PHYSICAL DATA WAREHOUSE DESIGN USING NEURAL NETWORK

Mayank Sharma
Galgotias College of Engineering & Technology
Greater Noida(NCR) India

Navin Rajpal
University School of Information Technology
GGS Indraprastha University Delhi

B.V.R.Reddy
University School of Information Technology
GGS Indraprastha University Delhi

ABSTRACT
Performance of the data warehouse depends on physical design. Index selection and storage of multidimensional data bases are important activities of physical designing process. Conventional indexing techniques such as bitmaps, B-trees and hash based indexing systems need large storage space for storing indexes along with data itself. Spelling variants, misspellings and transliteration differences are source of uncertainty in data with in the databases. Misspelled and distorted key values are also hard to map in present indexing systems. In this paper neural network based physical design is suggested, a class of artificial neural network known as self-organizing net is used for indexing data warehouse at physical level. Indexes of active neurons will be used for generating indexes for the data values. In conventional indexing techniques every key value is mapped to a specific point in space, while in neural network based database indexing system, every key value is mapped to a region in space. This region is a class to which the key values of similar type belong. Indexes generated through this method used optimal space for storage, as only final weight matrices after training of neurons are stored. Self-organizing net based indexing is very robust as distorted key values get indexed to right classes. Accuracy of our self-organizing net based indexing system in mapping key values with distorted keys is found to be high.

Keywords
Self-organizing net, multidimensional databases, indexing.

1. INTRODUCTION
Data warehouses stores historical data to support decision making process. Data warehouses are several orders of magnitude larger than the operational data- bases. Data warehouses and OLAP systems are based on a multidimensional model. The multidimensional model views data in an n-dimensional space, usually called a data cube. Data cubes are constructed around quantitative values called measure values, which are needed to analyze from various perspectives called dimensions [14, 15, 21, 22, 29, 30, 31]. Multidimensional model represented by $DC(D_1, D_2, ... D_n; M_1, M_2, ... M_m)$, where in data cube DC there are *n* dimensions and *m* measure values. A cell $c = (d_{1,i}, d_{2,i}, ... d_{n,i}, m)$ where *m* is measure value stored in cell defined by dimensional values as $d_{j,i}$ i.e. i^{th} value of j^{th} dimension[21,29]. Cell in multidimensional space represents a tuple.

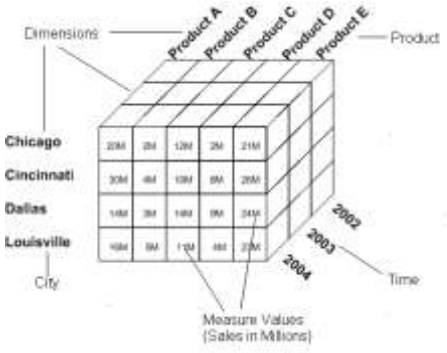

Figure 1. Pictorial representation of sales data cube

The data cube in the figure 1 is representing sales data cube with three dimensions "City", "Product" and "Time" and measure value in the cells are sales figure in millions taken for analysis of sales of products in the given cities for three years. Each measure value in cell is identified by the instances of the dimensions used to define that particular value. For instance sales value for product "Product A" at city "Dallas" in year "2004" is 14M (14 million) [14, 15, 22, 27, 30, 31]. Data cubes are constructive blocks for data warehouses. Implementation of data warehouse follows both logical and physical design considerations [22].

1.1. Logical data warehouse design
There are two approaches for implementing a multidimensional model, depending on the way in which data cube is stored. Traditionally, OLAP system has been built on top of a relational database system, therefore, referred as relational OLAP (ROLAP), where data is stored in tables. Relational OLAP model consists of a fact table and dimensional tables for each dimension. These dimension tables are referenced using foreign keys in fact tables [16, 17, 30]. Fact tables are made around some subject such as sales, purchase etc, for which data warehouse is defined. Alternatively, multidimensional OLAP (MOLAP), systems use multidimensional arrays as multidimensional data structures. In MOLAP, only measure values are stored and dimension values are treated as the indexes of the multidimensional arrays. Data is stored in multidimensional structures, which is a more natural way to express the multidimensionality of enterprise data. MOLAP system is faster and required less space as compared to ROLAP systems [14, 15, 16, 17, 21, 22, 29, 31]. Fast access to the data from OLAP system is primary goal. For analysis OLAP data cubes often use pre-computed aggregates at various levels of detail using various combinations of attributes [2, 21, 29]. Data cube will be stored in memory in the form of pre-computed aggregates called cuboids.

Various algorithms are present for data cube computation [2, 21, 29, 30], which are successfully implemented. In this paper, we use multi-way array cubing algorithm for computation of aggregated cube which are stored in sparse matrices. The multi-way algorithm is effective when the products of the cardinalities of the dimensions are moderate [30]. If A, B, C, and D are the dimensions then total 2^4=16 aggregate cuboids are possible. Taking ABCD as base cuboid figure 2 shows and aggregate at dimension B that results in cuboid ACD used to compute AD, which in turn can be used to compute A. This shared computation allows multi-way to perform aggregations simultaneously on multiple dimensions. These cuboids get physically stored in the memory for aggregate queries.

