ISSN:0975-9646

An Effective Indexing Method for High Dimensional Databases

Manjunath K.G. , Kallinatha H.D.
S.I.T.Tumkur, India,

Abstract **– A large number of database applications like business data warehouses and scientific data repositories deal with high-dimensional data sets. As the number of dimensions/attributes and the overall size of data sets increase, it becomes prime important to efficiently retrieve specific queried data from the database in order to utilise the database effectively. Normally users are interested in querying data over a relatively small subset of the entire attribute set at a time. A potential solution is to use lower dimensional indexes that accurately represent the user access patterns. If the query pattern change, then the query response using the physical database design that is developed based on a static snapshot of the query workload may based on frequent item set mining ,which calculate the support and confidence used for high-dimensional data sets and to dynamically adjust indexes as underlying query workload changes. A query pattern change detection mechanism is used to determine when the access patterns have changed which will influence the change in the physical database design. Another possible solution would be to use some dimensionality reduction techniques, index the reduced dimension data space, and transform the query in the same way that the data was transformed. However, the dimensionality reduction approaches are mostly based on data statistics and perform poorly, especially when the data is not highly correlated. They also introduce a significant overhead in the processing of queries.**

One of the solutions is to apply feature selection to keep the most important attributes of the data according to some criteria and index the reduced dimensionality space. However, traditional feature selection techniques are based on selecting attributes that yield the best classification capabilities. Therefore, they also select attributes based on data statistics to support classification accuracy rather than focusing on the query performance and workload in a database domain. In addition, the selected features may offer little or no data pruning capability, given query attributes.

Keywords **— Indexing, KDD, OLAP, RAID, X-tree, GC-tree**

I. INTRODUCTION

Applications such as business data warehouses and scientific data repositories deal with high-dimensional data sets. Since the number of dimensions/attributes and the overall size of data sets are large, it becomes essential to efficiently retrieve specific queried data from the database in order to effectively utilize the database. Indexing support is needed to effectively prune out significant portions of the data set that are not relevant for the queries. Multidimensional indexing, dimensionality reduction, and Relational Database Management System (RDBMS) index selection tools all could be applied to the problem. However, for high-dimensional data sets, each of these potential solutions has inherent problems. An ideal solution would allow us to read from the disk only those pages that contain matching answers to the query. We could build a multidimensional index over the data set so that we can directly answer any query by only using the index. However, the performance of multidimensional index structures is subject to Bellman's curse of dimensionality and rapidly degrades as the number of dimensions increases. For the given example, such an index would perform much worse than a sequential scan. Another possibility would be to build an index over each single dimension. The effectiveness of this approach is limited to the amount of search space that can be pruned by a single dimension .Another possible solution would be to use some dimensionality reduction techniques, index the reduced dimension data space, and transform the query in the same way that the data was transformed. However, the dimensionality reduction approaches are mostly based on data statistics and perform poorly, especially when the data is not highly correlated. They also introduce a significant overhead in the processing of queries. Another possible solution is to apply feature selection to keep the most important attributes of the data according to some criteria and index the reduced dimensionality space. However, traditional feature selection techniques are based on selecting attributes that yield the best classification capabilities. Therefore, they also select attributes based on data statistics to support classification accuracy rather than focusing on the query performance and workload in a database domain. In addition, the selected features may offer little or no data pruning capability, given query attributes.

II. PRELIMINARIES

Data mining: (sometimes called data or knowledge discovery) is the process of analysing data from different perspectives and summarizing it into useful information that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analysing data. It allows users to analyse data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases. Data: Data are any facts, numbers, or text that can be processed by a computer. Today, organizations are accumulating vast and growing amounts of data in different formats and different databases. This includes: Operational or transactional data such as, sales, cost, inventory, payroll, and accounting, Non operational data, such as industry sales, forecast data, and macro economic data ,Metadata -Data about the data itself, such as logical database design or data dictionary definitions.

Information: The patterns, associations, or relationships among all this *data* can provide *information*. For example, analysis of retail point of sale transaction data can yield information on which products are selling.

Knowledge: Information can be converted into *knowledge* about historical patterns and future trends. For example, summary information on retail supermarket sales can be analysed in light of promotional efforts to provide knowledge of consumer buying behaviour. Thus, a manufacturer or retailer could determine which items are most susceptible to promotional efforts.

Data Warehouses: Dramatic advances in data capture, processing power, data transmission, and storage capabilities are enabling organizations to integrate their various databases into *data warehouses*. Data warehousing is defined as a process of centralized data management and retrieval. Data warehousing, like data mining, is a relatively new term although the concept itself has been around for years. Data warehousing represents an ideal vision of maintaining a central repository of all organizational data. Centralization of data is needed to maximize user access and analysis. Dramatic technological advances are making this vision a reality for many companies. And, equally dramatic advances in data analysis software are allowing users to access this data freely. The data analysis software is what supports data mining. Knowledge Discovery in Databases (KDD): KDD is the process of identifying, valid, potentially useful and ultimately understandable structure in data. This process involves selecting or sampling data from a data warehouse, cleaning or pre-processing it, transforming or reducing it, applying a data mining component to produce a structure and then evaluating the derived structure.

Architecture for Data Mining: To best apply these advanced techniques, they must be fully integrated with a data warehouse as well as flexible interactive business analysis tools. Many data mining tools currently operate outside of the warehouse, requiring extra steps for extracting, importing, and analysing the data. Furthermore, when new insights require operational implementation, integration with the warehouse simplifies the application of results from data mining. The resulting analytic data warehouse can be applied to improve business processes throughout the organization, in areas such as promotional campaign management, fraud detection, new product rollout, and so on. Figure illustrates architecture for advanced analysis in a large data warehouse.

