SPIN: A MULTIDIMENSIONAL INDEX STRUCTURE FOR INFORMATION STORAGE AND RETRIEVAL

By

YING FAN

Bachelor of Science
FuDan University
Shanghai, China
1986

Master of Science
FuDan University
Shanghai, China
1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1996

SPIN: A MULTIDIMENSIONAL INDEX
STRUCTURE FOR
INFORMATION STORAGE AND RETRIEVAL

Thesis Approved:

[illegible]
Thesis Advisor

J Chandler

Blayne E. Mayfield

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Hedrick for his intelligent supervision, constructive guidance, inspiration and advice. I would also like to thank Dr. Mayfield and Dr. Chandler for serving on my graduate committee.

My sincere appreciation extends to those who provided suggestions and assistance for this study: Mr. Zhang, Yunpeng and Mr. Fan, Zili.

I would like to give my special appreciation to my parents for helping me to take care of my children. I could not finish my study at this time without their support, encouragement and sacrifice.

I wish to express my sincere gratitude to my husband Adong for his encouragement at times of difficulty, love and understanding throughout this whole process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Spatial data refer to a set of data which consist of points, lines, rectangles, regions, surfaces, volumes and even data of higher dimension which includes time. The data structures that represent spatial data are used efficiently in applications in computer graphics, environmental modeling, computer vision, database management systems, geographic information system(GIS), image processing and many other areas. Conventional database management systems cannot handle multidimensional data as efficiently as one-dimensional data such as integers, real numbers or strings.

Spatial database systems contain n-dimensional spatial data which represent objects and their positions in space. A spatial index structure not only represents the spatial data but also implements database maintenance and information retrieval operations such as insertion, deletion and query. Many spatial index structures have been developed. Some indexing techniques, such as B-trees and R-trees are dynamic techniques that employ pointers to navigate through their index trees. There is a growing demand to use database technology in engineering and science, such as computer-aided design, software engineering (CAD/CASE) and other automated tools. These tools require both the implementation of more complex data types and more complicated database design tools. Many design tools are built on top of raw file systems to gain more efficiency. System designers are allowed to pile layers of abstracted automation on top of the data structures responsible for performing the fundamental operations of insertion, deletion and query. Most of the data structures are linear which can not support a layered data relationship.

Coburn[COB89] introduced a new spatial indexing technique called the Single Point Index Network(SPIN). The purpose of the SPIN technique is to perform database maintenance and information retrieval on a multidimensional data space. Coburn presented three basic sets of SPIN functions, C_SPIN, S_SPIN and R_SPIN. All SPIN functions can convert multidimensional structured data into a layered one dimensional structure with consecutive index values in each layer. The SPIN structure supports layered data relationships using a multidimensional approach. This approach means when a key is given, the SPIN functions transform it into an index value which is used to find the address of data record. The SPIN structure speeds data storage and retrieval time[COB89]. The R_SPIN method can handle sparse data to reduce allocated, but unused, storage. The advantage of SPIN functions is that they use C library functions such as malloc() to allocate memory dynamically, they can store and retrieve data from a secondary storage device, and they do not require pointer variables.

A query is a question about the records stored in a database. The response is one or more records that match certain characteristics. When creating a query, the database user gives the database the fields and the criteria the database should use to select records.

Database applications often require a complicated class of storage structures in order to answer different types of queries such as exact match queries, partial match queries, and range match queries. An exact match query is the simplest type of query and specifies a value for each key in the record. A partial match query retrieves all records in a file having specified values where some of their attributes are to be found. A range match query retrieves all records for which the attribute values are each within specified ranges. It is easy to answer exact match queries using an R_SPIN function. None of Coburn's original SPIN functions can answer partial match queries and range match queries.

The research objective of this thesis is to introduce queries into SPIN techniques and to improve R_SPIN functions for data storage and retrieval. It proposes a set of improved R_SPIN functions called P_SPIN functions so that respond to exact match queries, partial match queries and range match queries. Two new algorithms for partial match queries and range match queries are presented and implemented. Their performance are analyzed.

The organization of the thesis is as follows. Chapter II is a literature review of some of the spatial data structures. It includes a survey of k-d trees, B-trees, k-d-B trees, R-trees, grid files, doubly-chained trees and multiple-attribute trees. Chapter III introduces basic SPIN functions and R_SPIN sparse data handling. Chapter IV discusses a set of improved R_SPIN functions that can implement exact match queries, partial match queries, and range match queries. Algorithms for the various queries are presented.

Chapter V contains the performance analysis of the improved R_SPIN functions.

Chapter VI presents the summary and suggestions for future work.

CHAPTER II

LITERATURE REVIEW

To handle spatial objects, spatial data structures must focus on some interesting subsets of data. Many of the data structures currently used to represent spatial data are hierarchical. They are based on the recursive decomposition of the embedding space into disjoint regions, until a certain level of resolution is reached. A top-down search in a hierarchical structure can focus on subtrees of nodes that satisfy the query and minimize the search of nodes in unspecified fields. In addition, the use of some spatial data structures provides a spatial index. The role of spatial indexing is to accelerate the retrieval of information based on location, especially for large databases. A spatial index should provide an access path to a location, but not necessarily directly to a particular object. This chapter provides a survey of some of the spatial data structures.

k-d tree

A k-d tree is a k-dimensional binary search tree. It can handle the case of a single record having multiple keys with the range [1...k]. In the term k-d tree, k denotes the dimension of the space being represented. It is a binary search tree with the distinction that at each depth a different attribute value is tested. These values determine the direction in which a branch is to be made. Each node in a k-d tree contains pointers and a discriminator. The pointers are either null or point to their children. The discriminator is an integer between 0 and k-1, inclusive. The nodes in the same level of the tree have the same discriminator. The discriminator of the root node is 0, the discriminator of the next level is increased by 1. The discriminator of the kth level is k-1, and of (k+1)th level, the discriminator is 0 again.

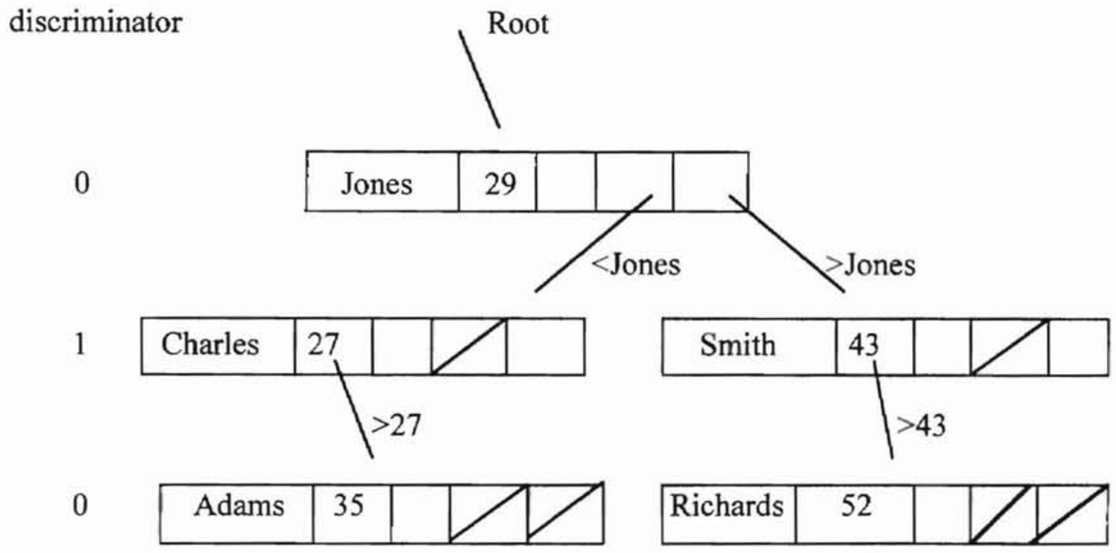

Figure 1 A homogeneous 2-d tree: k_1 is name and k_2 is age

There are many implementations of the k-d tree. A homogeneous k-d tree is a binary tree in which each record contains k keys, some data fields, right and left child pointers, and a discriminator. When inserting a new record, the first key is served as discriminator. The first key of the new record is compared with the first key of the record at the root of the tree to determine whether to go right or left. At the second level, the second key serves as discriminator, and so on to the k^{th} level. Figure 1 is a set of records stored in a homogeneous k-d tree.

In a nonhomogeneous k-d tree, an internal node contains only a discriminator, one key value, and right and left child pointers. All records in nonhomogeneous k-d trees are stored in external nodes or "buckets". Nonhomogeneous k-d trees offer substantial advantages in implementations on secondary storage devices. The k-d tree does not consider paging for secondary memory [BEN75].

Quadtree

A Quadtree is a type of hierarchical data model for storing point data. Each nonleaf node in a Quadtree has four children which represent north-west, north-east, south-west, south-east, and a field to identify the color used. It is able to treat point, line and area data all in the same way, capture metrical details for entities, facilitate various kinds of operations, deal with different ways of measuring attributes, and have consistent locational referencing[Sam89].

B-tree

A B-tree is not a binary search tree. In a B-tree of order d, the root is either a leaf or has between 2 and d children. Each internal node contains between d/2 and d keys, and has between d/2 and d children. A B-tree keeps all external nodes at the same depth. The B-tree's insertion and deletion method can always keep the tree balanced. It is difficult to implement partial match queries and range match queries that involve multiple attributes [COM79].

k-d-B tree

A k-d-B tree is a combination of the k-d tree and the B-tree. The root of the tree is the first partition of the k-dimensional space. Each internal node is a region page which partitions that region into non-overlapping, disjoint subregions. Each external node contains pointer pages that point to records which correspond to a region in the k-dimension space. The k-d-B tree is the first index structure that considers paged secondary memory [ROB81].

R-tree

An R-tree is an extension of the B tree to n-dimensions (n>=2). The R-tree [GUT84] and its variants are designed to organize a collection of arbitrary spatial objects by representing them as d-dimensional rectangles. Each node in the tree corresponds to a smallest d-dimensional rectangle that encloses its child nodes. Leaf nodes contain pointers to the actual objects in the database. The R-tree has the following structure:

Leaf nodes contain a pair of entries (I, tuple-id), where tuple-id is a pointer to a record or spatial object, and I is a n-dimensional rectangle which encloses the spatial object.

Non-leaf nodes contain a pair of entries (I, child-pointers), where child pointers are pointers to the children of a particular node, and I is the bounding box or smallest rectangle that covers all the rectangles in the entries of its children.

If the maximum number of entries is M, then $m<=M/2$, where m is the minimum number of entries in a node.