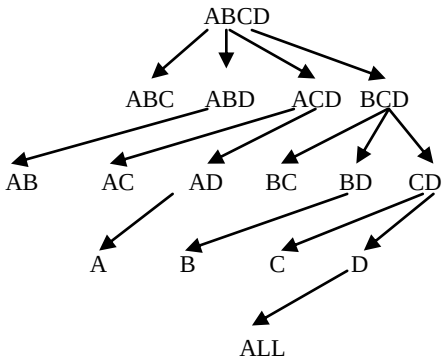

Figure 2. Top-down Cuboids computation

1.2 Physical data warehouse design

Performance of the multidimensional databases depends on the logical and physical design of these multidimensional databases [3, 6, 7, 14, 17, 27, 30]. Physical level designs of the data warehouse deals with the database partitioning materialized views, indexing and clustering of records [3, 7, 22]. Indexing at physical layer is used to improve the performance of logical layer [27]. Data warehouse are read-only data for complex multidimensional queries. Indexing on single attribute is not a solution for indexing multidimensional data [14, 15, 22, 27, 30, 31]. Conventional indexing techniques use data structures for indexing multidimensional data cubes, which consumes large amount of storage space for storing indexes apart from data itself. Some literature shows use of B-tree for indexing individual attributes rather than single key attribute for indexing multidimensional databases at physical layer [31]. Bit encoded sparse structure, are used for multidimensional indexing [15]. In Bit-encoded sparse structure, chunking of the multidimensional array based data cube takes place. For each cell in a chunk, a dimension index is encoded in bits for each dimension. Due to this data cube get compressed [15]. Bitmap indexing is a kind of indexing system in which a bit vector is associated with each value of column on which indexing is done. Bitmaps are useful for columns that have low cardinality [22], therefore, some products follow a hybrid approach. In a hybrid approach, a B-tree is used when the list of qualifying RIDs per entry are small, otherwise a bit-mapped index is used. Dynamic bit-maps are also used to handle high cardinality data and large range queries to construct the bit-maps dynamically from vertical partitioned fact table respectively [27]. High performance data warehousing techniques use parallel algorithms for implementing parallel indexing system to increase the performance of the analytical processing systems. Parallel and scalable infrastructure for multidimensional array based data cube is used for increasing the performance of the system [14, 15]. These indexing systems are not robust as distorted key values, noise and misspelled keys are not indexed properly. Storing indexes along with data consumes large memory.

Uncertainty in the data requires attention. Spelling variants, misspellings, and transliteration differences are source of uncertainty in data with in the databases. Conventional indexing techniques fail to map the misspelled and destroyed key values [13, 31]. Performances of these indexing methods are better for lower dimensional datasets and for online transaction processing systems. However, they could not provide adequate results in online analytical processing systems. These data structures needed space not less than the data itself [1, 4, 5, 7, 9, 12, 18, 19, 20, 23, 28]. Multidimensional database systems need an indexing system that stores indexes in less space and takes lesser time to generate multidimensional indexes. In database, individual indexes generated for each record or cluster index is generated for a group of records. Indexing is done on records using values of key attributes [1, 9]. Indexes are also needed to be stored in secondary storage devices. High performance string hash function is proposed in which each character of text requires only an exclusive-OR operation and an indexed memory read [25]. Hashing string type value and alphanumeric keys is solved by using classes of string hash function [26]. Problem of mapping misspelled words to memory was not solved using class of string hash functions. B-tree index need four times more space than the storage of corresponding database on which indexes are created. Dynamic hashing techniques are developed by combining them with the binary tree, which are better than static hashing. Their performance is better than the separate chaining method and memory requirement is also under control [23]. Hashing provides fastest possible access for retrieving an arbitrary record given the value of its hash field. However, collision management degrades the overall performance [22]. Hash based indexing method is hard to implement as maintaining large number of hash function for all dimensions is not possible. Some improvements are done through the use of compressed bitmap indexing of high dimensional data [18]. Many design techniques use horizontal table partitioning, materialized query table, and multidimensional clustering for fast query processing [6]. DB2 physical design advisor uses horizontal partitioning for distributing data on non-shared parallel machines and multidimensional clustering for data cube construction [3, 7]. Data structures such as SR-trees, skeleton R-trees and skeleton SR-trees are constructed by combining memory resident data structure (segment trees) and disk oriented indexing structures (R-trees) to provide data structure for multidimensional data sets. This approach is beneficial for indexing multidimensional databases and spatial data [19].