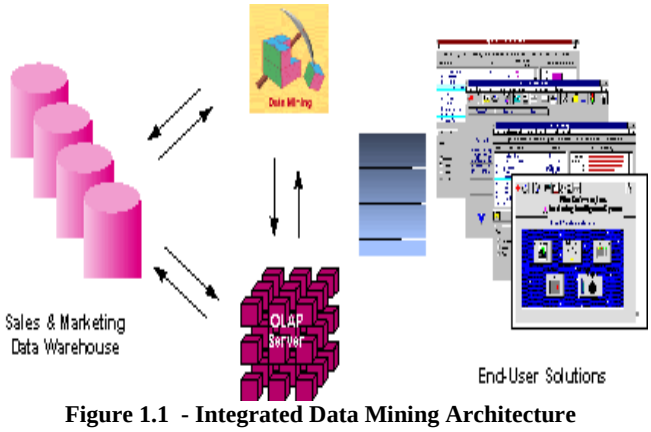


Figure 1.1 - Integrated Data Mining Architecture

III. RELATED WORKS

High Dimensional Indexing A number of techniques have been introduced to address the high-dimensional indexing problem such as the X-tree [2] and the GC-tree [4]. Although these index structures have been shown to increase the range of effective dimensionality, they still suffer performance degradation at higher index dimensionality.

A. EXTENDED TREE METHOD

The X-tree (extended node tree) is a new index structure supporting efficient query processing of high-dimensional data. The goal is to support both point data and extended spatial data and therefore, the X-tree uses the concept of overlapping regions. From the insight obtained in the previous section, it is clear that we have to avoid overlap in the directory in order to improve the indexing of high-dimensional data. The X-tree therefore avoids overlap whenever it is possible without allowing the tree to degenerate; otherwise, the X-tree uses extended variable size directory nodes, so-called super nodes. In addition to providing a directory organization which is suitable for high-dimensional data, the X-tree uses the available main memory more efficiently (in comparison to using a cache)[2].

The X-tree may be seen as a hybrid of a linear array-like and a hierarchical R-tree-like directory. It is well established that in low dimensions the most efficient organization of the directory is a hierarchical organization. The reason is that the selectivity in the directory is very high which means that, e.g. for point queries, the number of required page accesses directly corresponds to the height of the tree. This is only true if there is no overlap between directory rectangles which is the case for a low dimensionality. It is also reasonable, that for very high dimensionality a linear organization of the directory is more efficient. The reason is that due to the high overlap, most of the directory if not the whole directory has to be searched anyway. If the whole directory has to be searched, a linearly organized directory needs less space' and may be read much faster from disk than a block-wise reading of the directory. For medium dimensionality, an efficient organization of the directory would probably be partially hierarchical and partially linear. The problem is to dynamically organize the tree such that portions of the data which would produce high overlap are organized linearly and those which can be organized hierarchically without too much overlap are dynamically organized in a hierarchical form. The algorithms used in the X-tree are designed to automatically organize the directory as hierarchical as possible, resulting in a very efficient hybrid organization of the directory.

The research challenge which has led to the design of the GC-tree is to combine the capability of the vector approximation approach that accesses only a small fraction of real vectors with the advantage of the multidimensional index structure that prunes most of the search space and constructs the index dynamically. In order to achieve this goal, we partition the data space based on the analysis of the dataset and construct the hierarchical index that reflects the space partition hierarchy.

B. *GRID CELL TREE METHOD*

The GC-tree employs a *density-based* approach to partition the data space and to determine the number of bits to represent a cell vector for a partition. To approximate the density of the data points, the GC-tree partitions the data

space into non-overlapping hyper-square *cells* and find the points that lie inside each cell of the partition.

This is accomplished by partitioning every dimension into the same number of equal length intervals at a time. This means that every cell generated from the partition of a space has the same volume, and therefore the number of points inside a cell can be used to approximate the density of the cell [4].

In a static database, the *density* of a cell can be defined as the fraction of data points in the cell to the total data points. However, for a dynamic database environment, and especially, in the case of constructing a database from scratch, it is difficult to estimate the *density threshold* that identifies the dense and sparse cells because the density is relatively determined with respect to the total data points. Therefore, in the GC-tree, we define the *density* of a cell to be the proportion of data points in the cell to disk page capacity when we divide a space into 2^d cells by binary partitioning. A (sub)space in the data space corresponds to a *node* in the GC-tree and is physically mapped to a single *disk page*. There is a number *P* that identifies the maximum number of objects that can be accommodated in a disk page. That is, *P* represents the *page capacity* or the *fanout* of a page. When the number of objects inserted into a page exceeds *P*, the page is generally split into two. We call a cell c *dense* if the density of *c* is greater than or equal to a certain *density threshold* τ. Otherwise, it is called *sparse*. We call a dense cell a *cluster* and call the points that lie inside sparse cells *outliers*. If we determine the density threshold τ to be larger than a half of the page capacity, at most one cluster can be generated when we partition a space due to the insertion of an object.

The basic idea of the density-based partitioning is (1) to identify clusters and outliers when we partition a space, (2) to focus the partitioning on the subspaces of the clusters found because the subspaces covered by the outliers are unlikely to be pruned in the search, and (3) to deal together with all outliers found in the partitioning of a certain space.

It is difficult to bind the outliers within a small region since they are widely spread over the whole subspace. Thus it is very difficult to prune the outliers collectively during the search because the large $k\text{-}NN^{sphere}$ is likely to intersect the large bounding region in which the outliers lie. Therefore, we collect in a single node of the GC-tree all outliers generated from a single subspace partition, and concentrate the partitioning on the clusters to reduce the possibility that clusters are intersected by the search sphere $k\text{-}NN^{sphere}$. If the number of outliers generated from the partition exceeds the page capacity, the GC-tree allocates more pages for the outliers and simply links them. It makes multiple pages a single *virtual page*. This is based on the observation that the volume covered by the outliers is so large that it may not be pruned in the search.

It is well known that for low-dimensional indexes it is beneficial to partition the data space as *balanced* as possible. However, in high-dimensional spaces, the balanced partitioning results in large bounding rectangles for the partitions. When we apply balanced partitioning on a uniformly distributed dataset, the data space cannot be split in each dimension. For example, in a 256-dimensoinal data space, a split in each dimension results in a 2^{256} partitions (or disk pages). Therefore, the data space is usually split once in a number d' of dimensions. In the remaining (d - d') dimensions it has not been split and the bounding rectangles include almost the whole data space in these dimensions. Even for the non-uniformly distributed (e.g., clustered) dataset, the bounding rectangles are likely to be large because they still try to accommodate outliers and the outliers usually lie far apart. On the contrary, the GC-tree excludes the outliers in forming the bounding regions to reduce the size of the bounding regions.

