fields for a single record are stored together on a disk page. Attribute transposed files store the data as a collection of contiguous attribute columns, i.e., all of the data for a field (attribute) is stored together. Bit transposed files (BTF) store the data as a collection of bit columns, i.e., all of the data for a single bit position of an attribute encoding is stored together. Thus the file structure we propose can be seen to be an extreme form of the attribute transposed file.

The basic advantage of attribute transposed files is that only those attribute columns which are needed for a query need be retrieved. In many statistical applications only a small fraction of the attributes are needed for a query. Bit transposed files offer three advantages:

1. Clever data encodings will permit us to retrieve only a fraction of the bit vectors used to encode an attribute in order to perform a selection.

2. The bit vectors are amenable to data compression via run length encoding, especially if the data records have been sorted.

3. Selection criteria can be formulated as boolean expressions on the bit vectors, facilitating fast evaluation and specialized hardware.

In summary, the bit transposed file system offers an efficient means of performing selections.

4.2 Overview

The BTF system has three major components: an index encoder, transposed bit vector loader, and a query processor on bit vectors.

The index encoder translates each field in each record in the database into a series of bits based on several encoding schemes. The result is that each record of the database is translated into a bit pattern.

The second component, called the transposer, stores the bit patterns in a transposed manner so that for each bit position of the bit pattern, a file is produced which contains the bit value of that bit position from all the records in the database. The result is n BTFs where n is equal to the number of bit columns that result after encoding. Because values in large statistical databases tend to cluster, we have developed a compression method to compress the BTFs so that long runs of 0's and 1's can be stored more efficiently.

The third component of this file structure is the query processor on BTFs. The processor translates the retrieval requests on the database into a boolean expression on the BTFs. The translation algorithm takes as input the encoding schemes for the attributes and the query type of the query. The resultant boolean expression is evaluated by using the primitive boolean operators AND, OR, and NOT that can operate directly on compressed BTFs.

4.3 Results

A prototype system was implemented, which includes a bit compression package, a query language parser and evaluator, and an index encoding optimizer. There are four basic index encoding schemes and a composite scheme that allows designers of BTFs to combine the other encoding scheme. The index encoding optimizer accepts as input the usage statistics of the attributes and total size for the BTFs and generates the optimal index encoding schemes for each attribute. A dynamic programming technique is used to find successively larger subset of the attributes. The system was prototyped using a real database of 110,000 records. The result is a tenfold improvement of both time and space over typical commercial DBMSs such as Datatrieve.

There are several extensions to the BTF structure that are added since the publication of the BTF paper [WLO*85]. First, hierarchies (1-to-many relationships) are introduced between BTFs to better model the SSDB environment. An example of hierarchy in SSDBs is the hierarchical relationship between cities, counties, and states. An efficient file structure (called an association file) to support these kinds of hierarchical relationships has been designed and coupled with the bit transposed file structure. An user interface facility has also been built to make the BTF and association files into a self-contained system [WL86a].

Another extension to BTF is the idea we called "common subexpression removal". This idea uses compressed bit maps as efficient temporary storage for partial results. This approach saves the subset of data which is of interest to users. In turn this collection of bit maps can be used by other queries as an efficient access path. An algorithm has been developed that automatically incorporates these common subexpressions into the incoming queries to reduce the length of the boolean expressions and hence improve the efficiency of evaluating the queries [WLR86].

Another extension is the retrieval optimization. The problem here is that there could be a significant performance difference depending on the order of evaluating the boolean expression. The general solution of finding the optimal order has been shown to be NP-complete. Our approach is use heuristics that have been proven to find near optimal order for similar problems in order to evaluate the boolean expression on bit maps. The algorithm takes advantage of the different compression rates associated with the compressed bit maps as a guideline of choosing the order [LW86].

5 Multidimensional Partitioning Algorithms

5.1 Overview

The problem of multidimensional partitioning arises in many database applications where it is required to store files which are indexed by one or more search attributes on disk pages such that the mapping from the key space to the physical address space is order preserving. Many scientific applications which make use of spatial or temporal data require such files. Examples include results of hydrodynamics calculations and particle track data from high energy physics experiments. File structures which support such requirements are the Grid Files of different types and other order preserving file structures. In all of these methods the possible range of values for each attribute is partitioned into segments, the intersection of these segments define hyper-rectangles or cells. Each record is associated with a cell based on the segments to which its attribute values belong. Information about the partitioning is stored in a directory which is stored on disk or in fast storage depending on its size. Similar to multidimensional hashing methods, these methods allow retrieval of records in a fixed number of disk accesses. The advantage of this type of file organization over multidimensional hashing is that range queries can be processed efficiently as tuples with similar values on a search attribute tend to cluster together on the same page. Also periodic reports and tabulations which require that the data is sorted by one of the attributes can be readily obtained from the file without the need to sort the output. Such capabilities are very important in scientific and statistical data base management systems.