Clustering techniques in data mining are used to group data with similar properties. These clustering techniques solve nearest-neighbor queries. Data with similar properties get collected around data center of different clusters. Uncertainty in the data requires attention. Spelling variants, misspellings, and transliteration differences are source of uncertainty in data within the databases. Clustering of data values which are approximately similar to each other is a solution for finding similar values. On this basis, new method of approximation of similar strings is done. Clustering techniques are applied on strings for grouping them

together in files [13]. Different statistical and non-statistical clustering techniques are also available. Probability based statistical model for indexing, also called as density based indexing (DBIN), use probability density method to drive indexes over the data table. DBIN shows problem of numerical instability while indexing databases with high cardinality attributes. But an indexing system based on clustering technique is very useful in solving the problem of uncertainty in large databases. In this indexing system, search time reduces due to partitioning of the database into small clusters [5].

In this paper, self-organizing net is used for indexing data warehouses at physical level. According to theoretical and empirical justifications, neural network based methods are better than the statistical methods. Many self-organizing net like MAXNET, ART1, ART2 and kohonen network are available [24]. In this paper, self-organizing net describe by Yoh-Han Pao in taken into consideration due to its less complex implementation. It supports unsupervised learning based on discovery of clusters [24]. This neural network works as ART1 but also shows features of kohonen. Data taken into consideration consists of three dimensions and total six thousand five hundred records. Self-organizing net is used for clustering data values that have least Euclidean distance with weight values assigned to the neurons. Index value assigned to the data value, is equal to the index of the active neuron. In self-organizing net based indexing method, no extra efforts are needed to solve the problem of overlapped indexes. Indexes generated through this method use optimal space for storage, as only final weight matrices after training of neurons are stored for each dimension. Indexes are regenerated using saved weights, in less time. Single self-organizing net is used for indexing the data for all the dimensions. While in hashing different or multiple hash functions may be used. These indexes are used for the construction of MOLAP data cube. Self-organizing net based indexing is very robust as distorted key values also get indexed to the right class. Sparse structures are used to store these data cubes.

Organization of the paper is as follows: Section 2 describes neural network based indexing method with example. In Section 3, result of experiment is shown using precision-recall graph, which shows the performance of self-organizing net. Finally, in Section 4, we have given conclusion of the paper.

2. Self-Organizing Net for physical layer design of Data warehouse

In multidimensional database, attributes are referred as dimensions and records consist of values for each dimension [10]. Number of different values in a dimension is the cardinality of the dimension. Construction of multidimensional data cube is always better for fast data access in OLAP system. A cell in multidimensional space represents a tuple, with the attributes or dimensions values of the tuple identifying the location of the tuple in the multidimensional space and the measure values represent the content of the cell [13, 15, 23]. Indexing of the multidimensional database is done by indexing each data value of every dimension. Data values of each dimension have an index number, which is the index of its position in the multidimensional array taken for construction of the data cube. Data set taken in our indexing process consists of export data set providing information about the amount of sales of commodities to hundred countries in last eleven years. Data set consists of three dimensions as "Country_name" with cardinality 100, "Commodities" with cardinality 101, "Year" with cardinality 11 and the fact value attribute "Amount_of_Sales". Total records are about 6437. Sample of the fact table is shown in Table 1.

Table 1. Sample of data taken into consideration (Source: website, www.dgft.delhi.nic.in)

Country_Name	Commodities	Year	Amount_of_Sales
China	Pearls	1996-1997	74.99
China	woven pile fabrics	1997-1998	3.82
Russia	Pearls	1999-2000	1748.54
Nepal	Pearls	2000-2001	50.17
China	Beverages	2001-2002	127.46
Nepal	Beverages	2002-2003	1093.38
Russia	Beverages	2003-2004	70.09
China	chemical compounds	2005-2006	2372.57
Nepal	chemical compounds	2006-2007	1744.87

In conventional indexing system for indexing on each dimension, the dimension key values are used. Each key value of the dimension is stored in data structure used by the indexing system. In B-trees and their variants, key values are stored in the leaf nodes that point the key values to their indexes in multidimensional array [31]. Thus, for each dimension a separate data structure is used for storing indexes, which consumes separate memory. For example, to know the amount of sales of beverages in China in the year 2001-2002, we need the index number of "China", "beverages" and year "2001-2002" of the first, second and third dimension of multidimensional array based data cube and access the fact value 127.46. In this paper, indexing of multidimensional data cube is shown through neural network techniques. Indexes for all dimension values are generated and stored in sparse matrix. The sparse matrix defines data cube for data set.

2.1. Our approach for multidimensional database indexing

Indexing of the multidimensional database is done by indexing each data values of each dimension. Each dimension of the data cube will have size equal to the cardinality of the dimension. To achieve this, each unique value of the dimension should have unique index number on its dimensional axis. Value in the cell, will directly be accessed by its position in multidimensional array, will be calculated using indexes for the data values on the axis. Indexes for data values of each dimension can be generated using clustering technique applied separately on each dimension. The similar values of each dimension are grouped together in a class or cluster, with a cluster number assigned to it. This cluster

number will be the index of all values of the cluster. The cluster index number of a data value in a dimension will be its index on dimensional axis belonging to that dimension. For example clustering is done on the dimension "Country_name" and its data value "china" is assigned to cluster having index 1. Then "china" is having index 1 on dimension axis defined for dimension "Country_name". Similarly clustering is done on other dimension data values of second dimension "Commodities" and data value "PEARLS" is assigned to cluster with index number 3. Then "PEARLS" is having index 3 on dimension axis define for this dimension. Suppose the cluster index for "1996-1997" is 4 then its index on dimension axis will be 4. The first tuple ("china", "PEARLS", "1996-1997", 74.96) will be represented in data cube as a cell containing 74.96 having indexes (1,3,4) in three dimensional array. System does not have a prior knowledge of number of classes or clusters in which data will be classified. There is a need of an algorithm which performs clustering operation and gives indexes to each key value of the dimension taken for indexing. In this paper clustering technique is applied on data values of each dimension for grouping similar data values together and generates a class index number for it.

