

An Intelligent Data Structure Approaches for Mobile Devices

N.Mallikharjuna Rao¹

Dr. M.M Naidu²

P.Seetharam³

1. Associate Professor, Annamacharya P.G College of Computer Studies, Rajampet

2. Principal, S. V College of Engineering, Tirupathi, AP, India.

3. Systems Engineer, Annamacharya Institute of Technology and Sciences, Rajampet,

E-mails: drmallik2009@gmail.com , drgprrao@gmail.com and seetharam.p@gmail.com

Abstract - Mobile terminals like smart phones, mobile phones are becoming more and more popular and increasing day by day in recent years in India. However, the data on them is still managed in a way similar to that for traditional databases. This study proposes a method of file structure specifications in a fuzzy query database system which supports a search engine for component selection and which performs efficient and accurate queries. In this paper, we have developed a fuzzy database file structure frame work for mobile environment for solving transition between the terminals. A key requirement in such an environment is to support frequent connection and disconnection of database sites. We start with the overall architecture of the Fuzzy database, which is followed with some specific design issues like query, schema and transactions.

Keywords: Fuzzy Query, Fuzzy Database, Mobile network

I. INTRODUCTION

In recent years, the development of network technology, wireless communication technology, and corresponding software technology was accelerating; the performance of mobile phones and other mobile computing devices were increased.

Mobile applications run on mobile networks and allow users anywhere to use applications and information by handheld devices such as mobiles. In mobile computing environment, mobile application is an important trend of computing applications and development. Because of the combination of wireless networks and mobile devices, the share of mobile application will increase in the market, due to its convenience and flexibility. The characteristics of mobile computing and mobile applications are mobility, frequent disconnection, diversity network condition and so on. Because of these characteristics, the traditional distributed database technology can't effectively support the mobile computing environment.

In the current systems, data is stored as a file under a given file system. A file system is associated with a particular computer and its operating system. The data available on mobiles is highly unstructured and heterogeneous; therefore, the traditional way of accessing the data via file name of URL may not be adequate. Storing large number of meta-data in databases creates another management scale ability issue. As we are moving into the direction of accessing database on its semantics, how to store semantic information with data end how to allow data retrieval based on semantic become extremely challenging in recent trends.

While current file systems contain simple meta-data that map content to physical address. In this paper, we introduce Intelligent database system to add a rich mixture of additional meta-data, this additional content-specific information makes it possible for the data object itself to automatically initiate such actions as moving to remote locations for anticipated events, replicating in response to shifting usage pattern, or translating archived data to new formats as applications evolve, and assisting the information access via semantics.

In real world applications assuring precise data storage and retrieval is simplistic. When precise information is unavailable about the mini-world, it is often the case that some relevant information about the Universe of Discourse is often fuzzy in nature. Efficiency becomes even more crucial when dealing with fuzziness due to additional causes of performance degradation, such as the increased amount of storage, highly numerical nature of the proposed methods and, most importantly, inadequacy of the existing access structures. Thus research on fuzzy access structures is vital for efficiently querying fuzzy databases. The crucial difference between fuzzy queries and exact queries is the number of records brought into the memory.

A. The Characteristics of Mobile Database system

Mobile database is a kind of distributed databases supporting the mobile computing environment. Its data is dispersed in geography, but the data is concentrated in logic. Mobile databases system usually embedded in mobile phones and other mobile computing devices, then the user can connect to the server database for querying data and processing data by mobile terminals [1].

Mobile database system contains two meanings: first, the users can access the data of the server in the process of moving anytime and anywhere; second, users can take the copies of the server database in the process of moving. The distributed of data structure in the database system, can make the description of the inherent relationship among the data to modify, update and expand the data more easily, meanwhile ensuring the independence, reliability, security and integrity of the data, and reducing data redundancy. In this paper, we have described background in section 2, in section 3 presented several access approaches for fuzzy database and finally calculated performance and concluded in section 4 and 5.

II BACKGROUND

As we noticed that there are millions of subscribers adding in every day for mobile services in India. Subscribers are roaming from one place to another place receiving and making calls from and to several Mobile Terminals. If call is initiated then there are many things that are required to transfer from one location to another location. We are required to maintain database for all the transactions which are executed in mobile databases. In existing, using conventional databases for handling the mobile databases, the major problem facing by all the service providers in the telecom market is the transition of data retrieval and insertions. The conventional database will not support well to handle huge subscriber's database. In this study, we are proposing fuzzy databases for handling subscriber's databases.

In the conventional database model, a relation R of degree n and cardinality is composed of m columns and in n rows where; the columns denote the attributes and the rows denote the tuples. An attribute A , of tuple t takes values from domain D where the domains of R are $D_1, D_2, \dots, D_n$ a crisp attribute can take values of a single data type in relational database model, however, a fuzzy attribute can represent crisp and fuzzy values. For example, the attribute representing the monthly additions of the subscribers is a precise value such as 1000 or a fuzzy value such as medium. There are several extensions on employing fuzzy set theory to handle: imprecise data in relational database models [2].