C. INDEX SELECTION IN RELATIONAL-DATABASES

A problem of considerable interest in the physical design of databases is the selection of a good set of indices. Indices can be considered **as** auxiliary files that allow to retrieve tuples satisfying certain selection predicates without having to examine the whole relation. On the other hand, updating the database causes an index to be updated to remain consistent with the new database state. So, an index speeds up retrieval and slows down maintenance. In general two types of indices **can** be distinguished: primary and secondary indices. In the case of a primary index, the tuples in the relation are ordered on the indexed attribute. This is not the case for a secondary index [4].

PRIMARY AND SECONDARY INDICES

This section is devoted to the relation between a primary index and secondary indices. Indices are supposed to be organized often **as** B+-trees. Each node in the tree coincides with a page. The leaf level consists of {key, TID-list} **pairs** for every unique value of the indexed attribute(s). Figure 2.2(a) represents a primary index on the column ***name*** of a relation ***RP name, age, residence, blood group)*** and figure 2.2(b) a secondary index on the column ***blood group*** of R. In general the processing of a query roughly consists of two steps; first the number of tuples which satisfies possibly the **WHERE** clause of a query is determined; then these tuples are retrieved **.** Since a primary index may be considered **as** a special kind of a secondary index the optimizer may treat a primary index and secondary indices **as** same in processing the first step. In the second step it may use the ordening property of the primary index if at least both types of indices may be used. The following example illustrates this.

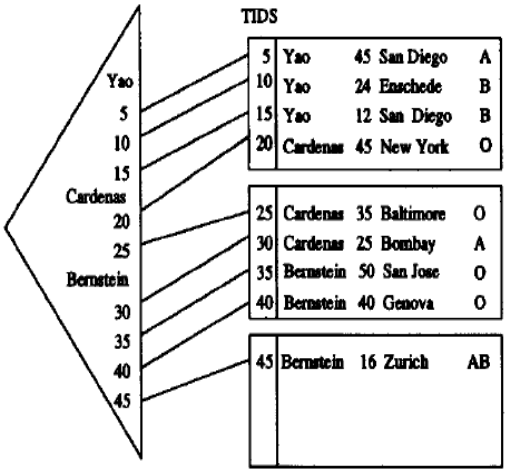


Fig 2.2 (a) Represents a primary index on attribute *name* of relation R.

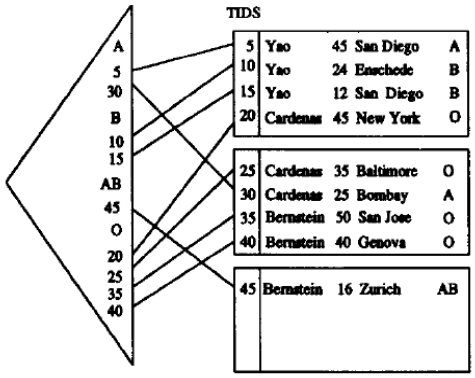


Fig 2.2 (b) Represents a secondary index on attribute *blood group* of R.

AUTOMATIC INDEX SELECTION

The ideas of having a database that can tune itself by automatically creating new indexes as the queries arrive have been proposed in[6].In[6] a cost model is used to identify beneficial indexes and decide when to create or drop an index at runtime. Costa and Lifschitz propose an agent-based database architecture to deal with an automatic index creation. Microsoft Research has proposed a physical-design alerter to identify when a modification to the physical design could result in improved performance.

ASSOCIATION RULE MINING

A main idea of association mining technique is to search a relationship of attributes and tuples, by discovering frequently occurring item sets in database. A result is patterns described as rules that represent one-way relationship. Furthermore, result rules consist of a confidential value and support value, a value of which is used to identify the pattern. The support is a number of instances that complies with the rules, whereas the confidential is a percentage of instances that must likewise be complied by rules. In basket analysis, for example, the association mining is a customer's behaviour analysis that determines the products the customer frequently buy together.

Association rule mining, one of the most important and well researched techniques of data mining, was first introduced in [1]. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control etc.

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those item sets whose occurrences exceed a predefined threshold in the database; those item sets are called frequent or large item sets. The second problem is to generate association rules from those large item sets with the constraints of minimal confidence.

In many cases, the algorithms generate an extremely large number of association rules, often in thousands or even millions. Further, the association rules are sometimes very large. It is nearly impossible for the end users to comprehend or validate such large number of complex association rules, thereby limiting the usefulness of the data mining results. Several strategies have been proposed to reduce the number of association rules, such as generating only "interesting" rules, generating only "non redundant" rules, or generating only those rules satisfying certain other criteria such as coverage, leverage, lift or strength [1].

In general, a set of items (such as the antecedent or the consequent of a rule) is called an item set. The number of items in an item set is called the length of an item set. Item sets of some length k are referred to as k-item sets. Generally, an association rules mining algorithm contains the following steps:• The set of candidate k-item sets is generated by 1-extensions of the large (k -1)-item sets generated in the previous iteration. Supports for the candidate k-item sets are generated by a pass over the database.• Item sets that do not have the minimum support are discarded and the remaining item sets are called large k-item sets. This process is repeated until no more large item sets are found.

IV. PROPOSED SYSTEM

Instead of using the query optimizer to estimate the query cost, we conservatively estimate the number of matches associated with using a given index by using a multidimensional histogram abstract representation of the data set. The histogram captures data correlations between only those attributes that could be represented in a selected index. The cost associated with an index is calculated based on the number of estimated matches derived from the histogram and the dimensionality of the index. Increasing the size of the multidimensional histogram enhances the accuracy of the estimate at the cost of an abstract representation size.

While maintaining the original query information for later use to determine the estimated query cost, we apply one abstraction to the query workload to convert each query into the set of attributes referenced in the query. We perform frequent item set mining over this abstraction and only consider those sets of attributes that meet a certain support to be potential indexes. By varying the support, we affect the speed of index selection and the ratio of queries that are covered by potential indexes. We further prune the analysis space using association rule mining by eliminating those subsets above a certain confidence threshold. Lowering the confidence threshold improves the analysis time by eliminating some lower dimensional indexes from consideration but can result in recommending indexes that cover a strict superset of the queried attributes.