2.2. Neural network based indexing of multidimensional databases

While indexing a multidimensional data base, each dimension is taken into the indexing process separately. Clustering is needed to perform on the data values so that similar data values are grouped together into different classes. Cardinality of the dimension is not fixed. So, number of classes can not be known previously. Due to unsupervised learning, clustering is best suited for indexing of dimensional data values. Neural network architecture can be employed for both supervised and unsupervised learning paradigms. Type of neural network architecture used for clustering of input patterns depends on the characteristic of given data set. Looking at the problem of indexing multidimensional data bases, the data set taken for clustering are the data values of each dimension. Input patterns for the neural network are n-dimensional feature vectors which describe each data value of a dimension. Construction of input data is described later. Outputs of the neural network are indexes of the classes in which all the patterns are classified. Since the cardinality of the dimension is not fixed, data in data warehouse are updated after a certain time interval. Its size increases exponentially. New data values are added to each dimension. In conventional indexing system no extra efforts are required for adding new data values. So, new system of indexing should have this facility. Clustering can be performed using neural networks.

Input to the self-organizing net is feature vector which define each value of a dimension. Output is the index of cluster to which data values of the dimension belongs. The algorithm followed by this network to make clusters of data values have following steps.

1. This self-organizing net creates a node and assign first pattern as weight of node.
2. Calculating Euclidean distances of all patterns with weight assign to this newly created cluster. Pattern belong to cluster will have at minimum Euclidean distance from that cluster.
3. A threshold value is taken as vigilance parameter. If square root of minimum Euclidean distance for a pattern is less than and equal to threshold then pattern will belong to cluster and its number will be the index of key value defined by this pattern and weights of this node get updated.
4. If square root of minimum Euclidean distance for a pattern is more than threshold then a new cluster node is created with pattern which does not belong to earlier created clusters. Repeat steps 2 to 4.
5. Finally after training updated weights are saved for a particular dimension.

This self-organizing net gives unique index of each key value in the dimension so that indexes do not overlapped. In case a new data value is added to the dimension then mapping of this data value is done with only following the steps 2 through 4. If no category exits, a new cluster is made for new data value, this mechanism is beneficial while indexing data is data bases. The weights saved for each dimensions will work both for data accessing and adding new data values to the respective or new clusters. This discovery of clusters based unsupervised net is well suited for indexing data cube. In Kohonen's network, any new pattern that does not belong to the clusters already constructed is still forced into one of them using the best match strategy, without taking into account how good even the best match is. This way overlapped data values in the same cluster will affect the quality of indexing process [24]. This problem of stability of weights as well the inability to accommodate patterns belonging to new data values, can be solved using neural nets based on adaptive resonance theory (ART). Our algorithm is based on the principle of ART network, also, shows the feature of kohonen network. It also preserves topology. Number of clusters generated equal to the cardinality of the dimension. Input features vectors are constructed for each dimensional data. This is the input to the self-organizing map used for indexing each dimensional value.

2.3 Input feature vector construction

For the input to self-organizing net, input matrix is defined, which consists of vectors of parameter values defined for each value of the dimension. Values of the dimension are chosen for preparing input parameters. Each character of the key value is assigned a numeric value from the character reference table of the particular dimension. These values are used to define parameter vector for the key value. Character reference table of the dimension is prepared using histogram of the frequency of occurrence of each character in the values of dimension. The numeric value assign to a particular character in the dimension character reference table will be higher for the character with lowest frequency. Sample of dimension character reference table for first, second and third dimension are as shown in the following Tables:

Table 2 Dimension First Character Reference Table

Char	Frequency	Value
a	7776	0.01
i	4428	0.02
n	4332	0.03
s	3384	0.04
e	3032.	0.05
r	2381	0.06
…..	…….	…..
….	….	….
q	30	26

Table 3 Dimension Second Character Reference Table

Char	Frequency	Value
a	12789	0.01
b	12515	0.02
s	11159	0.03
t	10110	0.04
r	9775	0.05
…	…….	……
…..	…….	…..
j	21	0.26

Table 4 Dimension Third Character Reference Table

Char	Frequency	Value
0	8768	0.01
9	5272	0.02
2	4677	0.03
1	2930	0.04
6	1173	0.05
3	588	0.06
8	587	0.07
7	584	0.08
5	583	0.09
4	582	0.1

The above tables are used to define the parameters of each key value of the first, second and third dimension. Each value of dimension is defined by the numeric values of each character of key value taken from the dimension character reference table. These values define a key value of the dimension, which are, considered as parameters for the key value taken into consideration. Length of the input vector is equal to the length of longest key in that dimension. Sample input table of first dimension is shown in the Table 5. Length of the input vector for the first dimension is 16, which are according the length of longest key value of the first dimensions. Similarly input tables of other dimensions are defined for their values.