A. The Fuzzy Query System

The access structure proposed by Bosc [4] is to use an index per fuzzy predicate tied to an attribute. The principle is to associate each grade of a fuzzy predicate with the list of tuples that satisfy that predicate. A possible efficient access structure for fuzzy data retrieval should be multidimensional. The reason for this is that non-fuzzy and fuzzy attributes need to be used as organizing attributes for the efficiency of both fuzzy and crisp queries. Given a record, a set of attributes which are chosen from the set of all attributes to organize the file, are called the organizing attributes. The selection of these attributes is usually based on types of common queries on the database and other physical database design. By utilizing a multidimensional structure the effectiveness of a fuzzy database is increases. For fuzzy database K-d Trees [3], grid files [4], Multi Level Grid File [5], and the other [9] may be utilized. In this paper, we substitute the same techniques for mobile database for reducing the transition and cost of retrieval of records from the mobile database.

The multidimensional binary search tree is known as k-d Tree and was first introduced by Bentley [3]. K-d Tree is a natural generalization of the well – known binary search tree to handle the case of a single record having multiple keys. K-d Tree requires relatively high pre-processing and storage cost when it is adapted to fuzzy indexing.

In the Grid File [5], any record is retrieved with two disk accesses; one disk access for retrieving the grid block and one for retrieving the data bucket. The size of the grid block is an important parameter. In addition, as records are added, deleted, and updated, the grid directory needs to be maintained by either splitting or merging that requires a considerable disk accesses. Grid File is good method for exact-match queries, range and fuzzy queries may require a considerable number of disk accesses.

The Multi Level Grid File [5] structure is utilized for fuzzy data access is also a multidimensional access structure that eases the drawbacks of Grid File by handling merge and split operation on directories locally without sacrificing the multi-attribute search capabilities of the Grid File. Multi Level Grid File structure is a flexible access structure for record insertion and deletion operations for fuzzy database. The proposed approaches are used for GSM and 3G-WCDMA key technologies.

III INDEX STRUCTURE APPROCHES FOR QUERY ACCESING FOR FUZZY DATABASES

In this paper, we are proposing three approaches for accessing fuzzy queries for mobile databases.

A. Grid File Structures

Traditional file structures are designed to handle single key access to speed up the querying process. However, the requirements of complex applications have made traditional access structures [4] ineffective. Grid File is a multi-dimensional base structure that divides file domain space into a k-dimensional space with k organizing attributes in a record. The structure consists of a grid directory (the root) and the data blocks (the leaves). The grid directory that is a dimensional array structure is a contiguous sequence of disk blocks. Each directory entry has a pointer to a grid region (i.e., data blocks). Each grid region is stored on a data bucket. More than one grid region can be stored in a data bucket since the data utilization may be too low if each bucket contains a single grid region. The grid regions assigned to a data bucket must form a rectangular partition of the attribute space. The grid directory and the data files are kept on a secondary storage. Linear scales for each dimension are stored in main memory. They are used to calculate the position of the directory entry. Therefore, an exact match query can be answered in two-disk accesses; one for the grid directory and the other for the data blocks.

When a data bucket overflows, it is split into two grids. The grid directory entry pointing to the split grid region must also be split. Consequently, if Grid file is k -dimensional. The split is performed along with $k-1$ dimension. Although the Grid File structure can be utilized effectively as long as the data distribution is uniform, the directory growth and the cost of directory split and merge operations can reach to an exponential rate when there is a correlation between the attributes. For example, suppose that we want a file of location in a city Rajampet and we

like to represent each location in a city by its attributes, latitude and longitude of the location of the city. The Grid File structure can be used to store the locations in a city together in a file by organizing the data by latitude and longitude values. The grid directory and data records of Grid File are shown in Figure 1.

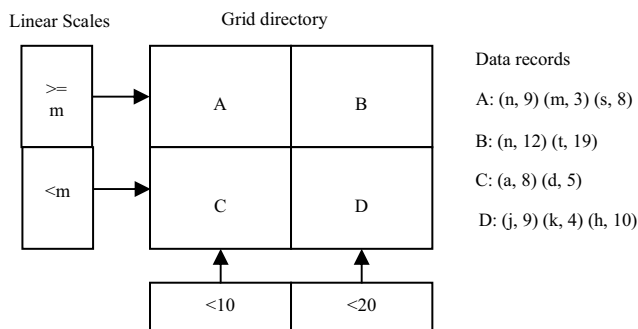


Figure 1: An Example of the Grid File Structure

B. Fuzzy Indexing

Conventional access structures [3] cannot be used directly since fuzzy predicates do not refer to entry values of the index. The proposed index structure per fuzzy predicate tied to an attribute. The principle of the index is to associate each grade of fuzzy predicate with a list of tuples that satisfy the predicates. This indexing schema results in many indexes per fuzzy predicate tied to an attribute except that some of the predicates issued from the same fuzzy primary term using modifiers (very, more or less,...) can use the same index. In this approach, an access on grades of membership is provided to a given primary term referring only to existing values of the attribute in concern. However, an additional level is added into the index for each primary term. This level contains the membership values of each fuzzy predicate. As an example, let us consider the fuzzy predicate (Mobile_Bill = 'high') addressing the crisp relation Subscriber (Name, Mobile No, IMEI, IMSI... Mobile_Bill) as shown in Figure 2. Using this index structure, it is also possible to access the subscribers satisfying more-or-less the predicate (Mobile_Bill = 'high').