The spatial index is determined by the rectangle in which the object is contained, with a level in a tree conveying information about resolution. Each object is associated with an R-tree node, just as of a quadtree. Precision of location may be determined for coordinate data contained in the relation.

The R*-tree is developed from the R-tree [BEC90]. Its node structure is the same as the R-tree. It stores multidimensional rectangles as complete objects without clipping or transforming them into higher dimensional points. Following are some of the parameters which are essential for good retrieval performance.

(1) Minimize the area covered by a directory rectangle.

This improves performance since decisions of which paths must be traversed can be made on higher levels.

(2) Minimize the overlap between directory rectangles.

If the overlap between directory rectangles is minimized, then the number of paths to be traversed can be decreased.

(3) Minimize the margin of a directory rectangle.

The margin is the sum of the lengths of the edges. By minimizing the margin, the directory rectangles are shaped more quadraticly, and thus improve the structure. This means the rectangles are clustered into bounding boxes with only little variance of the lengths of the edges. It reduces the area of directory rectangles.

(4) Optimize the storage utilization.

If the height of the tree is kept low, then the query cost is reduced.

Grid file

Several file structures such as inverted files are developed from file structure originally designed for single-key access. They do not adapt well to multikey access for dynamic files. Designing a balanced data structure for multidimensional data is more difficult than for one dimensional data. The grid file is designed to manage a disk allocation storage scheme that contains records in buckets. An overfull bucket results in a split of the space. It requires at most two disk access to retrieve exact match queries. It performs range match queries in large linearly ordered domains efficiently. The splitting and merging of a grid block involves only two buckets. It maintains a lower bound on average bucket occupancy[NIE84].

Doubly-chained tree and Multiple-attribute tree

The doubly-chained tree structure was introduced for file searching and updating [CAR77]. Figure 2 is the doubly-chained tree file organization.

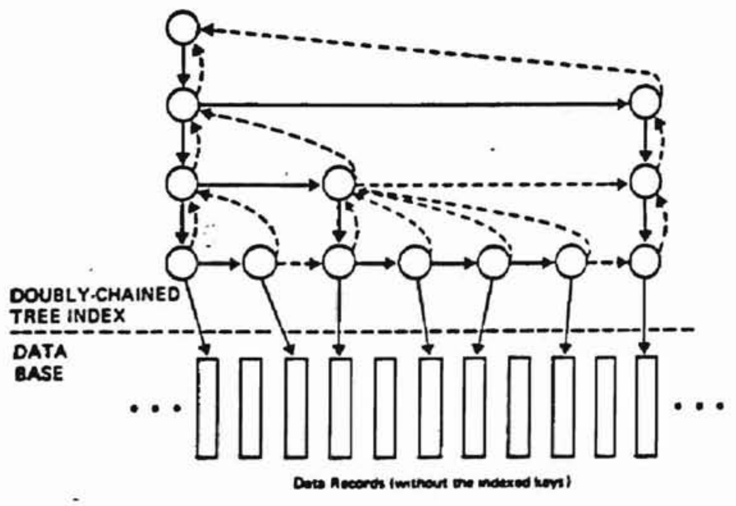
DOUBLY-CHAINED TREE INDEX
DATA BASE
Data Records (without the indexed keys)

Figure 2 Doubly-chained tree file organization[CAR77]

Each level represents the domain of a relation. An internal node contains a keyvalue and three pointers which point to its child, its sibling and its parent. External node contain the data records. The node's keyvalue and associated pointers are sufficient to speed the records' access time and to reconstruct the original record from any point in the tree. It considers the contents of the database, query complexity, device, processor time specifications, and implementation-oriented characteristics when calculating the average access time and storage requirements. The doubly-chained tree structure requires less storage space than sequential file organization. It is easy to update. The disadvantage is file generation, search and update routines are relatively complicated and difficult to implement.

Kashyap et al.[KAS77] analyzed a multiple-attribute tree database organization. It is a modification of the doubly chained tree. An m-level tree is constructed with the root at the top and the m levels of the tree corresponding to the m indexed attributes. A unique path connecting the root node to a terminal node corresponds to a unique combination of the values of the m attributes. Each terminal node has pointers to the pages that contain the corresponding physical records. Figure 3 shows the multiple-attribute tree representation. Figure 4 is the preorder linearization for the tree in Figure 3. For any node in the tree, the addresses of all the nodes in its subtree are consecutive. For instance, the node with value 1 at level 2 has its subtree with consecutive addresses from 3 to 8. Based on this linearization, a directory is constructed to give each node the information about its address, level, value, parent, siblings, page pointers, etc. Within the directory, search begins with the first row and continues with the second row, etc. The directory is helpful to find the page numbers containing the records needed for any given query.

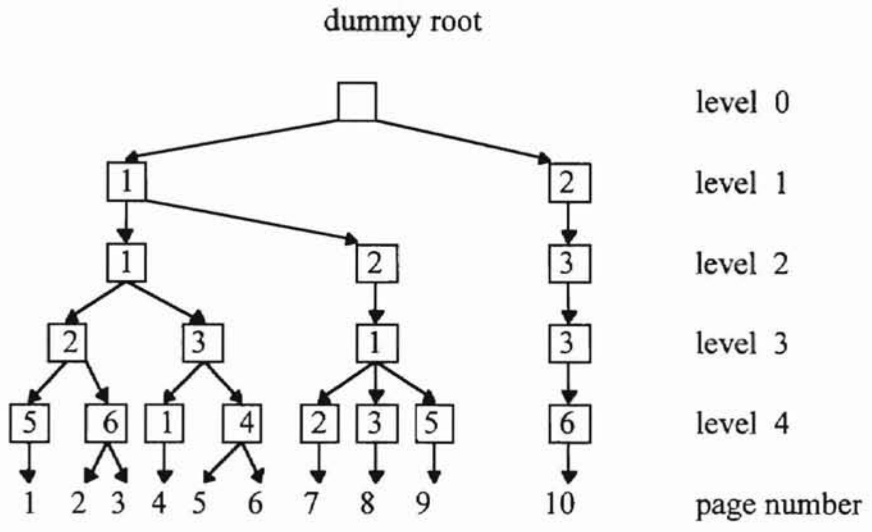

Figure 3 A multiple attribute tree representation [KAS77]

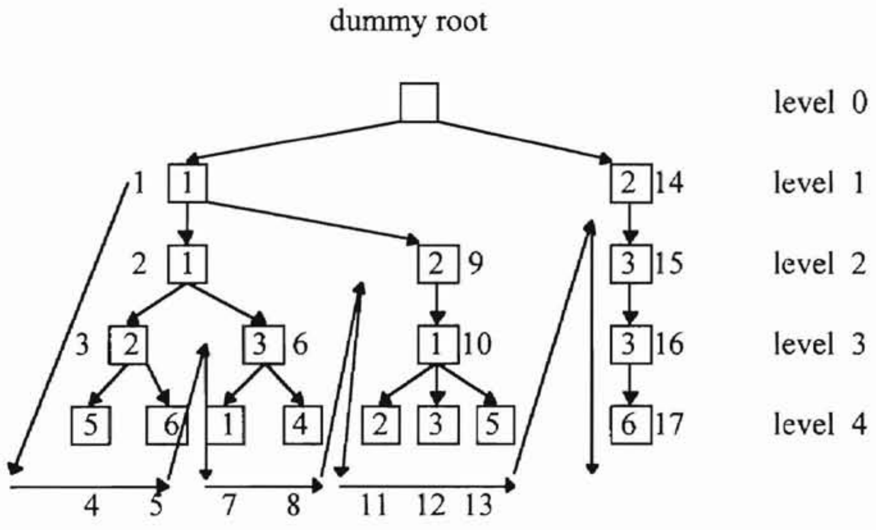

Figure 4 Linearization for the tree in Figure 3 [KAS77]

This method enhances the clustering effect because the records that have the same values of the important attributes are clustered. It is efficient to retrieve hierarchically clustered records in a database with this organization. The limitation is it is inefficient for large dimension size, because it will require a large directory table for each node.

CHAPTER III

THE SPIN FUNCTIONS

Most recent database systems provide database designers with single attribute indexing capability. Spatial data consists of records with k attributes each. Each record can be represented as a point in a k-dimensional space. Multidimensional attribute structures can be used in spatial databases. Some research has been done using multiple attribute structures. Coburn [COB89] modified the multidimensional array data structure and introduced the SPIN multidimensional indexing techniques.

A multidimensional array consists of a given number of dimensions. Each dimension has a fixed number of indices arranged sequentially starting from zero or one. In the C language, " float array [2][2]" is an array of 4 floats in two dimensions with two indices in each dimension. The multidimensional array can be used as an index to search and retrieve data, but some of its characteristics make it impossible both to organize data in a database environment and to perform fundamental database operations such as search, insertion and deletion. The following list summarizes the disadvantages of the multidimensional array.

1. It is not permitted to allocate memory using C library function such as malloc() dynamically with a fixed multidimensional array.
2. It is difficult to do partial key search, storage, and retrieval operations because pointer variables are used as navigators through the dimensions.
3. It cannot store and retrieve data from a secondary storage device easily.
4. The performance of subscript indexing is inefficient.
5. Multidimensional array structures are not sequential. The strucutre poses storage and retrieval difficulties for sequential machine.

The SPIN multidimensional indexing techniques convert multidimensional arrays into single dimensional arrays so that they can perform database operations on multidimensional data spaces. The basic SPIN functions are C_SPIN, S_SPIN, and R_SPIN. SPIN functions transform a given multidimensional subscript combination into an index value which is used to locate the address of data records. SPIN is comprised of a series of algorithms derived from the Fundamental Principle of Counting (FPC).

The FPC states*:

Given a series of m operations 1, 2, 3, ..., m, if the first operation can be performed in m_1 ways, the second in m_2 ways, and so on until the m^th operation, which can be performed in m_n ways, the number of ways the m operations can be performed is

$$II^{n}_{0}\, m_i \qquad (3.1)$$

* from [COB89]

C_SPIN function

The C_SPIN function is a C function which transforms multidimensional subscript combinations into a sequence of unsigned long integer values. For example, given an array "arr[2][2][2]", it follows from the FPC that there are 8 valid multidimensional subscript combinations (keys) for this array as in Figure 5.