Table 5 Input Table For First Dimension

x1	**X2**	**…..**	**x13**	**x14**	**x15**	**X16**	**Value**
0.01	0.24	…..	0	0	0	0	afghanistan
0.01	0.07	…..	0	0	0	0	albania
0.09	0.01	…..	0	0	0	0	cape verde
0.11	0.10	…..	0	0	0	0	philippines
0.08	0.03	…..	0.12	0.15	0	0	united kingdom

These parameters are available as a input to self-organizing net. Giving input to the self-organizing net, training starts and weights of the neighboring neurons of winning neuron gets updated for each input. After completion of training and testing of self-organizing net, final weights for each dimension are stored. These weights will be used to calculate indices back at the time of accessing the data. The index numbers of the active neurons for each value of the dimensions are collected. These are the indices of the values. All the values belonging to a cluster will have same cluster number. Using the self-organizing net, for indexing each value of the dimension, no collision between the index values takes place, and the indexes generated during the training are in sequence as the values in the dimension. The index number for each value of the dimension will be considered while construction of MOLAP cube. This will also be optimal for storage purpose.

2.4 Data cube construction using sparse matrices

After scanning the fact table record-by-record and getting indexes for each value of the tuple, fact values are inserted at the position described by the index of each value of tuple. As multidimensional data cubes are sparse, sparse matrices are used to store index values of the nonzero cell entries. Defining a sparse structure, which have three variables for storing indexes of three dimensional values and the fourth variable for storing corresponding values. One instance of sparse structure is used to store one tuple. So, array of structure of size equal to the number of records of fact table is defined. Scanning each tuple of fact table and considering the value of each dimension to store its measure value into the sparse structure along with the indexes. Indexes of the dimensional values are generated using the weight matrixes of the dimensions. The weight matrixes of each dimension are stored and having final weight value of each node of self-organizing net after the training is completed. To get index of any value of dimension, an input vector is created using numerical values of characters of each dimensional value from the dimensional character reference table. Secondly Euclidean distance of this vector is calculated from each node of self-organizing net using stored weight matrix of that particular dimension. The cluster index number generated, will be the index of that dimensional value. Index value for each value of the tuple is calculated and stored in the sparse structure along with it measure value. This way whole fact table is stored into the array of sparse structure. Sample of the data cube stored in the sparse structure is shown in the figure 3.

Country_Names	Commodities	Year	value
1	1	1	8579.65
1	1	2	0.0
1	1	3	5.07
1	1	4	7.19
1	1	5	12.17
1	1	6	15.31
1	2	5	7724.16
1	2	6	2580.03
1	2	7	2670.2
1	2	8	11321.22
1	2	9	571.42
1	2	10	6804.55
1	3	8	19496.47
1	3	10	36597.24

Figure 3 The data cube in sparse structure

This figure shows that in the data cube each value of the dimensions have an index number along the three dimensional axes of the cube corresponding to the tuple of the fact table. Each row of the data cube is one instance of the sparse structure containing one tuple. The total instances are equal in the number as the total number of records in the fact table. In the multidimensional array based data cube each instance of the sparse data cube represents the one cell containing value of tuple. The purpose of taking each tuple in one instance of sparse structure is concerned with memory management as number of tuple increases then it will help in partitioning of the fact table stored in the sparse structures. The above data cube contains all the dimensions. High storage capacity is required to store the whole data cube. Decision support systems frequently pre-compute many aggregates to improve the response time of aggregation queries. Data cube can computes aggregates along all possible combinations of dimensions called group-bys. Fast data accessing system need data cube to be stored in from of group-bys. 2^N-1 aggregates calculations are needed for N-dimensional data cube. In above case there are three dimensions and one measure value so 2^3=8 group-bys are calculated as {Country_Name, Commodities, Year}, {Country_Name, Commodities}, {Country_Name, Year}, {Commodities, Year}, {Country_Name},{Commodities}, {Year} and ALL. Collection of all the group-bys is called data cube. There are two ways to from group-bys formally before construction of sparse cube. The fact table can be partitioned and all the view of the fact dimension can be stored into sparse structures. This method is very costly as memory requirement is high. Secondly sparse data cube formed from original fact table can be use for aggregation along dimensions and group-bys formed will be stored in the sparse structure again [14,15]. Any group-by will be computed by selecting the smallest of the previously computed group-bys as its parent. In above data cube the cardinalities of the dimensions are 100,101 and 11 form Country_Name , Commodities and Year respectively. The original data cube will be used for calculating group-bys {Country_Name, Commodities}, {Country_Name, Year}, and {Commodities, Year} from these group-bys the group-bys {Country_Name}, {Commodities} and {Year} are computed using {Country_Name,Year} and {Commodities, Year} as they are smallest parents for these aggregates or group-bys. Similarly supper aggregate ALL is calculated from parent group-by {Year}. These group-bys are stored in sparse structures and used for aggregation queries which does not need to access data from original data cube this make it fast in accessing the data. Storing these group-bys also require less memory.