Our technique differs from existing tools in the method that we use to determine the potential set of indexes to evaluate and in the quantization-based technique that we use to estimate query costs. All of the commercial index wizards work in design time. The DBA has to decide when to run this wizard and over which workload. The assumption is that the workload is going to remain static over time, and in case it changes, the DBA would collect the new workload and run the wizard again. The flexibility afforded by the abstract representation that we use allows it to be used for infrequent index selection considering a broader analysis space or frequent online index selection.

USER MODULE

Clients are end users of the systems. The user interface resides in these client systems. This user interface enables the clients to submit their item search. The user interface

also enables the users to Purchase the items from the seller. A client is allowed to login from the client system. Only authenticated clients are allowed to login. Password and username is used to authenticate the client (user).

ALGORITHM

1. Start the system.2. Open the UI to the system.3. If new user then submit request for creating a new user account.
4. If registered user, then provide login ID, password and select category as *user* to log in. If error displays the error message and go back to login prompt.5. Display list of items provided by the company.6. Accept the no of items requested
7. Create bill with tax 8. If purchase is completed by the user then provide options for a. To go back and purchase the items. b. To go back the main page. 9. End

INDEX SELECTION MODULE

We identify three major components in the index selection framework:

- The initialization of the abstract representations.
- The query cost computation, and
- The index selection loop.

In the following sections, we describe these components and the data flow between them.

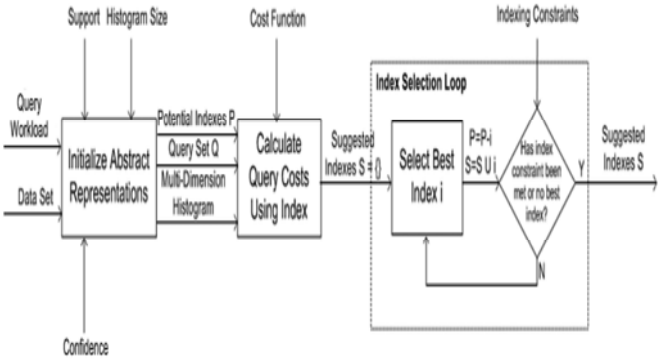


Fig 5.3 Index Selection Flow chart and its components.

INITIALIZE THE ABSTRACT RESENTATION

In this module we are monitoring the user queries and initialize the abstract representation**.** In this module we collection the user transaction and from that we are finding the frequently selected item. And by applying the association rule we are calculation the relationship between the records and finding the support and confidence. Based on that we are initializing the abstract representation. The initialization step uses a query workload and the data set to produce a set of Potential Indexes P, a Query Set Q, and a Multidimensional Histogram H according to the support, confidence, and histogram size specified by the user. The description of the outputs and how they are generated are given as follows:

Potential index set p: This is a collection of attribute sets that could be beneficial as an index for the queries in the input query workload. This set is computed using traditional data mining techniques. Considering the attributes involved in each query for the input query workload to be a single transaction, *P* consists of the sets of attributes that occur together in a query at a ratio greater than the input support. Formally, the support of a set of attributes *A* is defined as

$$S_A = \frac{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } A \subseteq Q_i \\ 0 & \text{otherwise} \end{cases}}{n},$$

Where *Qi* is the set of attributes in the *ith* query, and *n* is the number of queries.

For instance, if the input support is 10 percent and attributes 1 and 2 are queried together in greater than 10 percent of the queries, then a representation of the set of attributes {1, 2} will be included as a potential index. Note that because a subset of an attribute set that meets the support requirement will also necessarily meet the support, all subsets of attribute sets meeting the support will also be included as potential indexes (in the example above, both sets {1} and {2} will be included). As the input support is decreased, the number of potential indexes increases. Note that our particular system is built independently of a query optimizer, but the sets of attributes appearing in the predicates from a query optimizer log could just as easily be substituted for the query workload in this step. If a set occurs nearly as often as one of its subsets, an index built over the subset will likely not provide much benefit over the query workload if an index is built over the attributes in the set. Such an index will only be more effective in pruning data space for those queries that involve only the subset's attributes. In order to enhance analysis speed with limited effect on accuracy, the input confidence is used to prune the analysis space. Confidence is the ratio of a set's occurrence to the occurrence of a subset.

While data mining the frequent attribute sets in the query workload in determining *P*, we also maintain the association rules for disjoint subsets and compute the confidence of these association rules. The confidence of an association rule is defined as the ratio that the antecedent (left-hand side of the rule) and consequent (right-hand side of the rule) appear together in a query, given that the antecedent appears in the query. Formally, the confidence of an association rule {set of attributes A} -→ {set of attributes B}, where A and B are disjoint, is defined as

$$C_{A \to B} = \frac{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } (A \cup B) \subseteq Q_i \\ 0 & \text{otherwise} \end{cases}}{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } A \subseteq Q_i \\ 0 & \text{otherwise} \end{cases}},$$

In our example, if every time attribute 1 appears, attribute 2 also appears, then the confidence of {1}→{2}= 1.0. If attribute 2 appears without attribute 1 as many times as it appears with attribute 1, then the confidence {2}→{1}= 0.5. If we have set the confidence input to 0.6, then we will prune the attribute set {1} from P, but we will keep attribute set {2}.

We can also set the confidence level based on the attribute set cardinality. Since the cost of including extra attributes that are not useful for pruning increases with increased indexed dimensionality, we want to be more conservative with respect to pruning attribute subsets. The confidence could take on a value that is dependent on the set cardinality. Although the A priori algorithm was appropriate for the

relatively low attribute query sets in our domain, a more efficient algorithm such as the FP-Tree could be applied if the attribute sets associated with queries are too large for the A priori technique to be efficient. Although it is desirable to avoid examining a high-dimensional index set as a potential index, another possible solution in the case where a large number of attributes are frequent together would be to partition a large closed frequent item set into disjoint subsets for further examination. Techniques such as CLOSET could be used to arrive at the initial closed frequent item sets.