If the dimensional size of an array is N, and the number of indices in dimension i (i = 0, 1, 2,..., N-1) is n_i, C_SPIN function will transform the $n_0 * n_1 * n_2 * \ldots\ldots * n_{N-1}$ subscript combinations into a set of sequential numbers I such that

$$I = \{ 0, 1, 2, \ldots\ldots, n_0 * n_1 * n_2 * \ldots\ldots * n_{N-1} - 1 \}.$$

[0][0][0]	←→	0	[1][0][0]	←→	4
[0][0][1]	←→	1	[1][0][1]	←→	5
[0][1][0]	←→	2	[1][0][1]	←→	6
[0][1][1]	←→	3	[1][1][1]	←→	7

Figure 5 The layout of the multidimensional array [2][2][2] converted by Equation (3.1).

The transformation has the following properties:

1. The transformation is well ordered. If two subscript combinations(keys), I_1 and I_2 , are passed to a C_SPIN function, the return values are O_1 and O_2 corresponding to I_1 and I_2 respectively. If $I_1 > I_2$, then $O_1 > O_2$.

2. The transformation is one-to-one and onto. The return value of each distinct subscript combination passed to C_SPIN function is one and only one.
3. The transformation is sequential. Assuming the distinct subscript combinations I_1 and I_2, $I_1 < I_2$, are considered as normal base 10 integers. If there does not exist an I_k such that $I_1 < I_k < I_2$, the return values O_1 and O_2 corresponding to I_1 and I_2 have the relationship:

   $O_2 = O_1 + 1$.

There are two limitations of C_SPIN: first, C_SPIN is restricted to a dimension size of not more than five, because the size of unsigned long integer values converted by formula (3.1) grows exponentially when the number of dimensions increases. Consider "arr[2][2][2][2][2][2][2]" transformed by C_SPIN, there are 128 sequential numbers such that I = 0, 1, 2, ……, 128; Secondly, indexing of partial combinations is not allowed, because it cannot access to the middle dimensions of the multidimensional array structure.

S_SPIN function

The S-SPIN function is a modification of the C_SPIN function. The S_SPIN iteratively applies C_SPIN one dimension at a time. S_SPIN uses formula (3.2) [COB89] to transform a multidimensional array to a layered single dimensional array.

$$\text{rec\_number}[i+1] = \text{rec\_number}[i] * 10^{ex[i]} + k[i+1] - \text{rec\_number}[i] * kr[i] \quad (3.2)$$

where

rec_number[i+1] = the index value for the $(i+1)^{st}$ level

rec_number[i] = the index value for the previous level or iteration.

max[i] = the number of indices in the ith dimension.

ex[i]s are exponents computed as follows:

if (max[i+1] > 0 && max[i+1] <= 10)   ex[i] =1,

else if (max[i+1] > 10 && max[i+1]<=100)   ex[i] = 2,

else if (max[i+1] > 100 && max[i+1]<=1000)   ex[i] = 3,

etc.,

k[i] = the value of the multidimensional subscript within the i^{th} dimension,

kr[i]s are values computed as follows:

if (max[i+1] >0 && max[i+1] <=10)   kr[i] = 10 - max[i+1],

else if (max[i+1] >10 && max[i+1] <=100)   kr[i] = 100 - max[i+1],

else if (max[i+1] >100 && max[i+1] <=1000)   kr[i] = 1000 - max[i+1],

etc.,

i = the level (0,1,2,…., N-2).

If the dimension size is two. Formula (3.2) is applied only once to compute the index values of level one. Coburn defined the level as "one less than the number of the dimension within a multidimensional array"[COB89]. In an array "arr[2][2][2][2]", level 0 refers to the first dimension "arr[2]", level 1 refers to the first two dimensions "arr[2][2]", and so on. Figure 6 is the layout of the multidimensional array "arr[2][2][2][2]" converted by formula (3.2)

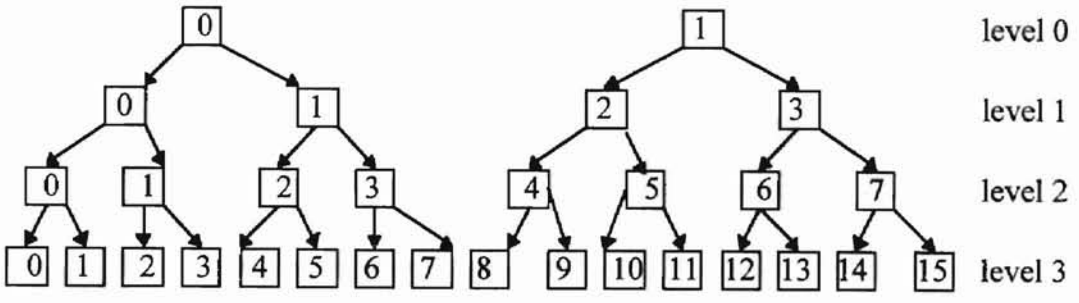

Figure 6 The layout of a multidimensional array in SPIN

in this example :

dimension size N = 4,

max[0] = max[1] = max[2] = max[3] = 2,

ex[0] = ex[1] = ex[2] = ex[3] = 1, since max[i] <10,

kr[0] = kr[1] = kr[2] = kr[3] = 10 - max[i] = 7,

max[i], ex[i] and kr[i] remain unchanged for a given multidimensional array. The value in the rectangle is the index value in each level. To calculate the index value in level one, the first iteration produces one of the possible 4 numbers 0, 1, 2, 3. The second iteration in level two transforms one of the 4 values from the first iteration into one of the 8 numbers 0, 1, 2, ..., 7. The third iteration in level three transforms one of the 8 values from the second iteration into one of the 16 numbers 0, 1, 2, ..., 15.

The S_SPIN function iteratively applies C_SPIN at each level to calculate the index values. This improvement makes it convenient to use S_SPIN functions in large dimension size (N > 5) array. Unlike C_SPIN functions, the S_SPIN functions generate return values at each level. This permits the computation of a partial subscript combination (partial key). In the array "arr[2][2][2][2]", index values can be computed for "arr[2][2]", "arr[2][2][2]" and "arr[2][2][2][2]".

S_SPIN is also a reversible procedure. The index value calculated by the S_SPIN function can be converted by an RS_SPIN function into its original subscript combination. This reverse procedure can be done at any level. In the an array of "arr[2][2][2]", when the subscript combination "0,1,1" is passed to S_SPIN, the return integer value (index value) is "3". If in the same array structure, the index value "3" is passed to RS_SPIN, a pointer is returned that points to an array of three unsigned integers such that k[0] = 0, k[1] = 1 and k[2] = 2.

Both C_SPIN and S_SPIN functions convert a multidimensional subscript to a one dimensional index value. Neither of them can handle the representation of the sparse data efficiently. R_SPIN's sparse data handling makes it more practical than the other two SPIN functions.

R_SPIN function

The R_SPIN function can convert multidimensional subscripts into one dimensional layered sequential index values as the S_SPIN function does. In addition, the R_SPIN function considers the sparse data situation and creates a multidimensional array that is less sparse. The definition of the sparse data is " when a set of indices at one level of a multidimensional array are, in reality, mapped to only a very few indices of the next level, the array is said to be sparse at that level[COB89]". It happens sometimes in database operations that the number of indices actually required in each dimension of a multidimensional array decreases when the number of dimensions increases. The R_SPIN function can predict the actual mapping in each dimension and reduce the amount of memory or disk storage.

Consider an array " arr[20][30][40]", when this array is declared, every possible subscript combination is mapped as described above. Equation (3.1) computes the number of mappings to be 24,000 (20 * 30 * 40). If in reality, each of the 20 indices in the level 0 might map to at most 3 of the 30 indices in the level 1, and each of the indices in the level 1 might map to at most 2 of the 40 indices in the level 2. The R_SPIN reduces the number of mapping in the array representation arr[20][30][40] (=24,000) to a level representation arr[20][3][2] (=120).

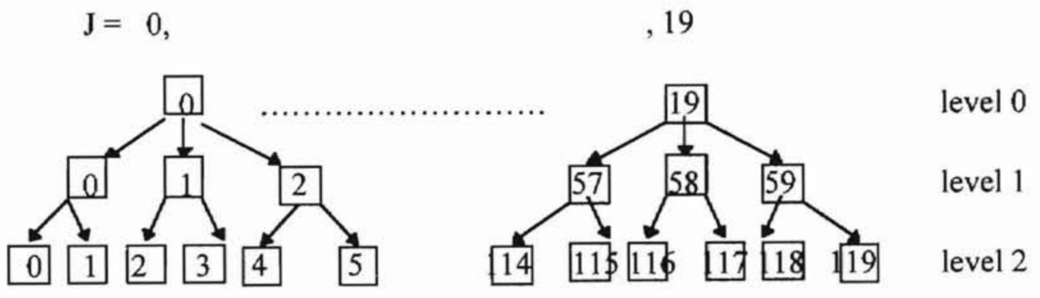

Figure 7 The layout of a multidimensional array in R_SPIN

Figure 7 shows the layout of the reduced dimensional size of the multidimensional array "arr[20][30][40]" index file returned by the R_SPIN function. When "arr[20][3][2]" is declared, an index file is created. The index values of level one are 0, 1, 2, ……, 19. They are used as tree index. The nodes in level one and two are initialized to zero.

A multidimensional subscript is converted by the R_SPIN function into an unique unsigned long integer called a V-value to help to store and retrieve data. It creates an index file full of V-value instead of using navigational pointers. A new datum is stored at each level in the first available storage location. When the subscript combination "p[4][5][6]" is passed to R_SPIN, the R_SPIN argument "k" is set to k[0] = 4, k[1] = 5, k[2] = 6. The V-values corresponding to "p[4][5]" and "p[4][5][6]" are calculated by formula (3.2). They are stored at first available storage locations at level one and level two. When another subscript combination "p[4][7][8]" is passed to R_SPIN, the V-values corresponding to "p[4][7]" and "p[4][7][8]" are stored at next available storage locations at corresponding levels. To calculate the index value of a subscript combination, the R_SPIN applies formula (3.2) by using the location value in which it deposits the V-value. Assume the V-value of the subscript combination "p[4][5][6]" is stored in the first node at each level, its location value is "k[4][0][0]". The location value of the next subscript combination "p[4][7][8]" is "k[4][1][0]".

Each element in level zero can map to three storage locations at level one, and each element in level one can map to two storage locations at level two. This leads to a problem. What will happen if there are more than three elements mapped from level zero to level one or the number of storage locations required exceeds the number allotted within the index file? The R_SPIN function will give an "overflow" error message. The reason for an overflow situation is the inability to predict the sparse mapping in a multidimensional array accurately.

It is assumed that there are m keys consisting of D characters with n_i possible choices for the i th character. The probability of encountering an overflow in each level can be calculated by formula (3.3):

$$p(c)_{i+1} = p(a > R_{i+1})_{i+1} \mid (b > R_{i+1}) \qquad (3.3)$$

where a = the number of times a distinct character string appears at level i.

b = the number of distinct mappings for single character from level i to i+1.

c = an overflow at level i+1 of a multidimensional array.