3. EXPERIMENT AND RESULTS

Three-dimensional dataset taken for experiment is a fact table with dimensionality 3 with one fact value attribute. Cardinality of the three dimensions is 100, 101, and 11 respectively. Number of records is 6437. For this experiment, program is implemented using matlab6.0. Self-organizing net consists of variable number of neurons for each dimension. Single self-organizing net is used for each dimension. The indexes of the active neuron generated. These indices of active neurons are the cluster numbers for each value in a dimension. Final weights for neurons are stored for each dimension. These weights are used to regenerate the indices of data values at the time of testing, data accessing and for inserting new values. For the experimental verification cluster files for each dimension contain all the data values along with the index number generated for the value in the dimension. The index number of the data value will be the index of that value in the multidimensional array based data cube for the dimension to which it belongs. For the training of the self-organizing net for each dimension total 6437 input patterns are taken while testing 1713 input patterns are taken with distorted data values. After training the updated weights for the particular dimension are saved. For the testing of the classifier the test data set with distorted data values of the particular dimension is taken for input to the self-organizing net with the weight matrix which is saved for that dimension. As a result most of the distorted data values are correctly recognized and mapped to their respective clusters.

For performance analysis of the self-organizing neural net a matching matrix is constructed to represent the result of the test. This matching matrix is between the actual and predicted classes also called n-dimension confusion matrix [8, 10, 11]. In this matrix the diagonal elements consists of truly positive and truly negative values. The row elements except the diagonal element for each class are the false positive values for a particular class. The column elements except the diagonal element for each class are the false negative values for a particular class [10, 11]. From this n-dimension matching matrix, binary confusion matrices are constructed for each class with respect to other classes. So, there will be N binary confusion matrices for N classes. The entries of binary confusion matrix for a class is done by summing up of false positive, false negative, and true negative values with reference to this class.

After training for the data values of first dimension, test data values with distorted data values is given as input to the self-organizing net. The results of the classification of data into classes are compared using matching matrix and construction of binary confusion matrix for each class. Figure 4 show the sample of matching matrix and binary confusion matrix for first class.

		True Classes	afghanistan	albania	algeria	andorra	armenia	australia	...	ukraine	...	zambia
		Index Number	1	2	3	4	5	6	...	93	...	100
Predicted classes	Index Number	Total values										
afghanistan	1	10	9							1		
albania	2	12		11						1		
algeria	3	10			7		3			3		
andorra	4	9				5	3					
armenia	5	10					8					
australia	6	11					2	7				
...	...	...										
ukraine	93	15								15		
...	...	...										
zambia	100	21										13

Figure 4(a). Matching matrix for 100 classes

Accuracy= (sum of diagonal elements/total elements of matrix) x 100 = 92.3977.

	Actual Classes	
Predictive Classes	TP=9	FP=1
	FN=0	TN=1571
Column Total	P=9	N=1572

TP/P=Recall=1 Precision=TP/(TP+FP)=0.9

F-score=Precision x Recall=0.9

Figure 4(b) Binary Confusion matrix for single class afghanistan

In the above figure complete matching matrix shows the distribution of total 1713 patterns among the 100 classes. For example 10 patterns of afghanistan then matrix show 9 are correctly classified while 1 pattern is misclassified as ukraine. Form the matrix performance of the classifier for each class can be calculated and represented in a binary confusion matrix. For example binary confusion matrix is shown for first class afghanistan and also value of the performance parameters is calculated for this class. Now precision and recall values of all the classes are calculated and graph between these values is plotted as shown in the figure 5.

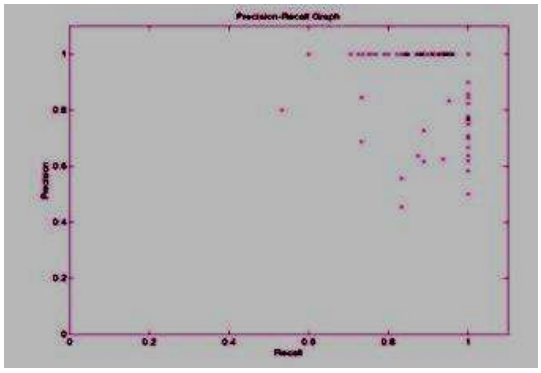

Figure 5 Precision-Recall Graph

The graph shows that large numbers of data values are nicely classified as many points in the graph have high precision and recall values. If the values of the precision are high for constant value of recall and vise-versa then the Precision-Recall graph is of best quality. The above graph shows that a break -even point at (1,1) this also show that the classifier performance is good for the test data. Average F-score for all the classes is above 80%. This also justifies the higher performance of classifier. In conventional indexing techniques every key value is mapped to a specific point in space, while in self-organizing net based data base indexing system, every key value is mapped to a region in the space. This region is a class to which the key values of similar type belong. The class index number is the index for all data key values belong to that class. In self-organizing map based indexing method no extra efforts are needed to solve the problem of overlapped indices, as there is least possibility of collisions because every key value belong to independent class so this way it gives better response than hash technique. In hashing collisions occurs very frequently more than one hash function is used for indexing a key to avoid collision. Also distorted key value cannot be mapped using hashing or B-trees. In our method of indexing single self-organizing map is used for indexing all the data values of the dimensions. Self-organizing maps show robustness in the indexing as they map the distorted keys to the right class. This is not possible with other indexing systems.