Query set Q: This is the abstract representation of the query workload. It is initialized by associating the potential indexes that could be beneficial for each query with that query. These are the indexes in the potential index set P that share at least one common attribute with the query. At the end of this step, each query has an identified set of possible indexes for that query.

MULTIDIMENSIONAL HISTOGRAM H: An abstract representation of the data set is created in order to estimate the query cost associated with using each query's possible indexes to answer that query. This representation is in the form of a multidimensional histogram H. A single bucket represents a unique bit representation across all the attributes represented in the histogram. The input histogram size dictates the number of bits used to represent each unique bucket in the histogram. These bits are designated to represent only the single attributes that met the input support in the input query workload. If a single attribute does not meet the support, then it cannot be part of an attribute set appearing in P. There is no reason to sacrifice data representation resolution for attributes that will not be evaluated. The number of bits that each of the represented attributes gets is proportional to the log of that attribute's support. This gives more resolution to those attributes that occur more frequently in the query workload.

Data for an attribute that has been assigned b bits is divided into 2b buckets. In order to handle data sets with uneven data distribution, we define the ranges of each bucket so that each bucket contains roughly the same number of points. The histogram is built by converting each record in the data etc to its representation in bucket numbers. As we process data rows, we only aggregate the count of rows with each unique bucket representation, because we are just interested in estimating the query cost. Note that the multidimensional histogram is based on a scalar quantize designed on data and access patterns, as opposed to just data in the traditional case. A higher accuracy in representation is achieved by using more bits to quantize the attributes that are more frequently queried.

Sample Dataset			
A_1	A_2	A_3	Encoding
2	5	5	0000
4	8	3	0110
1	4	3	0000
6	7	1	1010
3	2	2	0100
2	2	6	0001
5	6	5	1010
8	1	4	1100
3	8	7	0111
9	3	8	1101

Histogram	
Value	Count
00.0.0	2
00.0.1	1
01.0.0	1
01.1.0	1
01.1.1	1
10.1.0	2
11.0.0	1
11.0.1	1

Table 5.1 Histogram Example

For illustration, Table 5.1 shows a simple multidimensional histogram example. This histogram covers three attributes and uses 1 bit to quantize attributes 2 and 3, and 2 bits to quantize attribute 1, assuming that it is queried more frequently than the other attributes. In this example, for attributes 2 and 3, values from 1 to 5 quantize to 0, and values from 6 to 10 quantize to 1. For attribute 1, values 1 and 2 quantize to 00, 3 and 4 quantize to 01, 5, 6, and 7quantize to 10, and 8 and 9 quantize to 11. The **.**'s in the column "Value" denote attribute boundaries (that is, attribute 1 has 2 bits assigned to it).

Note that we do not maintain any entries in the histogram for bit representations that have no occurrences. Thus, we cannot have more histogram entries than records and will not suffer from exponentially increasing the number of potential multidimensional histogram buckets for high-dimensional histograms.

CALCULATE THE QUERY COST

The query cost will be calculated based on the potential index and Query Set. The query will be used to find the best index. We say the index is best one if it gives result for all the queries. Once generated, the abstract representations of the query set Q and the multidimensional histogram H are used to estimate the cost of answering each query by using all possible indexes for the query. For a given query-index pair, we aggregate the number of matches that we find in the multidimensional histogram by looking only at the attributes in the query that also occur in the index (bits associated with other attributes are considered to be don't cares in the query matching logic). To estimate the query cost, we then apply a cost function based on the number of matches that we obtain by using the index and the dimensionality of the index. At the end of this step, our abstract query set representation has estimated costs for each index that could improve the query cost. For each query in the query set representation, we also keep a current cost field, which we initialize to the cost of performing the query by using sequential scan. At this point, we also initialize an empty set of suggested indexes S.

Cost function: This is used to estimate the cost associated with using a certain index for a query. The cost function can be varied to accurately reflect a cost model for the database system. For example, one could apply a cost function that amortized the cost of loading an index over a certain number of queries or use a function tailored to the type of index that is used. Many cost functions have been proposed over the years. R-Tree, which is the index type used for this work.

Although these published cost estimates can be effective to estimate the number of page accesses associated with using a multidimensional index structure under certain conditions, they have certain characteristics that make them less than ideal for the given situation. Each of the cost estimates formulas require a range radius. Therefore, the formulas break down when assessing the cost of a query that is an exact match query in one or more of the query dimensions. These cost estimates also assume that data distribution is independent between attributes and that the data is uniformly distributed throughout the data space.

In order to overcome these limitations, we apply a cost estimate that is based on the actual matches that occur over

the multidimensional histogram over the attributes that form a potential index. The cost model for R-trees that we use in this work is given by

$$(d^{\lfloor d/2 \rfloor} * m),$$

Where *d* is the dimensionality of the index, and *m* is the number of matches returned for query matching attributes in the multidimensional histogram. Using actual matches eliminates the need for a range radius. It also ties the cost estimate to the actual data characteristics (that is, incorporates both data correlation between attributes and data distribution, whereas the published models will produce results that are dependent only on the range radius for a given index structure). The cost estimate provided is conservative in that it will provide a result that is at least as great as the actual number of matches in the database. By evaluating the number of matches over the set of attributes that match the query, the multidimensional subspace pruning that can be achieved using different index possibilities is taken into account. There is an additional cost associated with higher dimensionality indexes due to the greater number of overlaps of the hyperspaces within the index structure and additional cost f traversing the higher dimensional structure. A penalty is imposed on a potential index by the dimensionality term. Given equal ability to prune the space, a lower dimensional index will translate into a lower cost. The cost function could be more complicated in order to more accurately model query costs. It could model query cost with greater accuracy, for example, by crediting complete attribute coverage for coverage queries. It could also reflect the appropriate index structures used in the database system such as B+-trees. We used this particular cost model, because the index type was appropriate for our data and query sets.