R_{i+1} = the maximum number of characters allowed in the $(i+1)$ th position.

The probability of overflow is

$$p\,(\text{overflow}) = p(c)_1 + p(c)_2 + p(c)_3 + \ldots\ldots + p(c)_{D-2}$$

$i = 0, 1, 2, \ldots\ldots, D.$

D = number of dimensions.

From formula (3.3), we can see the probability of overflow is determined by

1) The probability of the number of times a distinct character string appears at level i exceeds R_{i+1}.

The number of times a given character will appear in the i th position is found by applying the binomial distribution $[\,m!/r!(m-r)!]p^r q^{m-r}$. The probability is,

$$P_i(a) = [m!/r_i!(m-r_i)!]p_i^{ri}q_i^{m-ri}$$

where m is the number of keys,

r_i is number of times a given character appears in position i,

p_i is the probability that a character appears in position i,

$q_i = 1 - p_i$,

$i = 0, 1, 2, 3, \ldots\ldots., D - 2$,

D is the number of dimensions in the multidimensional array.

2) The probability of the number of distinct mappings exceeds R_{i+1}.

The problem is P_i changes with different applications. It is impossible to calculate P_i without knowing the application environment. This is a tradeoff. The R_SPIN can reduce the sparse mapping and save memory or disk storage but also increase the risk of overflow.

To retrieve the index value of a subscript combination, the R_SPIN function compares the value calculated by formula (3.2) with the V-value stored in the index file. Data are retrieved if the corresponding values are equal. If "p[4][7][8]" is passed to R_SPIN, it employs formula (3.2) to calculate a value by using "4" and "7" as subscripts. It finds the location in the index file whose tree index is "3". It will compare this value with the V-value stored in each node in level one. If it equals to one of the node's V-value, it employs formula (3.2) to calculate another value by using "4", "7" and "8". When this value equals to one of the node's V-values in level two, then "p[4][7][8]" is found in the index file and the index value of the multidimensional subscript is retrieved.

The location of each node in the index file can be computed according to the following indexing formula:

$$(\Sigma_1^{i-1} J_n * K_n + j * Ki + k) * \text{sizeof} (V) \quad (3.4)$$

where:

i = the level number.

j = the index value in level $i - 1$.

k = the storage location in the j^{th} entry of level $i-1$.

K_n = the size in the dimension n.

J_n = the total entries in the dimension $n-1$.

V = the value stored in each node.

An example can be given in Figure 7. Consider the node whose index value is 114 in level 2, in this case, $i = 2, j = 57, k = 0, K_2 = 2, J_1*K_1 = 60$. If sizeof (V) is assumed to be 1, the location of this node in the index file is $60 + 57 * 2 + 0 = 174$. Here the level zero value 0, 1, 2, ……, 19 are used as tree index. They are not stored in the index file.

We can conclude from the processing of R_SPIN that it has all the advantages of C_SPIN and S_SPIN functions. It can significantly reduce the number of sparse mappings. It is an efficient way to store and retrieve keys.

CHAPTER IV

IMPROVED R_SPIN OPERATIONS

Methods for the storage and retrieval of multidimensional data are of prime importance in the design of database systems and for specific applications including both the management of geographical data and graphics algorithms. The R_SPIN function can be used to store and to retrieve multidimensional data. The primary R_SPIN algorithm is given by Coburn[COB91]. It first dynamically allocates memory for several integer arrays that contain the input array and return index values, and it calculates parameters for formula (3.2). When the input multidimensional array is given, the R_SPIN creates an index file (index tree). The subscript at level 0 is used as the tree index. When a subscript combination is passed to R_SPIN, it calculates the V-value using formula (3.2) and stores the V-value in the first empty node at the corresponding level. The R_SPIN uses the location value of these nodes to calculate index values using formula (3.4). Each node contains two values, the V-value and the index value. The index file is full of V-values instead of navigational pointers. The index values are consecutive within each level.

To retrieve a record, a subscript combination is passed to R_SPIN. It uses formula (3.2) to convert the multidimensional subscript into a one dimensional V-value. It then uses formula (3.3) to locate the node and read the node's value into temporary buffer. If the value in the temporary buffer equals the V-value, then the program calculates and returns the corresponding index value.

Searching in a multidimensional space employs a breadth first approach. The program first searches level 0 to determine whether the key component matches the value of the current node. If it matches, the program searches its first child node until a terminal node is reached. If it doesn't match, the program searches its brother node. The tree shown in Figure 6 can be represented as the tree shown in Figure 8. Each node has pointers both to its next brother and its first child.

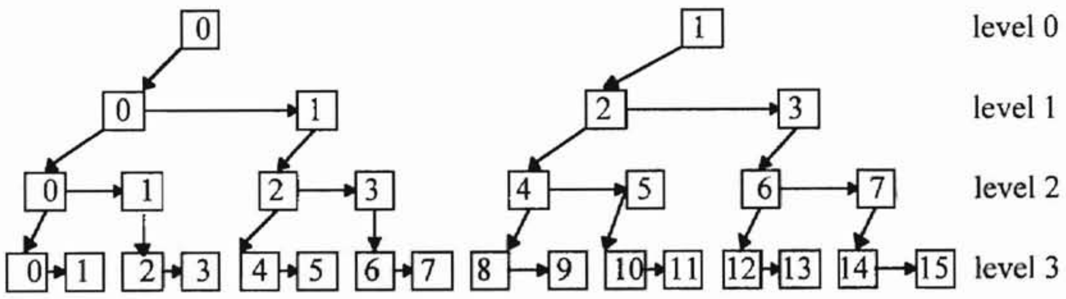

Figure 8 The layout of R_SPIN index

The algorithm as given by Coburn [COB91] is:

[INPUT]: 1. A pointer to an index file

2. The number of dimensions in the input array

3. An unsigned integer between 0 and the number of dimensions which determine the number of return value

4. Maximum number of elements in each dimension of the input array
5. Number of mappings which actually exist between each level of the input array
6. The subscript for each dimension of the input array
7. A mode which determine the characteristic of the index file
8. A control which will change if the array structure is changed

[OUTPUT]: Index values of the input subscript combination

RS 1. Set level $i = 0$. Initialize the node value as 0 except that the index value of level 0 is the level 0 subscript combination. If the value of level 0 subscript combination equals the tree index value, choose the root node of the index tree as root "T".

RS 2. Increment i by 1 and go to next level. If i is greater than the number of dimensions, terminate the R_SPIN function and return the index values.

Set the subscript of the dimension $k = 0$.

RS 3. In level i, convert the multidimensional subscript into a one dimensional V-value using formula (3.2). This V-value is used as an identifier for each subscript combination.

RS 4. Calculate the location of k^{th} child node in the index file by using formula (3.4). Read the node value into a temporary buffer "temp".

If "temp" equals 0, go to RS 5.

If "temp" equals the V-value that already deposit in the node, go to RS 6.

If k is greater than the number of dimension size within that level, go to RS 7.

In default case, go to RS 8.

RS 5. The kth node is the first empty node in level i of this subtree. Deposit the V-value computed in RS 3. Compute the index value of the subscript combination by using the node's location value in the index file and applying formula (3.2). Choose this node as new root "T", and go to RS 2.

RS 6. The subscript combination is already in the index file. Computer the index value of the subscript combination by using the node's location value in the index file and apply formula (3.2) one more time. Choose this node as new root "T", and go to RS 2.

RS 7. This is the overflow case. The number of storage locations required exceeds the number allotted within the index file. The R_SPIN will assign -1 to the index value of that level and terminate the execution.

RS 8. Go to next sibling and increase k by 1.

Go to RS 4. [COB91]

From the R_SPIN algorithm, we can see that the R_SPIN applies formula (3.2) two times to calculate the index value and it can only retrieve data that is specified in each level. This is the procedure of exact match query: to ask about a specific record (defined by the k keys) in the file.

In database usage, partial match queries and range match queries are used more frequently than exact match queries. Assuming that each record contains k keys, a partial match query specifies s key values, $s<k$, that must be matched. The remaining k-s keys are left unspecified. An example of a partial match query with three keys specified might

happen in a student database: search for all students with sex = male and math2103 = F and semester = fall 1994.

A range match query is the same as an exact match query except that a range of required values rather than a single value may be specified for each key. All records that have values between specified ranges are retrieved. An example is finding all students who have credit hours between 30 and 50, grade point average between 3.2 and 3.7, and an age between 20 and 22.

The R_SPIN function has difficulty in answering both partial match queries and range match queries because each node is identified by its V-value. A V-value at level i is calculated by the subscript combination of level i and the V-value of its parent. To retrieve the index value of a permutation P[1][4][5][6] in the structure of Figure 6, the level 1's V-value V[1] is calculated by formula (3.2) using level one subscript combination [1][4], the level 2's V-value V[2] is calculated by V[1] and [5], the level 3's V-value V[3] is calculated by V[2] and [6]. For a query [1][?][0][?], the V-value at level 1 is unknown, so are the V-values of level 2 and level 3. The solution is to store the subscript combination in each node as its real value instead of a V-value. For example, for P[1][4][5][6], 1 is stored in level 0, 4 is stored in level 1, 5 is stored in level 2 and 6 is stored in level 3. This method also saves time in calculating V-values by using formula (3.2). The modified R_SPIN function is called P_SPIN which contains three functions. They are Exact_Match, Partial_Match and Range_Match functions. When a subscript combination is passed to P_SPIN, it stores the value of the subscript combination in the first empty node at the corresponding level, then it uses the location value of these nodes to calculate index values by using formula (3.4). Each node contains the subscript value and the index value. It will call Exact_Match, Partial_Match or Range_Match function depending on different queries. The algorithms for Exact_Match, Partial_Match and Range_Match functions are given as follows.

Exact- Match Algorithm

[INPUT]: 1. A pointer to an index file

2. The number of the current level
3. An unsigned integer between 0 and the number of dimensions which determines the number of return value
4. Number of mappings which actually exist between each level of the input array
5. Parameter ex[i]
6. Parameter kr[i]
7. Index value of the current level
8. Node's location in the index tree
9. Input queries

[OUTPUT]: Index values of the input subscript combinations

EM 1. In the current level (level i) of the subtree, set the subscript of the dimension k= 0.

EM 2. Calculate the location of k th child in the index file by using formula (3.4). Read the node value into a temporary buffer "temp".

EM 3. If "temp" equals the node value that already deposit in the node, go to EM 4.