The performance of different partitioning algorithms can be measured in terms of retrieval time for different types of queries, the directory size, and the storage utilization. In this work, we study partitioning algorithms which associate every cell of the partitioning with a disk page of fixed capacity. There is a constraint on the total number of pages available for storing the database. In case the algorithm assigns to a cell more records than its capacity, overflow occurs and the overflow records are stored in a separate area. Therefore overflow records introduce additional retrieval cost and our goal is to find partitionings which minimize the amount of total overflow generated by the mapping.

5.2 Approach

The approach in grid file type organization, is to change the partitioning dynamically as the file grows while guaranteeing zero overflow. The insertion (or removal) of new partitioning lines, in order to refine the segmentation, can be quite costly in terms of pointer updates and other directory maintenance operations. In the case that the file is relatively static or that the growth structure is predictable, it may be beneficial to determine a "good" partitioning *in advance* in order to avoid frequent changes in the directory. This approach can also be used when converting an existing file into a dynamic grid file where a good initial partitioning is required, or when periodically reorganizing a grid file.

An example of this problem is shown in Figures 6 and 7, where a file with two attributes, each attribute having 5 possible values is given. The numbers in each cell indicate the number of records with specified attribute values. It is required to partition this file into 16 pages, where the capacity of each page is 3 records. As can be seen, the partitioning shown in Figure 7 has less total overflow records than the partitioning shown in Figure 6.

5.3 Results

We have developed dynamic programming algorithms for finding the optimal partitioning of a given multidimensional file. Our objective is to minimize

1	2*	4*	0	0
1	1	1	1	0
2	1	1	1	4*
1	1	1	1	2
1	1	2	1	1

Figure 6: Example of partitioning with c=3 and K=16. Total overflow is 5. Asterisks indicate cells with overflow.

1	2	4*	0	0
1	1	1	1*	0
2	1	1	1	4*
1	1	1	1	2
1	1	2	1	1

Figure 7: Another example of partitioning with c=3 and K=16. Total overflow is 3. Asterisks indicate cells with overflow.

the total overflow subject to constraints on page capacity, storage utilization and total number of pages available to accommodate the file. This approach is practical for small problems and it was tested extensively on inputs generated under different assumptions on the distribution of values in the file. We then tested three different heuristics with relatively fast running times for obtaining sub-optimal solutions. The idea behind the heuristics is to solve the partitioning problem separately on each dimension in an optimal way and then combine these solutions to yield an overall solution to the problem. The decision as to how many segments to allocate to each dimension is made in a different way in each one of the heuristics.

Extensive testing for large problems and comparisons with the optimal solution for smaller problems were made. The heuristics were shown experimentally to provide much better solutions than existing known heuristics. The results of this work were reported in [RS85].

In the future, we plan to look at flexible partitioning algorithms which divide the space of possible attribute values into rectangular regions not necessarily forming a grid. Initial experiments with such designs indicate that the total overflow can be reduced significantly by removing the grid restriction at the expense of having to store more information about the structure to allow efficient retrieval.

We plan to study other related optimization problems such as minimizing the maximum overflow from a page instead of the total overflow. The maximum overflow represents a worst case situation whereas the total overflow is a measure of average performance.

We also plan to investigate probabilistic algorithms which attempt to minimize the expected amount of overflow when some assumptions can be made about the distribution of attribute values. We think that this approach can lead to interesting results because such algorithms were successfully employed previously on related problems such as bin-packing. In that case, the expected performance of the probabilistic algorithm was shown to be very close to that of the optimal algorithm.

6 Temporal Data Structures

6.1 Problem description

In a previous section entitled Modeling Temporal Data we discussed the motivation for paying special attention to temporal data in scientific applications. In the following, we describe physical data structures and access methods that are needed to support the requirements of temporal data.

We use here the concepts of a *time sequence* (TS) and a *time sequence array* (TSA) as described in the section on modeling temporal data. Briefly, a TS is a sequence of data values in the time domain that are associated with a certain entity (e.g., a detector). We assume that each such entity has a unique identifier that we refer to as a *surrogate*. As was pointed out before, it is convenient to view the collection of TSs for all surrogates as a two dimensional array, called a TSA, where each row represents a TS. Our problem is to design data structures and access methods for the TSAs that are efficient for the expected access of such data. Obviously the two major concerns are efficient storage and efficient access time.

6.2 Approach and results

Our approach to the design of efficient physical support for temporal data is to take advantage, whenever possible, of the properties of temporal data so as to minimize the amount of storage used while maintaining reasonable access time. There are three such properties.