INDEX SELECTION LOOP

After initializing the index selection data structures and updating estimated query costs for each potentially useful index for a query, we use a greedy algorithm that takes into account the indexes that were already selected to iteratively select indexes that would be appropriate for the given query workload and data set. For each index in the potential index set P, we traverse the queries in query set *Q* that could be improved by that index and accumulate the improvement associated with using that index for that query. The improvement for a given query-index pair is the difference between the cost for using the index and the query's current cost. If the index does not provide any positive benefit for the query, no improvement is accumulated. The potential index *i* that yields the highest improvement over the query set *Q* is considered to be the best index. Index *i* is removed from the potential index set *P* and is added to the suggested index set S. For the queries that benefit from *i,* the current query cost is replaced by the improved cost.

After each *i* is selected, a check is made to determine if the index selection loop should continue. The input indexing constraints provides one of the loop stop criteria. The indexing constraint could be any constraint such as the number of indexes, total index size, or the total number of dimensions indexed. If no potential index yields further improvement or the indexing constraints have been met, then the loop exits. The set of suggested indexes *S* contains the results of the index selection algorithm.

At the end of loop iteration, when possible, we prune the complexity of the abstract representations in order to make the analysis more efficient. This includes actions such as eliminating potential indexes that do not provide better cost estimates than the current cost for any query and pruning from consideration those queries whose best index is already a member of the set of suggested indexes. The overall speed of this algorithm is coupled with the number of potential indexes analyzed, so the analysis time can be reduced by increasing the support or decreasing the confidence.

Different strategies can be used in selecting the best index. The strategy provided assumes an indexing constraint based on the number of indexes and therefore uses the total benefit derived from the index as the measure of index "goodness." If the indexing constraint is based on the total index size, then the benefit per index size unit may be a more appropriate measure. However, this may result in recommending a lower dimensional index and, later in the algorithm, a higher dimensional index that always performs better. The recommendation set can be pruned in order to avoid recommending an index that is non useful in the context of the complete solution.

CALCULATE THE PERFORMANCE

For each response of the query we are calculating the Performance. Based on that performance the index modification will be performed.

ADMIN MODULE

Administrator is a part of the organization, so he should be allowed to see the sales of the items and administrator can also be a database administrator so he can see the details like support and confidence for different combinations of items that the end users of the system have purchased, which reflects the frequent pattern for our index calculations. The user interface provided is same as an end user but while logging in he has to specify as 'Admin' in category field of the Login panel. This user interface enables the Admin to submit their item search. Only authenticated users are allowed to login. Password and username is used to authenticate the user (Admin).

Algorithm:

1. Start the system
2. Open the UI to the system
3. If new user then submit request for creating a new user account.
4. If registered user, then provide login ID, password and select category as Admin to log in. If error display the error message and go back to login prompt.
5. Display list of items provided by the company and its sales.
6. Display list of different combinations of items with their support and confidence which shows the frequent items sales.
7. Display Indexes for Data retrieval.
8. If all pages are over then provide options for
   a. To go back and see details again.
   b. To go back to main page.
9. End.

V. IMPLEMENTATION

The J2EE platform is used for implementation. It uses a multi-tiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multi-tiered J2EE environment to which the application component belongs. Figure 6.1 shows two multi-tiered J2EE applications divided into the tiers described in the following list. The J2EE application parts shown in Figure 6.1 are presented in J2EE Components.

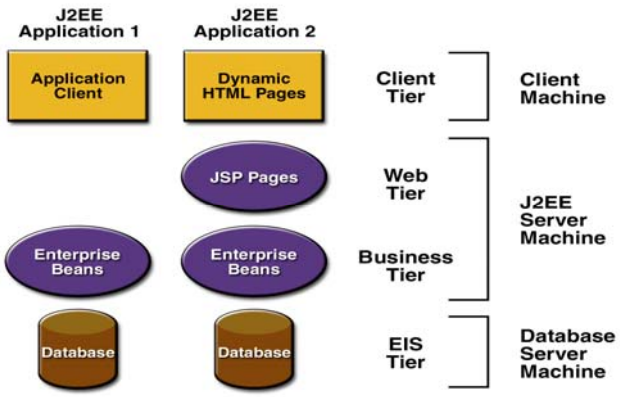


Figure 6.1 Multi-tiered Applications

- Client-tier components run on the client machine.
- Web-tier components run on the J2EE server.
- Business-tier components run on the J2EE server.

Enterprise information system (EIS)-tier software runs on the EIS server. Although a J2EE application can consist of the three or four tiers shown in Figure 6.1, J2EE multi-tiered applications are generally considered to be three-tiered applications because they are distributed over three different locations: client machines, the J2EE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and back-end storage.

J2EE COMPONENTS

J2EE applications are made up of components. A *J2EE component* is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components. The J2EE specification defines the following J2EE components: Application clients and applets are components that run on the client. Java Servlet and Java Server Pages™ (JSP™) technology components are Web components that run on the server.

Enterprise JavaBeans™ (EJB™) components (enterprise beans) are business components that run on the server.J2EE components are written in the Java programming language and are compiled in the same way as any program in the language.

The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

BUSINESS COMPONENTS

Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. Figure 6.3 shows how an enterprise bean receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it (if necessary), and sends it back to the client program.

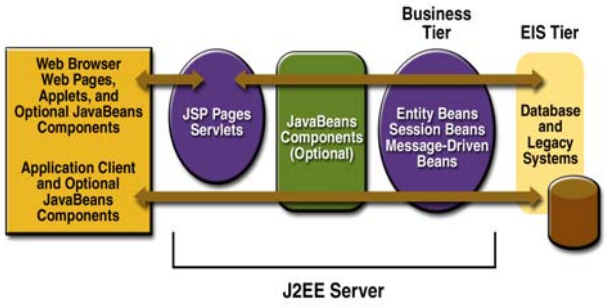


Figure 6.3 Business and EIS Tiers

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans. A *session bean* represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. In contrast, an *entity bean* represents persistent data stored in one row of a database table. If the client terminates or if the server shuts down, the underlying services ensure that the entity bean data is saved. A *message-driven bean* combines features of a session bean and a Java Message Service ("JMS") message listener, allowing a business component to receive JMS messages asynchronously. This tutorial describes entity beans and session beans.