If k is greater than the number of dimension size within that level, go to EM 7.

In default case, go to EM 8.

EM 4. The subscript combination is already in the index file. Computer the index value of the subscript combination by using the node's location value in the index file and apply formula (3.2).

If the current node is a terminal node, go to EM 5.

If the current node is not a terminal node, go to EM 6.

In default case, go to EM 8.

EM 5. Print out the index value of the input query and terminate.

EM 6. Check the value of next level input subscript.

If it is specified, call Exact_Match(i + 1).

If it is not specified, call Partial_Match(i + 1).

EM 7. This is the overflow case. The P_SPIN will assign -1 to the index value of that level.

EM 8. Go to next sibling and increase k by 1, go to EM 2.

The partial match algorithm can retrieve more than one key from the index file. It allows the wild card search P[1][?][0][?] where the "?" character is the wild card designation. Partial_Match(i) is called when the current level is unspecified. If the current level is the terminal level, it gives the index value of the input query. If the current level is not the terminal level, calculate the index value of this level and go to next level. At the next level, if P[i] is specified, Exact_Match(i + 1) function is called. If P[i] is not specified, Partial_Match(i + 1) function is called.

Partial -Match Algorithm

[INPUT]:
1. A pointer to an index file
2. The number of the current level
3. An unsigned integer between 0 and the number of dimensions which determines the number of return value
4. Number of mappings which actually exist between each level of the input array
5. Parameter ex[i]
6. Parameter kr[i]
7. Index value of the current level
8. Node's location in the index tree
9. Input queries

[OUTPUT]: Index values of the input subscript combinations

PM 1. If the current level is the terminal level, go to PM5.

PM 2. In the current level (level i) of the subtree, set the subscript of the dimension k= 0.

PM 3. Calculate index values of all nodes in this level of this subtree by using formula (3.2).

PM 4. Go to next level, check the value of next level input subscript.

If it is specified, call Exact_Match().

If it is not specified, call Partial_Match().

PM 5. Compute the index value of the subscript combination by using the node's location value in the index file and apply formula (3.2).

Print out the index values and terminate.

A range match algorithm can retrieve keys within a range of values for each dimension. All records that have values within the ranges are retrieved. It first reads two input queries P1 and P2, where P1 is the lower bound and P2 is the upper bound of the range. In the current level i, formula (3.4) can be used to calculate the location of the kth child node, compare the node's value with P1[i] and P2[i]. If the node's value is in range of P1[i] and P2[i], then calculate its index value and go to next level until the terminal level is reached.

Range-Match Algorithm

[INPUT]: 1. A pointer to an index file

2. The number of the current level

3. An unsigned integer between 0 and the number of dimensions which determines the number of return value

4. Number of mappings which actually exist between each level of the input array

5. Parameter ex[i]

6. Parameter kr[i]

7. Index value of the current level

8. Node's location in the index tree

9. Upper bound of the input query

10. Lower bound of the input query

[OUTPUT]: Index values of the input subscript combinations

RM 1. In the current level (level i) of the subtree, set the subscript of the dimension k= 0.

RM 2. Calculate the location of k^{th} child in the index file by using formula (3.4). Read the node value into a temporary buffer "temp".

RM 3. If "temp" is greater than the lower bound and less than the upper bound, go to RM 4.

If "temp" is less than P1[I], or "temp" is greater than P2[I], then go to RM 8.

If k is greater than the dimension size in level I, assign its index value to -1 and go to RM 7.

In default case, go to RM 8.

RM 4. The subscript combination is already in the index file. Computer the index value of the subscript combination by using the node's location value in the index file and apply formula (3.2).

If the current node is a terminal node, go to RM 5.

Else go to RM 6.

RM 5. Print out the index value of the input query and terminate.

RM 6. Go to next level, increase k by 1.

Go to RM 2.

RM 7. This is the overflow case. The P_SPIN will assign -1 to the index value of that level.

RM 8. Go to next sibling and increase k by 1, go to RM 2.

CHAPTER V

IMPLEMENTATION

Programming Environment

The programs are written in C programming language. The environment used to develop the programs is a personal computer. The operating system on this machine is LINUX, a complete UNIX clone. It has full UNIX features. LINUX is a multi-user, multitasking operating system. It uses the X-window system graphical user interface and offers several different configureurable window managers, TCP/IP, UUCP, PPP networking and much more.

Experimentation Strategy

A testing program is used to retrieve multidimensional data, to test the average time complexity and the search usuage of the P_SPIN function. The number of keys in each record is k =3, 5, 7. For each k, a set of N records (N > 2000) is generated recursively. The experimentation proceeds in three phases. In the first phase, the entire multidimensional tree structure is built by a series of insertions. This phase is followed by data retrieval process. Finally, the average retrieval time of exact match query and partial match query is tested.

When running the testing program, a menu is displayed:

1. Create a multidimensional tree structure
2. Exact_Match query data retrieval
3. Partial_Match query data retrieval
4. Range_Match query data retrieval
5. Test the average retrieval time of an exact match query
6. Test the average retrieval time of a partial match query

The user should first choose 1 to initialize a multidimensional tree structure. This operation asks the user to enter the dimension size, the maximum size of each dimension, and the actual mapping of each dimension. After all data are entered, the test program will calculate the parameters used in the P_SPIN function. It will then recursively generate distinct multidimensional subscript combinations and pass them to the P_SPIN function. The P_SPIN function will deposit subscript in each level as node value and display its index value. The index value and node value are stored in each node.

The multidimensional tree structure is unchanged during other operations. Choose 2. Exact match query data retrieval, the user is asked to enter a multidimensional subscript combination(query). It is passed to the P_SPIN function and Exact_Match function is called. If the subscript combination is already stored in the index file, its index value is retrieved. It will give a message if the record is not found. Operation 3. Partial match query data retrieval allows the user to enter an input query that contains one or more wild card designation in any level except level 0(because level 0 is used as tree index). It is passed to the P_SPIN function and Partial_Match function or Exact_Match function is called. It will retrieve more than one key from the index file. Operation 4. Range match query data retrieval asks the user to enter a lower bound and an upper bound of the multidimensional subscript combination. The index values of the multidimensional subscript combinations between the lower bound and the upper bound are retrieved.

Operation 5 is to test the average retrieval time of an exact match query. Because the retrieval time of an exact match query is less than 1 second, the retrieval time is more accurate if more than 100 queries are tested at a time. Each data set contains more than 2000 records. When operation 5 is chosen, the test program will read the multidimensional subscript combination at least 10 times. At the first time, the number of the multidimensional subscript combination is 100. At the second time, the number of the multidimensional subscript combination is 200. A loop of 10 times can calculate the average time of 100, 200, 300, …, 1000 input queries. Operation 6 is to test the average retrieval time of a partial match query. Its procedure is similar to operation 5, except that the input query contains one or more unspecified subscript.

CHAPTER VI

ANALYSIS OF RESULTS

In this chapter, we present some examples to analyze the time complexity and search usuage of the P_SPIN function. The usage usuage is defined as the difference between the number of nodes accessed per query and the number of total nodes.

Table I Exact_Match algorithm test data

DATA SET	1	2	3
Dimension size	3	5	7
Max size of each dimension	14 15 10	5 10 10 20 15	3 5 8 10 12 15 5
Actual mapping of each dimension	14 14	5 5 5 5	3 3 3 3 3 3
Number of total nodes	2744	4096	2187

Three P_SPIN algorithms and the original R_SPIN algorithm were implemented. The same set of data were used to obtain the performances of the Exact_Match algorithm and the R_SPIN algorithm for comparison purposes. This set of experiments was performed with three groups of data shown in table I. To make experimentation and analysis easily, we assume that the actual mapping of each dimension has the same size.

The results are reported from Figure 9 to Figure 11. Figure 9 gives the experimental results of the Exact_Match function using the set of data shown in table I. In Figure 9, "Total Number of Permutations" refers to the total permutation numbers used in the experiment.

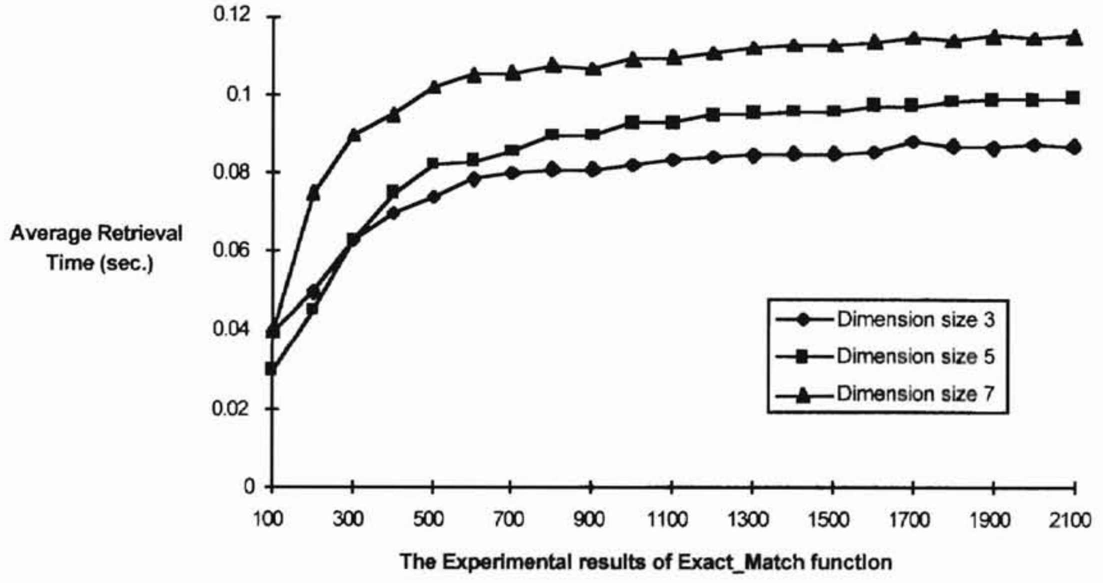

Figure 9. The experimental results of Exact_Match function

In the structure of dimension size 3, each nonleaf node has 14 children, while in the structure of dimension size 7, each nonleaf node contains only 3 children. Figure 9 shows that the average retrieval time increases when the number of dimensions increases.

From the layout of R_SPIN index in Figure 8, we can see that it takes more time to visit between levels than to visit between siblings.