Further work is possible on the robustness of this system of indexing as in some cases if the value of the lost character, is maximum in the character reference table then generating the right index of that key value is not possible. This limitation needs to be further studied. Indexes generated through this method were found to use optimal space for storage, as only final weight matrices after training of neurons are stored. Space to store the final weight is less than the storage needed to store hash tables. Multidimensional data get compressed because self-organizing map transforms an incoming n-dimensional input data into one or two dimension discrete map. Regenerating the indexes at the time of data cube construction at conceptual layer is quite easy as compare to hashing. This system will perform better than hash based indexing system with respect to space and robustness.

4. CONCLUSION

Performance of data warehouse depends on its physical design. Indexing of data cubes plays major role in performance enhancement. In our work we support a neural network based indexing technique for indexing the multidimensional data cube. Self-organizing net can be used for data indexing in the multidimensional database that is better technique for indexing all the dimension values as compared to the traditional methods like Bit-map based indexing. As in data warehouses queries are based on read only system, no frequent updates are required in data warehouses. In conventional indexing techniques every key value is mapped to a specific point in space, while in self-organizing net based database indexing system, every key value is mapped to a region in space. This region is a class to which the key values of similar type belong. The class index number is the index for all data key values of the class. In self-organizing net based indexing method, no extra efforts are needed to solve the problem of overlapped indexes. Indexes generated through this method used optimal space for storage, as only final weight matrices after training of neurons are stored. Self-organizing net based indexing is very robust as distorted key values get indexed to right classes. It will perform well on a single or multi-computer system.

5. REFERENCES

[1] Abraham Silberschatz, Henry F. Korth, and Sudarshan (2002). Database System Concepts (pp 445-489). 4th Edition, McGraw Hill.

[2] Agarwal Sameet, Agarwal Rakesh, Deshpande Prasad M. Gupta Ashish, Naughton Jeffrey F., Ramakrishnan Raghu, Sarawagi, Suntia (1996). On the Computation of multidimensional Aggregates, Proc. 22nd VLDB Conf. Mumbai, India,1996.

[3] Agrawal, S., Narasayya V., and Yang, B (2004). Integrating vertical and horizontal partitioning into automated physical database design. In Proc. ACM SIGMOD international Conference on Management of Data (Paris, France, June 13 - 18, 2004). SIGMOD '04. ACM, New York, NY, 359-370.

[4] Almasi S. George, Lawreence Douglas, and Rushmeier Edith (2001) Scalable Parallel algorithm for self-organizing maps with applications to sparse data-mining problems, United States Patent, Patent No. US 6,260,036 B1 July 10, 2001.

[5] Bennett, K. P., Fayyad, U., and Geiger, D (1999). Density-based indexing for approximate nearest-neighbor queries, In Proceedings of the Fifth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (San Diego, California, United States, August 15 - 18, 1999). KDD '99. ACM, New York, NY, 233-243.

[6] Dainel C. Zilio, Jun Rao San Lightstone, Guy Lohman, Adam Strom, Christian Garcia-Arellano, and Scott Fadden (2004), Recommending Materialized views and indexes with IBM's DB2 Design Advisor, International Conference on Autonomic Computing 2004.

[7] Daniel C. Zilio, Jun Rao, San Lightstone, Guy Lohman, Adam Strom, Christian Garcia-Arellano, and Scott Fadden (2004). DB2 Design Advisor :Integrated Automatic Physical Database Design, Proc. 30th VLDB Conf. Toronto, Canada, 2004, pp. 1087-1097.

[8] Davis, J. and Goadrich, M.(2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international Conference on Machine Learning,* (Pittsburgh, Pennsylvania, June 25 - 29, 2006), ICML '06, vol. 148. ACM, New York, NY, 233-240.
DOI=http://doi.acm.org/10.1145/1143844.1143874

[9] Elmasri Ramez, Somayajulu V. L. N. Durvansula, Navathe B. Shamkant, and Gupta K. Shyam (2007). Fundamentals of database systems(pp 297-356). 3rd edition: Pearson Education.

[10] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recogn. Lett.* 27, 8 (Jun. 2006), 861-874. DOI= http://dx.doi.org/10.1016/j.patrec.2005.10.010

[11] Fawcett, T. (2003). ROC graphs: Notes and practical considerations for data mining researchers, Tech report HPL-2003-4. HP Laboratories, Palo Alto, CA, USA. Available:http://www.purl.org/net/tfawcett/papers/HPL-2003-4.pdf.