MAJOR ACTIVITIES BY ALL MODULES

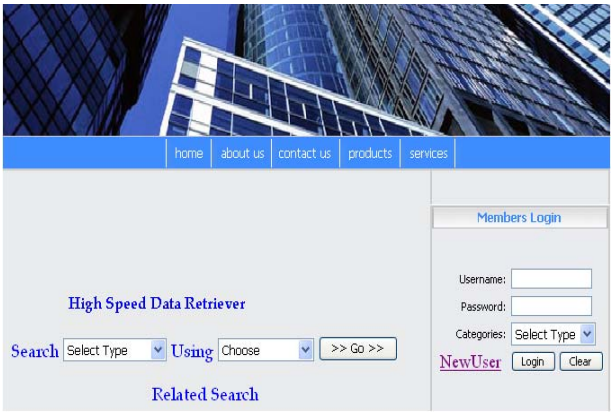
Following are the major activities implemented in the all modules of the project implemented.

MAJOR USER PROGRAM ACTIVITIES **are** Initialize components , Validate user ,Encode instruction commands , Encode data , Transmit instruction command and data , Create new user ,Change password , Display items options panel , Calculate bill ,Allow the user to

purchase , Acknowledge and exit. There are several sub activities in each of the above activities, for example validate user does the following activities. Display login panel, Verify client side validation, Verify user name field and password file for not empty , Send user name to the server, Get password from the server , Decode the password , Verify the password is correct or not. Major Admin Program Activities : It remains same technically that is the user interface provided is same as an end user but while logging in he has to specify as 'Admin' in category field of the Login panel. This user interface enables the Admin to submit their item search. Only authenticated users are allowed to login. Password and username is used to authenticate the user (Admin).It displays the item wise sales details, frequent item sets, support and confidence for different combinations of items (Index).

Major Index selection Program Activities: After every 5 transactions create index , Before creating index find frequent item set, Calculate support and confidence for different combination of items, When a new query arrives try to find index with less query cost , Display that as the answer for the search, Otherwise display nothing.

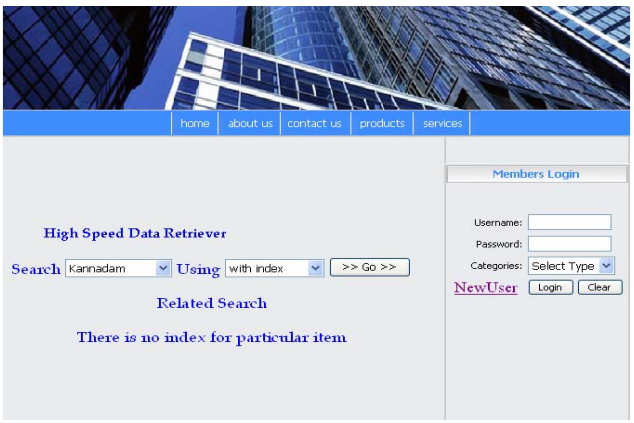
VI. SCREEN SHOTS



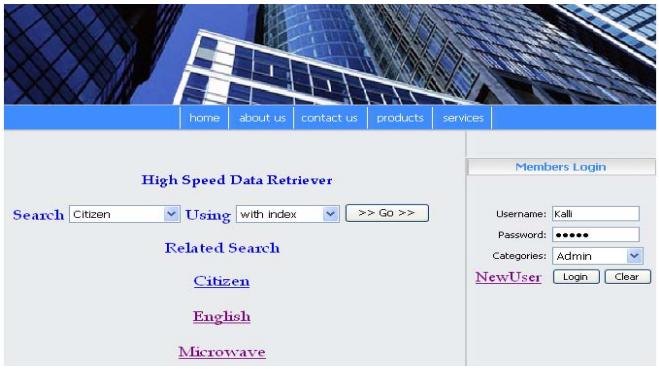
Screen Shot 1 showing the Starting Screen the System with Login Panel and Search box.

Screen shot 2 showing the new member Login Registration forms for User and Admin.

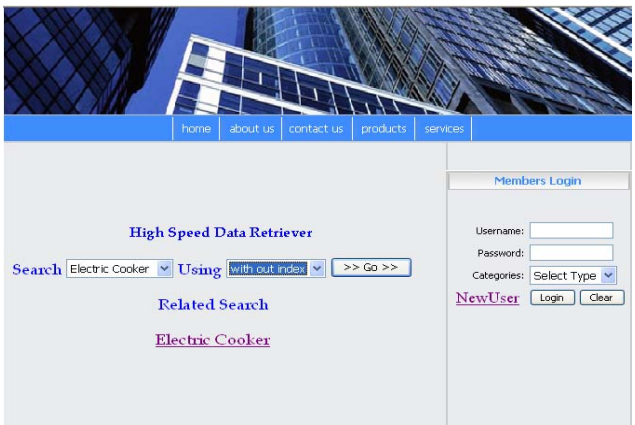
Screen shot 3 showing the 'With Index' method of Index recommendations.



Screen shot 4 showing the 'With Index' method of recommendations when there is no recommended index.



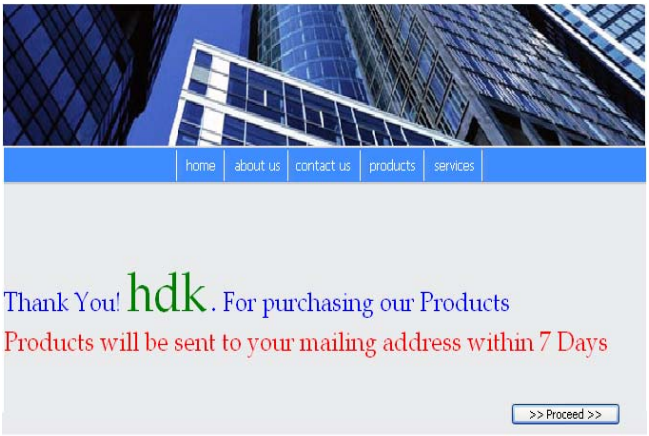
Screen shot 5 showing the without index method.



Screen shot 6 showing the Admin Login panel.

Screen shot 7 showing the List of items for Purchase and command to purchase.