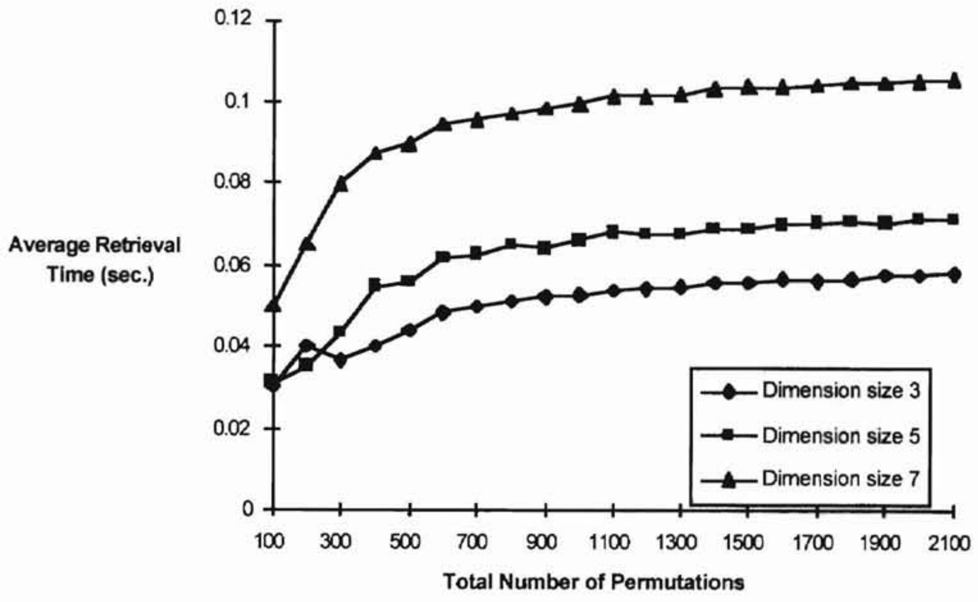

Figure 10. The experimental results of Coburn's R_SPIN function

Figure 10 gives the experimental results of Coburn's original R_SPIN function using the same data set as in Figure 9. The average retrieval time grows as the dimension size increases.

The comparison of the Exact_Match function and the R_SPIN function is given in Figure 11(a). In the same dimension size, the average retrieval time per query of the R_SPIN function is less than the average retrieval time of the Exact_Match function. Although the Exact_Match function saves the time to calculate the V-value, it takes time to do comparison in each level to determine if it is specified and which function to be called.

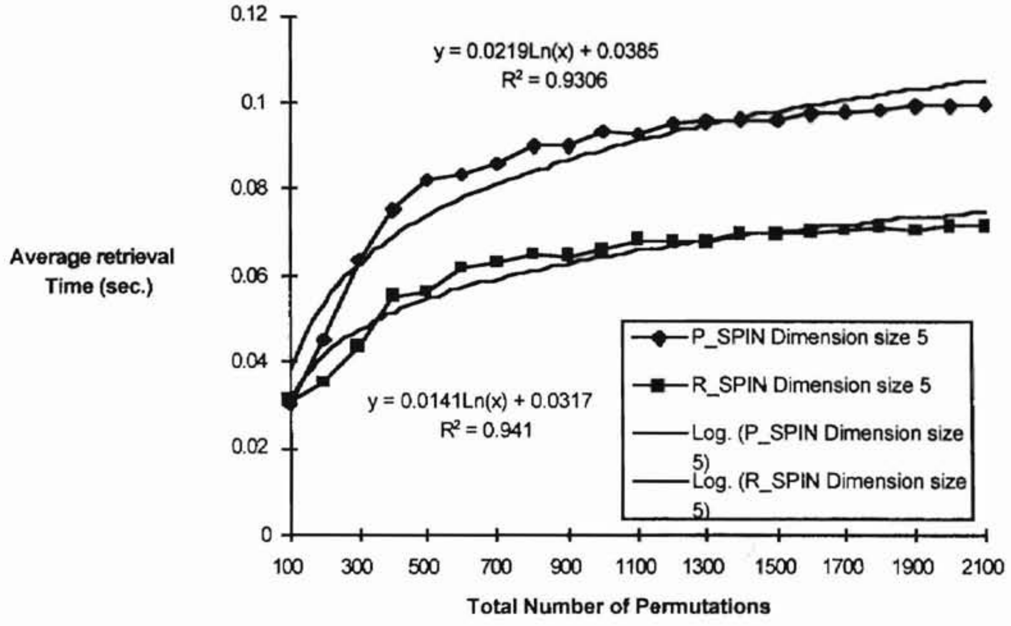

Figure 11(a). Comparison of Exact_Match function and R_SPIN function

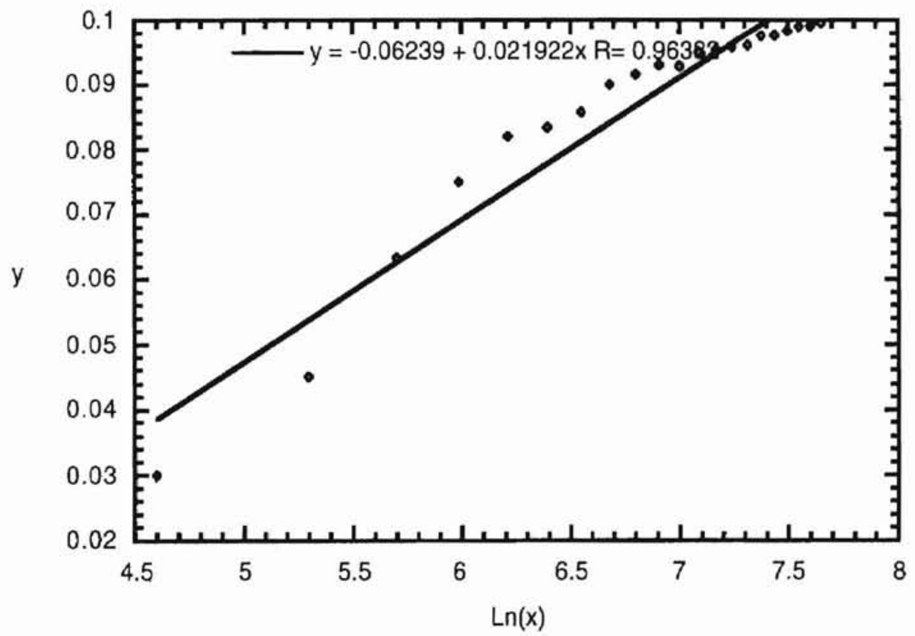

Figure 11(b). Analysis of Figure 11(a) Exact_Match function

Figure 11(b) is the analysis of the Figure 11(a). x and y denote the total number of permutations and the average retrieval time of Figure 11(a) respectively. It uses a nature logarithm scale of x instead of x. The linear curve shows that x and y has logarithmic relation. Both Exact_Match function and R_SPIN function has logarithmic behavior which is shown in Figure 11(a) and Figure 11(b).

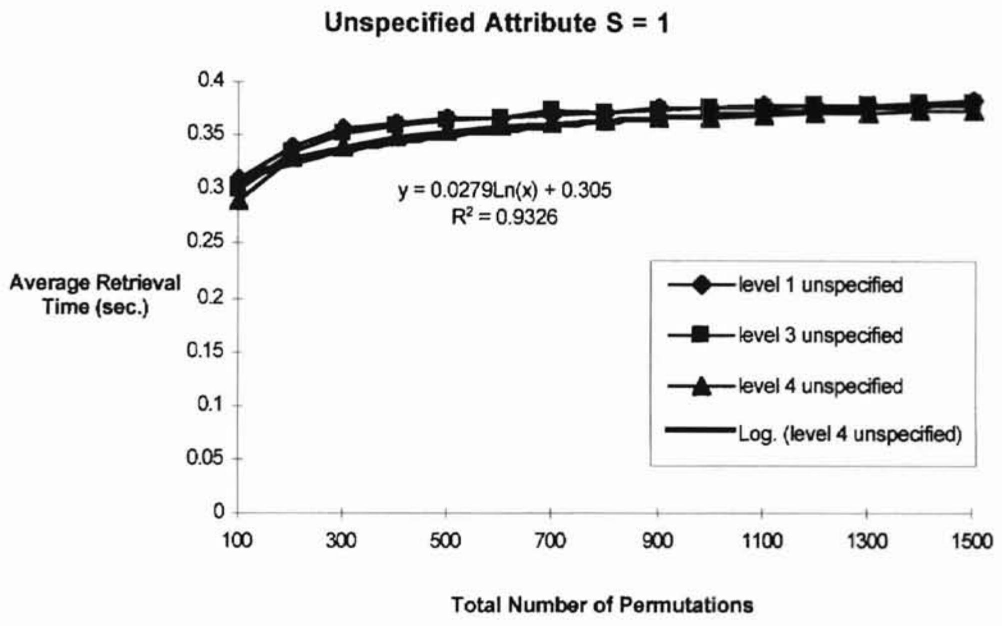

Figure 12. Average retrieval time of unspecified attribute S = 1.
S is the number of unspecified attribute

To analyze the performance of the Partial_Match function, the average retrieval time and the average number of node accessed per query in different number of specified attribute are tested. The data set 2 in table I is used as test data. The testing partial match queries are shown in table II.

TABLE II. Partial Match Queries

Number of attribute specified	Query
1	(10111)
	(11011)
	(11101)
	(11110)
2	(10101)
	(11010)
3	(10001)
	(11000)

1 denotes the specified level. 0 denotes the unspecified level

Figure 12 is the average retrieval time of unspecified attribute S = 1. The unspecified level is 1, 3, 4 respectively. The results are very close. There is no big difference of which level is unspecified when the unspecified attribute is 1.

Figure 13 gives the results of unspecified attribute S = 2. The average retrieval time of unspecified in level 1 and 3 is longer than the average retrieval time of unspecified in level 2 and 4. When a certain level is unspecified, the Partial_Match function is called to calculate all the index values of this level and go to the next level. If the next level is also unspecified, the index values of the next level in the subtree are calculated. It is obvious that more index values are calculated when higher level is unspecified.