The first property that can be exploited is that temporal data are essentially "append only". Once they are collected, no updates or deletions are made, except to correct mistakes. Thus, the physical organization should mainly support retrieval operations well; update and delete operations need not be supported efficiently. This property suggests that we can store values consecutively in storage without concern to the reorganization that insert and delete operations could cause.

the second property is that TSs are often regular; i.e. the time points of the TS are equally spaced, and there exists a value associated with each time point. In such a case, it is not necessary to store the times associated with each value explicitly; rather it is sufficient to keep a description of the time sequence (start time, interval, end time). Regularity is a very important property for efficient storage utilization and access methods. The most storage efficient physical structure that can be hoped for is a structure that stores the surrogate values and the time values only once, rather than with each data value, i.e. both the surrogate and time values are "factored out". It is easy to see that when TSs are regular, the two dimensional array representation of a TSA can provide the desired storage efficiency since we can store the identifiers of the rows (i.e. the surrogates) and columns (i.e. the time points) only once.

The third property that can be taken advantage of is that temporal data is often static. By "static" we mean a data set that has been fully collected, and no more additions over the time dimension are expected. In many SSDB applications, such as running a physics experiment or collecting statistics on gasoline production, the entire set of time sequences are collected ahead of time, and then are subject to analysis. More efficient data structures can be designed for static TSAs because the data can be analyzed ahead of time for better storage utilization.

In addition to taking advantage of the properties of temporal data, we need to characterize the access patterns to the data to ensure that the physical structures are appropriate for the applications. We make several assumptions.

First, we assume that operations in the time domain often involve time ranges. Thus, storing the data values according to their time sequence order would cause physical clustering of the values, and minimize the number of pages (blocks) read from secondary storage for range queries in the time domain. Obviously, the order of values in TSs should be preserved.

Second, we assume that random access in the time domain is necessary since we want to support efficiently queries that select any time point (or time segment) of TSAs.

Third, we assume that random access of the surrogates is necessary since we want to support queries that request only a subset of the surrogates.

Fourth, we wish to provide the option that the surrogate domain is ordered. This assumption implies that physical clustering along the surrogate domain may be desirable for some applications.

In [SK86], we have designed several models for data structures and access methods of temporal data. The variations of the models depend on the properties discussed above, and on expectations of different access patterns. The three parameters that determine the most desirable design are: regular/irregular, static/dynamic, and random access requirements. All models use the basic structure of a two dimensional array representing the TSA, with variation to accommodate the parameters mentioned above. In general, the models for regular TSAs are more efficient than irregular TSAs (in terms of storage and access time). Some of these designs directly benefit from our work on multi-dimensional partitioning algorithms, and rearrangement for efficient compression discussed in other sections of this paper.

7 Current Work

The research on several of the topics discussed in this paper is still in progress. In temporal data modeling we are developing a syntax for operators over temporal data. Our research in sampling continues into weighted sampling and sampling from queries involving multiple relational operators. Our research into multi-dimensional data partitioning algorithms continues with investigations of non-grid partitionings, alternative optimization criteria (e.g. minimax overflow, average overflow). Our work on bit transposed files continues on optimal index encoding methods, and retrieval query optimization. Our research into data structures for temporal data still requires detailed analyses of the proposed data structures.

Acknowledgements

Some of the work described in this paper was a result of the efforts of persons no longer with the LBL data management research group: Paul Chan, Susan Eggers, Paula Hawthorn, Rowland Johnson, and Peter Kreps, and some of the work was done by other members of our department, e.g., John McCarthy. The authors also wish to thank Ann-Marie Soulsberg, for her efforts in typing portions of the bibliography.

References

[CS81] P. Chan and A. Shoshani. A directory driven system for organizing and accessing large statistical databases. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 553–563, 1981.

[DKO*84] David J. DeWitt, Randy H. Katz, Frank Olken, Leonard Shapiro, Michael R. Stonebraker, and David Wood. Implementation techniques for main memory databases. In *ACM SIGMOD International Conference on the Management of Data*, pages 1–8, ACM, Boston, 1984.

[DNSS83] D. Denning, W. Nicholson, G. Sande, and A. Shoshani. Research topics in statistical database management. In *Proceedings of the Second International Workshop on Statistical Database Management*, pages 46–51, September 1983.

[EOS81] S. J. Eggers, F. Olken, and A. Shoshani. A compression technique for large statistical databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 424–434, 1981.

[ES80] S. J. Eggers and A. Shoshani. Efficient access of compressed data. In *Proceedings of the 6th International Conference on Very Large Databases (VLDB)*, pages 205–211, 1980.