[12] Freeston, M(1995). A general solution of the n-dimensional B-tree problem, In Proceedings of the 1995 ACM SIGMOD international Conference on Management of Data (San Jose, California, United States, May 22 - 25, 1995). M. Carey and D. Schneider, Eds. SIGMOD '95. ACM, New York, NY, 80-91.

[13] French J.C., Powell, A. L., and Schulman, E.(1997). Applications of approximate word matching in information retieval. In Proceeding of the sixth international conference on information and knowledge Management, Las Vegas, Nevada, US, November 10-14,1997, CIKM'97. ACM. New York, NY, 9-15.

[14] Goil, S. and Choudhary, A. 1997. High Performance OLAP and Data Mining on Parallel Computers. *Data Min. Knowl. Discov.* 1, 4 (Dec. 1997), 391-417. DOI= http://dx.doi.org/10.1023/A:1009777418785

[15] Goil Sanjay, Choudhary Alok(1996). Design and Implementation of a scalable parallel system for multidimensional analysis and OLAP, 13th Int'l symposium on parallel and distributed processing.

[16] Gray J., Reuter A., Layman A., and Pirahesh H.(1996). Data cube: A relational aggregation operator generalizing group-by, cross-tabs, and sub-totals. In Proc. of the 12th Int'l Conference on Data Engineering, pp 152-159.

[17] Harinarayan, V., Rajaraman, A., and Ullman, J. D. (1996). Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international Conference on Management of Data* (Montreal, Quebec, Canada, June 04 - 06, 1996). SIGMOD'96. ACM, NewYork, NY, 205216. DOI=http://doi.acm.org/10.1145/233269.2333333

[18] Kesheng Wu, Ekow Otoo, and Arie Shoshani(2004). On the performace of bitmap indices for high cardinality attributes, Proceedings of the 30th VLDB Conf. Toronto, Canada, 2004 pp. 24-35.

[19] Kolovson, C. P. and Stonebraker, M. 1991. Segment indexes: dynamic indexing techniques for multi-dimensional interval data. *SIGMOD Rec.* 20, 2 (Apr. 1991), 138-147. DOI= http://doi.acm.org/10.1145/119995.115807

[20] Lanka, S. and Mays, E. 1991. Fully persistent B+-trees. *SIGMOD Rec.* 20, 2 (Apr. 1991), 426-435. DOI= http://doi.acm.org/10.1145/119995.115861

[21] Li Jianzhong, Srivastava Jaideep(2002), Efficient Aggregation Algorithms for Compressed Data Warehouses, IEEE Trans. Knowledge and data engineering, Vol. 14. No.3, pp 515-529.

[22] Malinowski, E. and Zimnyi, E.(2008). Advanced Data Warehouse Design: from Conventional to Spatial and Temporal Applications (Data-Centric Systems and Applications).1st ed.2008., pp 51-55,Springer Publishing Company, ISBN: 978-3-540-74404-7.

[23] Md. Mehedi Masud, Gopal Chandra Das, Md. Anisur Rahman, and Arunashis Ghose(2006). A Hasing Technique Using Separate Binary Tree", Data Science Journal, Volume 5, 19, October 2006, pp 143-161.

[24] Pao Y.H.(1989). Adaptive Pattern Recognition and Neural Networks,Addison-Wesley, Reading, MA, 1989.

[25] Pearson, P. K. 1990. Fast hashing of variable-length text strings. *Commun. ACM* 33, 6 (Jun. 1990), 677-680. DOI= http://doi.acm.org/10.1145/78973.78978

[26] Ramakrisha M.V., Justin, Zobel (1997).,"Performance in Practice of String Hashing Functions", Proceedings of the fifth International Conference on Database Systems for Advanced Applications, Melbourane, Australia, April 1-4, 1997.

[27] Sarawagi S.(1997). Indexing OLAP data, IEEE Data Engineering Bulletin, March.

[28] Seeger, B. and Larson, P. 1991. Multi-disk B-trees. *SIGMOD Rec.* 20, 2 (Apr. 1991), 436-445. DOI= http://doi.acm.org/10.1145/119995.115862

[29] Xin Dong, Han Jiawei, Li Xiaolei, Shao Zheng, and Wah. Benjamin W.(2007). Computing Iceberg Cubes by Top-Down and Bottom-Up Integration : The StarCubing Approach, IEEE Trans. Knowledge and data engineering, Vol. 19. No.1, pp 111-126.

[30] Zhao Y., Deshpande P., and Naughton J.(1997). An array-based algorithm for simultaneous multi-dimensional aggregates. In Proc. ACM-SIGMOD International Conferences on Management of Data, pp 159-170.

[31] Zhao Yihong, Tufte Kristin, Naughton F Jeffrey (1996), On the Performance of an Array-based ADT for OLAP workloads, Technical Report CS-TR-96-1313, University of Wisconsin-Madison, CS Department, May, 1996.