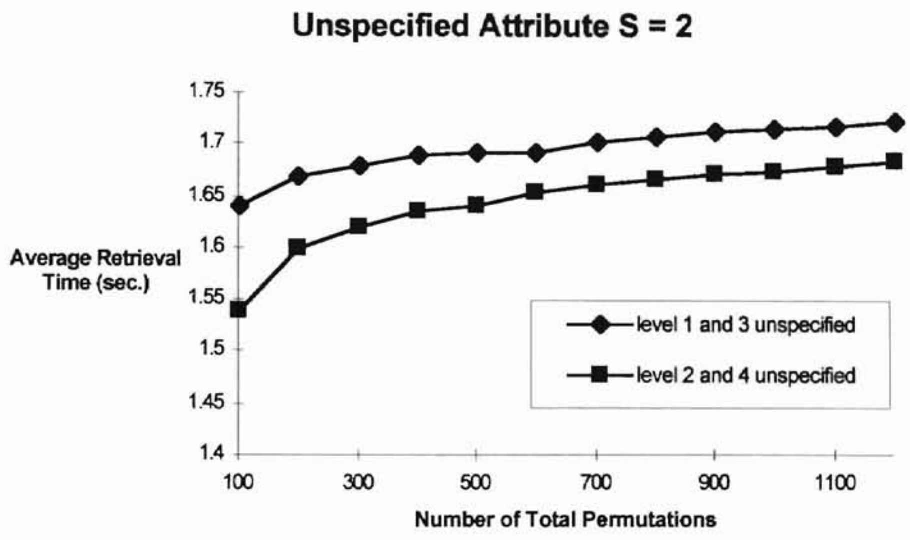

Figure 13. Average retrieval time of unspecified attribute = 2

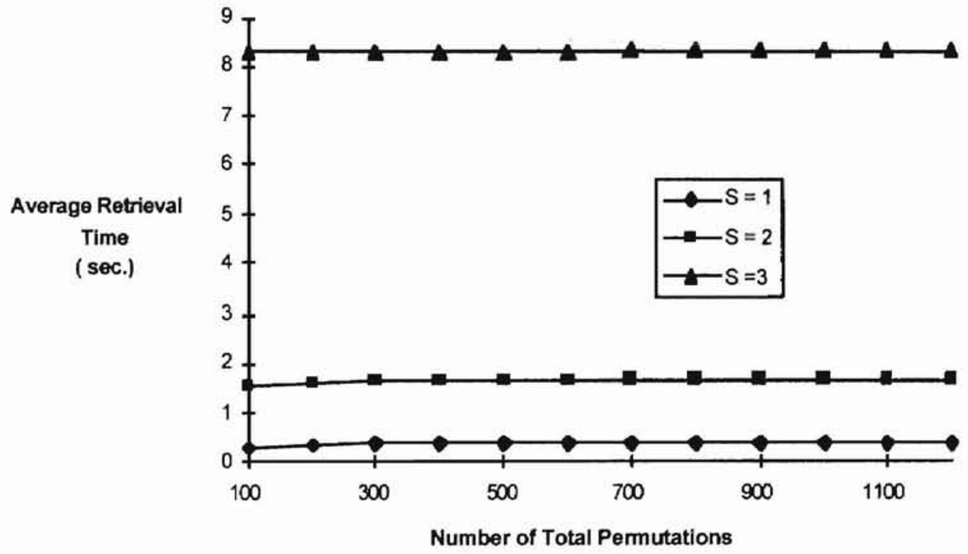

Figure 14. Comparison of average retrieval time in different S

Figure 14 is the comparison of the average retrieval time when the unspecified attribute equals 1, 2, and 3. The two curves at the bottom are exactly the same as in Figure 12 and Figure 13. The average retrieval time increases greatly when the S increases from 2 to 3. The experiments shows the logarithmic time complexity for the average retrieval time of the Partial_Match function.

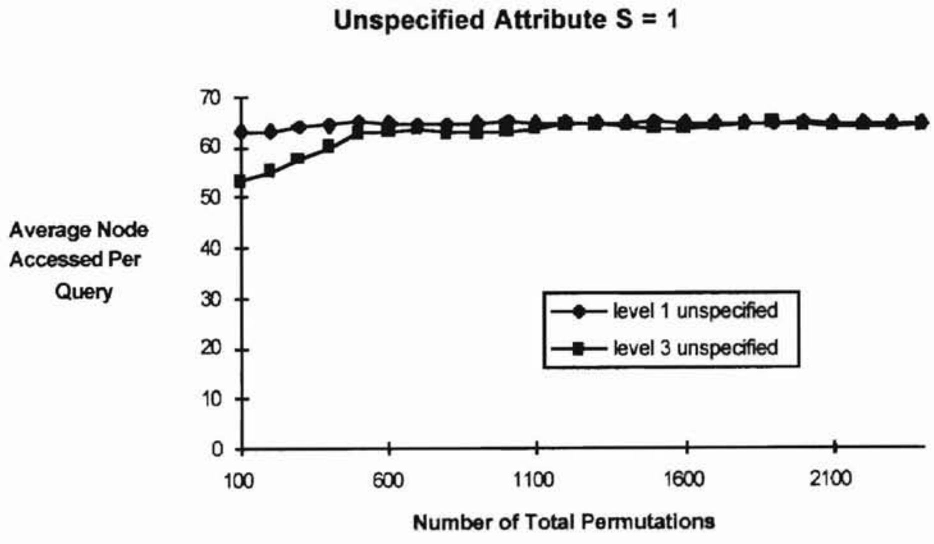

Figure 15. Average node accessed of unspecified attribute = 1

Figure 15 is the results of average node accessed per query when the unspecified attribute S=1. From the experiment, the average number of node accessed per query is same when the unspecified level is 3 or 4. Unspecified in level 1 results in more node accessed than unspecified in level 3 when the number of total permutation is small. We

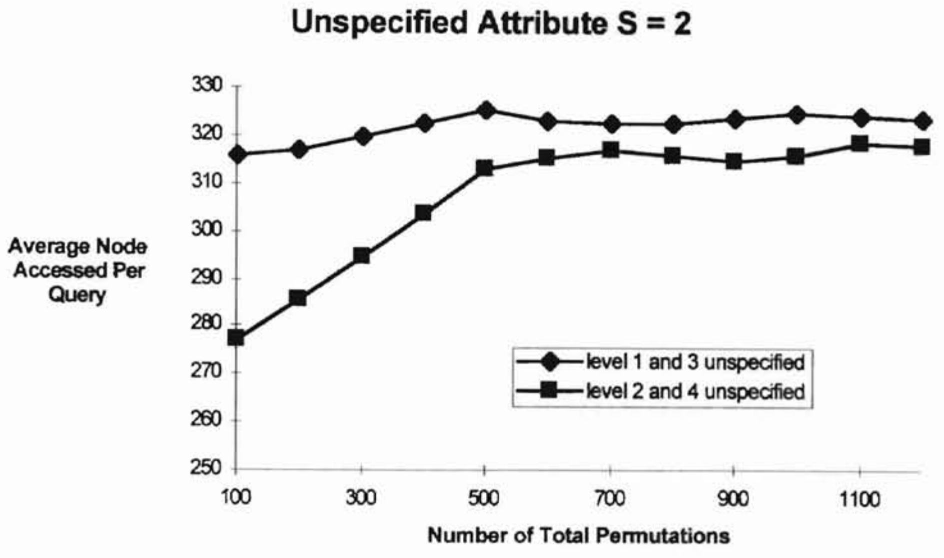

Figure 16. Average node accessed of unspecified attribute = 2

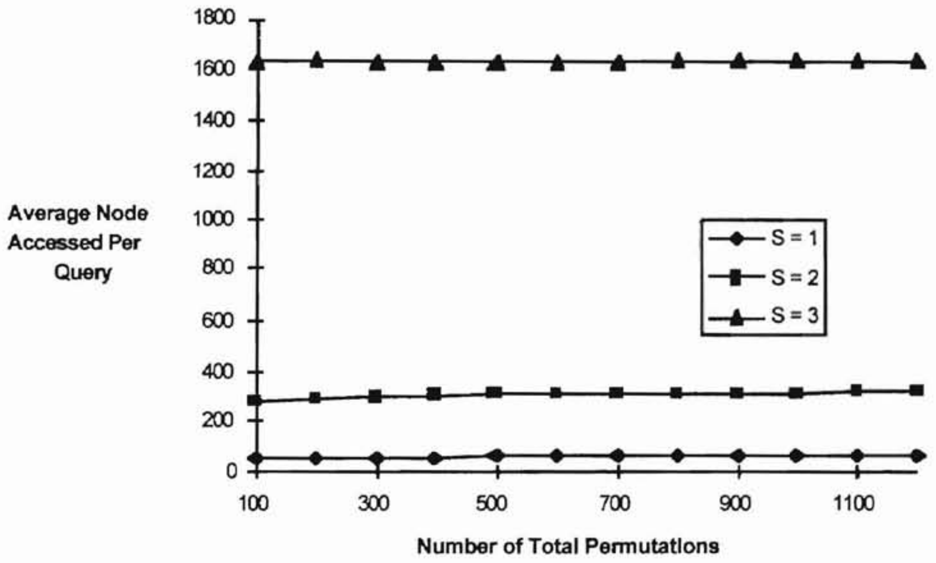

Figure 17. Comparison of average node accessed in different S

can tell that the average number of node accessed is a constant value if only one level is unspecified.

Figure 16 gives the results of unspecified attribute S = 2. More nodes are accessed when level 1 and 3 are unspecified. The reason is the same as in Figure 13. It shows that the average node accessed per query is constant when the number of total permutations is large enough.

The performance of Range_Match function is analyzed by testing the average number of node accessed/ total number of node per query, and the average retrieval time in different edge size and dimension size.