Screen shot 8 showing the List of items Purchased and total Bill.



Screen shot 9 Acknowledging the User Purchase.

The description of the various forms (snap shots)of the project implemented is as follows: Screen Shot 1 shows Main menu of the system with Login Panel and Search box. Screen shot 2 shows the new member Login Registration forms for User and Admin. Screen shot 3 shows the 'With Index' method of Index recommendations. Screen shot 4 shows the 'With Index' method of recommendations when there is no recommended index. Screen shot 5 shows the without index method. Screen shot 6 shows the Admin Login panel. Screen shot 7 shows the List of items for Purchase and command to purchase. Screen shot 8 shows the List of items Purchased and total Bill. Screen shot 9 shows form acknowledging the User Purchase.

VII. RESULTS AND DISCUSSIONS

We have used J2EE to implement Effective Indexing method for High dimensional Databases .The Index recommendations are done by using frequent item set mining. Test results show that whenever a query pattern change, this intern changes the frequent item set then the support and confidence values and change in item set change the recommended indexes and this makes the system more attractive. However extra bit of time is needed to these activities such as calculating frequent item set, confidence, support and query cost. Therefore the index recommendations based on frequent item set mining is appropriate for the product purchasing sites. If want to get a particular item in the system then better to choose the without index method available for search .In our model we have used a new data mining technique which is visible to users. The foundation provided here will be used to explore this trade-off and to develop an improved utility for real-world applications. The proposed technique affords the opportunity to adjust indexes to new query patterns. A limitation of the proposed approach is that if index set changes are not responsive enough to query pattern changes, then the control feedback may not affect positive system changes.

VIII. CONCLUSION AND SCOPE FOR FUTURE WORK

A flexible technique for index selection is introduced, which can be tuned to achieve different levels of constraints and analysis complexity. A low-constraint more complex analysis can lead to more accurate index selection over stable query patterns. A more constrained less complex analysis is more appropriate to adapt index selection to account for evolving query patterns. The technique uses a generated multidimensional histogram to estimate cost and, as a result, is not coupled to the idiosyncrasies of a query optimizer, which may not be able to take advantage of knowledge about correlations between attributes. Indexes are recommended in order to take advantage of multidimensional subspace pruning when it is beneficial to do so.

The proposed technique affords the opportunity to adjust indexes to new query patterns. A limitation of the proposed approach is that if index set changes are not responsive enough to query pattern changes, then the control feedback may not affect positive system changes. However, this can be addressed by adjusting control sensitivity or by changing control sensitivity over time as more knowledge is gathered about the query patterns.

Index creation is quite time consuming. It is not feasible to perform real-time analysis of incoming queries and generate new indexes when the patterns change. Potential indexes could be generated prior to receiving new queries and, when indicated by the analysis, moved to the active status. This could mean moving an index from the local storage to the main memory or from a remote storage to the local storage, depending on the size of the index.

Future Enhancement:

Future work is to reduce the no of computation while computing the index.

REFERENCES

[1] "Data mining", Arun K.pujari ,
[2] "The X-Tree: An Index Structure for High-Dimensional Data," S. Berchtold, D. Keim, and H. Kriegel , Proc. 22nd Int'l Conf. Very Large Data Bases (VLDB '96), pp. 28-39, 1996.
[3] "The GC-Tree: A High-Dimensional Index Structure for Similarity Search in Image Databases," C.-W. Chung and G.-H. Cha, IEEE Trans. Multimedia, vol. 4, no. 2, pp. 235-247, June 2002.
[4] "Index Selection in Relational Databases," K. Whang, Proc. Second Int'l Conf. Foundations on Data Organization (FODO '85), 1985.
[5] "Mining Frequent Patterns without Candidate Generation", J. Han, J. Pei, and Y. Yin, Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '00), W. Chen, J. Naughton and P.A. Bernstein, eds., pp. 1-12, 2000.
[6] "online index recommendations for high dimensional data bases using Query work load", Michael Gibas, Guadalupe Canahuate, and Hakan Ferhatosmanoglu, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 20, NO. 2, FEBRUARY 2008
[7] An efficient high-dimensional indexing method for content-based retrieval in large image databases I. Daoudi a,c, , K.Idrissi a, S.E.Ouatik b, A.Baskurt a, D.Aboutajdine c, Article history: Received29September2008 Received in revised form 28 August2009 Accepted4September2009.
[8] "Application of Web usage mining and product taxonomy to collaborative recommendations in e-commerce" , Yoon Ho Choa, Jae Kyeong Kimb,* ,Elsever publication .
[9] "hybrid data mining technique for knowledge discovery from engineering materials data sets ", Doreswamy, Hemanth K S , International Journal of Database Management Systems (IJDMS), Vol.3, No.1, February 2011.
[10] S. Kai-Uwe, E. Schallehn, and I. Geist, "Autonomous query-driven index tuning," in International Database Engineering & Applications Symposium, Coimbra, Portugal, 2004.
[11] R. L. D. C. Costa and S. Lifschitz, "Index self-tuning with agentbased databases," in XXVIII Latin-American Conference on Informatics (CLIE), Montevideo, Uruguay, 2002.
[12] N. Bruno and S. Chaudhuri, "To tune or not to tune? a lightweight physical design alerter." in VLDB, 2006, pp. 499–510.
[13] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in 2000 ACM SIGMOD Intl. Conference on Management of Data, W. Chen, J. Naughton, and P. A.Bernstein, Eds. ACM Press, 05 2000, pp. 1–12. [Online]. Available:citeseer.ist.psu.edu/han99mining.html
[14] J. Pei, J. Han, and R. Mao, "CLOSET: An efficient algorithm for mining frequent closed itemsets," in ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp.21–30. [Online]. Available: citeseer.ist.psu.edu/pei00closet.html
[15] C. Faloutsos, T. Sellis, and N. Roussopoulos, "Analysis of object oriented spatial access methods," in SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data. New York, NY, USA: ACM Press, 1987, pp. 426–439.
[16] C. Bohm, "A cost model for query processing in high dimensional data spaces," ACM Trans. Database Syst., vol. 25, no. 2, pp. 129–178, 2000.