[Haw82] P. Hawthorn. Microprocessor assisted tuple access, decompression and assembly for statistical database systems. In *Proceedings of the International Conference on Very Large Data Bases, (VLDB)*, pages 223–233, September 1982.

[Joh81a] R.R. Johnson. A data model for integrating statistical interpretations. In *Proceedings of the First LBL Workshop on Statistical Database Management*, pages 176–189, December 1981.

[Joh81b] R.R. Johnson. Modelling summary data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 93–97, 1981.

[Kre82] P. Kreps. *A semantic core model for statistical and scientific databases.* Technical Report LBL-15393, Lawrence Berkeley Laboratory, 1982. (in LBL perspective on statistical database management).

[LW85] J. Li and H. Wong. *Further Results on Interpolation Searching of Database.* Technical Report LBL-20708, Lawrence Berkeley Laboratory, December 1985.

[LW86] J.Z. Li and H.K.T. Wong. *On Formal Properties of Bit Transposed Files.* Technical Report LBL-21281, Lawrence Berkeley Laboratory, 1986.

[McC82] J. McCarthy. Meta-data management for large statistical databases. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, 1982.

[Mer82] D. Merrill. Problems in spatial data analysis. In *Proceedings of the Seventh Annual SAS Users Group International Conference*, February 1982.

[Olk83] F. Olken. How baroque should a statistical database management system be? In *Proceedings of the Second International Workshop on Statistical Database Management*, September 1983.

[Olk86] F. Olken. Physical database support for scientific and statistical database management. In *Proceedings of the Third International Workshop on Statistical and Scientific Data Management*, July 1986. (also issued as LBL-19940rev.).

[OR86a] F. Olken and D. Rotem. Rearranging data to maximize the efficiency of compression. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, March 1986.

[OR86b] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proceedings of the International Conference on Very Large Databases*, VLDB Endowment, August 1986. condensed version of LBL-20707.

[OR86c] F. Olken and D. Rotem. *Simple Random Sampling from Relational Databases.* Technical Report LBL-20707, Lawrence Berkeley Laboratory, February 1986.

[RS85] D. Rotem and A. Segev. *Optimal and Heuristic Algorithms for Multi-Dimensional*

Partitioning. Technical Report LBL-20676, Lawrence Berkeley Laboratory, December 1985.

[Sho78] A. Shoshani. *CABLE: A Language Based on the Entity-Relationship Model.* Technical Report UCID-8005, Lawrence Berkeley Laboratory, 1978.

[Sho82] A. Shoshani. Statistical databases: characteristics, problems, and some solutions. In *Proceedings of the 8th International Conference on Very Large Data Bases (VLDB)*, pages 208–222, 1982.

[SK86] A. Shoshani and K. Kawagoe. Temporal data management. In *Proceedings of the International Conference on Very Large Databases*, VLDB Endowment, August 1986. (condensed version of LBL-21143).

[SM82] A. Shoshani and J. McCarthy. Physical database research at the lawrence berkeley laboratory. In *IEEE Database Engineering Newsletter*, March 1982.

[SOW84] A. Shoshani, F. Olken, and H.K.T. Wong. Characteristics of scientific databases. In *Proceedings of the 10th International Conference on Very Large Data Bases (VLDB)*, pages 147–160, 1984.

[SW85] A. Shoshani and H.K.T. Wong. Statistical and scientific database issues. *IEEE Transactions on Software Engineering*, SE-11(10):1040–1047, October 1985.

[WK82] H.K.T. Wong and I. Kuo. Guide: graphical user interface for database exploration. In *Proceedings of the International Conference on Very Large Data Bases, (VLDB)*, pages 22–32, September 1982.

[WL86a] H.K.T. Wong and J.Z. Li. *Hierarchical Bit Transposed Files.* Technical Report LBL-21284, Lawrence Berkeley Laboratory, 1986.

[WL86b] H.K.T. Wong and J.Z. Li. Transposition algorithms for very large compressed databases. In *Proceedings of the International Conference on Very Large Databases*, VLDB Endowment, August 1986. (condensed version of LBL-21157D).

[WLO*85] H.K.T. Wong, F. Liu, F. Olken, D. Rotem, and Wong L. Bit transposed files. In *Proceedings of the International Conference on Very Large Databases*, pages 448–457, August 1985.

[WLR86] H.K.T. Wong, J.Z. Li, and D. Rotem. *Common Subexpression Removal for for Bit Transposed Files.* Technical Report LBL-21283, Lawrence Berkeley Laboratory, 1986.

[Won84] H.K.T. Wong. Micro/macro statistical database management. In *The First International Conference on Data Engineering*, March 1984.