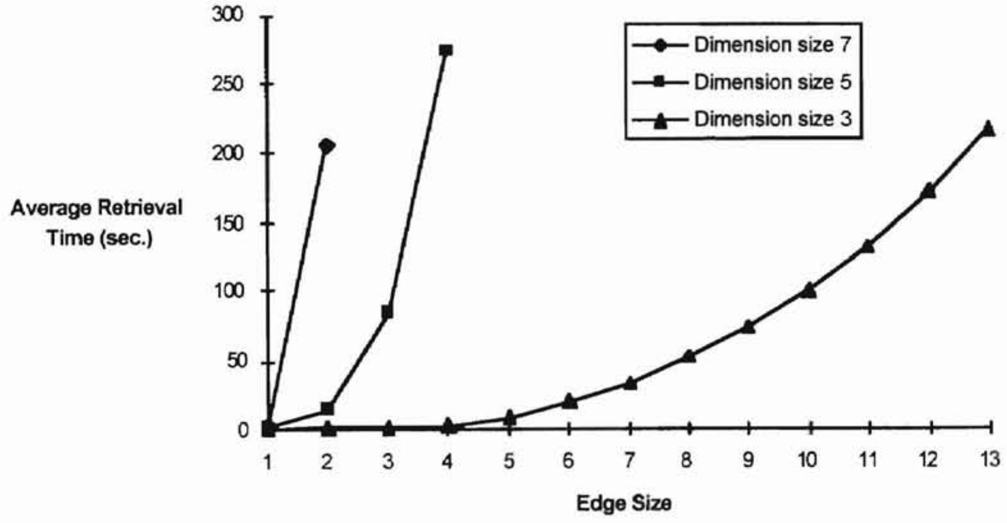

Figure 18(a). Experimental results of range match function

The experiment shows that the average number of node accessed equals the total number of node within the edge size. That means to retrieve a range query in a certain edge size, all nodes within edge size should be visited. This is because SPIN structure is a randomly organized tree structure. A new subscript combination is stored in the first

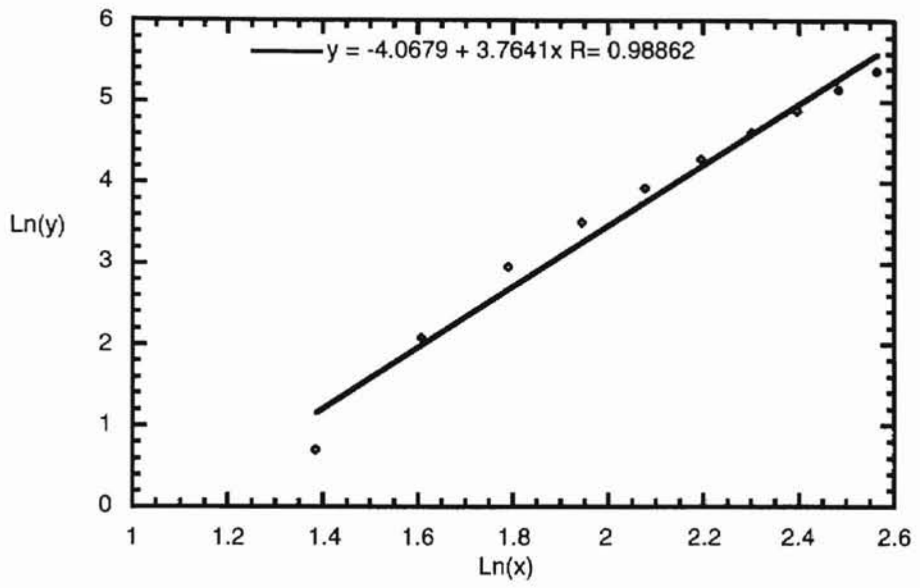

Figure 18(b). Analysis of Figure 18(a) dimension size 3

empty node in each level. When doing a range search, it cannot decide to go right or left path of a given node.3

Figure 18(a) is the average retrieval time for a range match query in different edge size where (0, 1) represents range search in [0, 0] to [1,1]. We represent this range as $[0,1]^2$. It shows the average retrieval time grows exponentially when the edge size increases from $[0, 1]^2$ to $[0, 13]^2$. Figure 18(b) is the analysis of the Figure 18(a). x and y denote the edge size and the average retrieval time of Figure 18(a) respectively. It uses a natural logarithm scale of x and y. ln(x) is linear with ln(y). The slope is 3.7641. We can get from Figure 18(b) that the average retrieval time grows as a power of edge size. The power is between 3 and 4.

CHAPTER VII

SUMMARY AND FUTURE WORK

In this thesis, a new spatial index technique – SPIN is discussed. After the discussion of various spatial index structures, SPIN functions show their advantages in handling multidimensional data both to speed data storage and retrieval time and to perform database operations in multidimensional data space. One of the disadvantages of the original SPIN functions is that they cannot retrieve different types of queries such as exact match queries, partial match queries and range match queries. We present a set of improved R_SPIN functions called P_SPIN functions to implement exact match queries, partial match queries and range match queries. Their algorithms are given and their performance are analyzed.

There are some assumptions in the experiments discussed in this paper:

1. There is no overflow situation during testing.
2. All insertions and searches are successful operations. If there are some unsuccessful insertions or searches, then the P_SPIN function will terminate at some level between level 0 and the leaf level.

The conclusions are:

1. The P_SPIN function can answer exact match query, partial match query amd range match query.
2. The time complexity of Exact_Match function and Partial_Match function has logarithmic behavior.
3. The SPIN function cannot answer range match queries efficiently.

The third conclusion can be explained by the comparison of the k-d tree and the SPIN structure searching method. A k-d tree is constructed as mentioned in chapter II. Searching in a k-d tree starts at the root node. When a node is visited whose discriminator is j^{th} key, the j^{th} range of the query is compared with the j^{th} key value. If the range query is above (or below) that value, only the right subtree (left subtree) of that node must be searched. If the query range overlaps the node's key, both of the subtrees should be searched. The construction of a SPIN structure is discussed in chapter III. Although the index values at each level are sequential, the key values in the same level are arbitrary. This causes the searching problem. When a node's key value in level i is compared with the i^{th} range of the query, it is impossible to decide whether to search its right subtree, left subtree or both of the subtrees. That why a range query will visit all nodes within the edge size.

Future improvement can be done to consider the overflow situation in SPIN operations and to analyze their theoretical time complexity. The SPIN structure will be more suitable as a database foundation if it can be modified to answer range match queries efficiently.

BIBLIOGRAPHY

[BAY77]
Bayer, R., and Schkolnick, M., "Concurrency of Operations on B-Trees" Acta Informatica, 9(1) 1977, pp. 1-21.

[BEC90]
Norbert Beckmnn, Hans-Peter Kriegel, Ralf Schnerder, and Bernhard Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles" Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 322-331, Atlantic City New Jersey, May 1990.

[BEN75]
Bentley, J.L., "Multidimensional Binary Search Trees Used for Associative Searching" Communications of the ACM, Vol. 18, No. 9, September 1975, pp. 509-517.

[BEN79]
Bentley,J.L., "Multidimensional Binary Search Tree in Database Application" IEEE Transactions on Software Engineering SE-5 (4), 1979, pp. 333-340.

[BEN79]
Bentley, J.L., Friedman, J. H.,"Data Structures for Range Searches" Computer Surveys, Vol. 11, No. 4, December 1979, pp. 397-409.

[BEN75]
Bentley, J.L., Stanat, D.F., "Analysis of Range Searchings in Quad Trees" Information Processing Letters, Vol. 3, No. 6, July 1975, pp170-173.

[BRE93]
Breene,L.A., "Quadtrees and Hypercubes: Grid Embedding Strategies Based on Spatial Data Structure Addressing" The Computer Journal, Vol. 36, No. 6, 1993, pp.563-569.

[CAR77]
Cardenas, A.F., Sagamang, J.P., "Doubly-Chained Tree Database Organization – Analysis and Design Strategies" The Computer Journal, Vol. 20, No. 1, 1977, pp.15-36.

[COB89]
Coburn, Ty C. "An Introduction to SPIN Hashing: an Approach to Managing Multidimensional Spaces" Tinker AFB, OK: Oklahoma City Air Logistics Center, n.d.

[COB91]
Coburn, Ty C. CSPIN toolkit (program documentation). Oklahoma City, OK: c1991.

[COM79]
Comer, D., "The Ubiquitous B-Tree" ACM Computing Surveys, 11(4), 1979, pp. 121-137.

[DAN86]
Dandamudi, S.P. and Sorenson, P.G., "Algorithms for BD trees" Software-Practice and Experience 16(12), 1986, pp. 1077-1096.

[DAN88]
Dandamudi, S.P. and Sorenson, P.G., "Performance Analysis of Partial-match Search Algorithms for BD trees" Software-Practice and Experience 18(1), 1988, pp. 83 - 105.

[FAL86]
Faloutsos, C., Sellis, T., "Analysis of object oriented spatial access methods" Proceedings of the 1986 ACM-SIGMOD International Conference on the Management of Data, San Francisco, v. 16, no. 3, pp 426-439.

[FLA86]
Flajolet, P., Puech, C., "Partial Match Retrieval of Multidimensional Data" Journal of the Association for Computer Machinery, Vol. 33, No. 2, April 1986, pp. 371-407.

[GUE90]
Guenther, O., Buchmann, A., "Research Issues in Spatial Databases" ACM-SIGMOD Record, Vol. 19, No. 4 Dec. 1990, pp 61 - 68.

[GUT84]
Antonin Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching" Proceedings of the 1984 ACM-SIGMOD International Conference on the Management of Data, pp 47-57, Boston, Massachusetts.

[KAS77]
R.L. Kashyap, S.K.C. Subas, S.B.Yao, "Analysis of the Multiple-Attribute-Tree Data-Base Organization" IEEE Transactions on Software Engineering, Vol. Se-3, No.6, November 1977 pp 451-465.

[LOM92]
LOMET, D., "A Review of Recent Work on Multi-Attribute Access Methods" SIGMOD RECORD, Vol. 21, No. 3, September 1992.

[NIE84]
J. Nievergelt, H. Hinterberger, and K.C. Sevcik, "The Grid File: an Adaptable, Symmertric Multikey File Structure" ACM Transaction on Database Systems, Vol. 9, 1, pp 38-71, 1984.

[OHS83]
Ohsawa, Y. and Sakauchi, M., "The BD-tree a new N-dimensional data Structure with High Efficient Dynamic Characteristics" Proceedings of the IFIP Conference. Paris, 1983, pp. 539-544.

[ROB81]
J.T. Robinson, "The K-d-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes" Proceedings of the ACM-SIGMOD International Conference on the Management of Data, pp 10-18, Apr 1981.

[SAM89]
Samet, H., The Design and Analysis of Spatial Data Structures, Addison-Wesley, Massachusetts, 1989.

[SAM90]
Samet, H., Applications of Spatial Data Structures: Computer Graphics, Image Proceeding and GIS, Addison Wesley, Massachusetts, 1990.

[SEL87]
Sellis, T., Roussopoulos, N., and Faloutsos, C., "The R^+ - tree: A Dynamic Index for Multidimensional Objects" Proceedings of the Thirteenth VLDB Conference, Brighton, England, Sep. 1987, pp 507 - 518.

[SUR86]
Suraweera, F., "Use of Doubly Chained Tree Structures in File Organisation for Optimal Searching" The Computer Journal, Vol. 29, No. 1, 1986, pp. 52-59.

VITA

YING FAN

Candidate for the Degree of

Master of Science

Thesis: SPIN: A MULTIDIMENSIONAL INDEX STRUCTURE FOR INFORMATION STORAGE AND RETRIEVAL

Major Field: Computer Science

Biographical:

Personal Data: Born in Shanghai, China, on December 14, 1964, the daughter of Fan, Gengzai and Lu, Peizhen.

Education: Received Bachelor of Science degree in Mathematics from FuDan University, Shanghai, China in July 1986; Received Master of Science degree in Applied Mechanics from FuDan University, Shanghai, China in July 1989. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1996.