We have the following observations about the fuzzy index structure as shown in Figure 2.

- This index structure only deals with homogeneous domains.
- Each index entry points to a record thereby leading to a large index structure.
- Each crisp value may have an index for each fuzzy term that increases the size of the index structure
- The duplicate data block address in the fuzzy indexing structure causes performance degradation in the retrieval of fuzzy query results.
- The fuzzy indexing structure does not impose any ordering of the data on the fuzzy attribute thus leading to a poor performance in fuzzy queries. The grouping of the records is achieved at the fuzzy Index level and ignored at the physical database level.

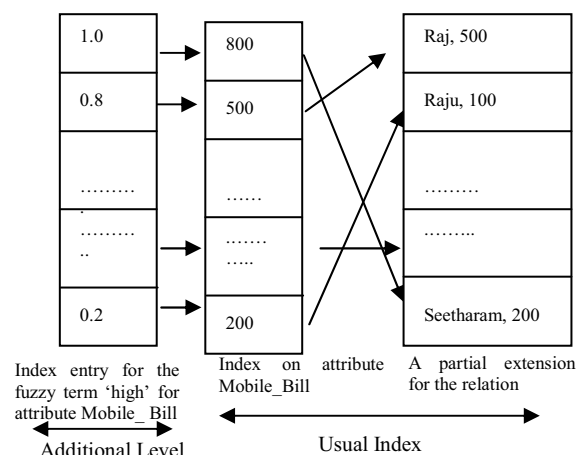


Figure 2: A Conceptual Schema of the Index Method

This index structure is developed for indexing on crisp relations, although it is argued that it can be extended to the case where the attributes have 'ill-known' values and represented as a possibility distribution.

Although this index schema somewhat speeds up the fuzzy querying process with respect to sequential scan, the storage costs increase significantly.

In many cases, the homogeneous fuzzy domain assumption is restrictive. For example; consider the attribute Mobile_Bill, sometimes we may not detect the exact Mobile_Bill of Subscriber and give it a fuzzy value like high. On the other hand, often we know the Mobile_Bill of Subscriber; and thus store a crisp value: say Rs. 2000. For the case of fuzzy attributes with homogeneous domains taking on only fuzzy values (i.e., low, medium, high), we first determine the fuzzy predicates satisfying the stated conditions and the given threshold by using the similarity relations in order to execute a fuzzy query. Then a search through the index is done and the tuples with those pre-detected fuzzy values are retrieved. On the other hand, when the attributes have only crisp data, membership functions are used to determine the required ranges to carry out a fuzzy query. Then, a range search is done on the database. If an attribute is allowed to take both fuzzy and crisp data; that is, it has heterogeneous domain, we need to have an approach which combines these in a consistent manner.

Our access structure described in the coming section handles both crisp and fuzzy data with a single access structure.

C. The MLGF Data Structure for Fuzzy Databases

We assume that the database contains fuzzy attributes that may consist of both crisp and fuzzy values and that the database model is a similarity based extension of the relational database model. The queries can be crisp as well as fuzzy. As pointed out previously, the existing fuzzy indexing approaches revolve; on building structures to index fuzzy data. However, a fuzzy query can be mixed with crisp queries. Therefore, a multidimensional structure that can allow both the non-fuzzy and fuzzy attribute to be

indexed together will increase the effectiveness of fuzzy database while decreasing the storage costs incurred by additional index structures. Before we discuss how the Multi Level Grid File (MLGF) structure is utilized as an access structure for fuzzy databases, we should first describe the MLGF data structure.

1) Multi Level Grid File

Multi Level Grid File (MLGF) [5] is a data structure which used for mobile database for illustrating access of records, storage of mobile based data. The grid directory is maintained as a multi-level structure where each directory entry points to a lower level directory block. In MLGF, splitting and merging a directory is performed locally, thereby decreasing the amount of I/O that is required for a global split or merge. This causes the structure to be flexible during record insertion and deletion operations on mobiles. Besides, empty directory entries do not take place in MLGF; therefore, the directory size is more compact compared to Grid File. The organizing attributes are turned into bit patterns. Bit patterns of each organizing attribute are merged to form a key (usually hashed) bit pattern. In MLGF a directory entry is formed of a region vector and a pointer [6]. The region vector is a composite bit pattern of the hashed bit patterns of the organizing attributes.

The MLGF structure in Figure 3 represents an organizing-attribute file and its two-dimensional (logical) grid, where the physical blocks are shown by dotted inner rectangles as shown in figure 4. The MLGF structure in the figure is composed of two levels, D_1 and D_2 , where D_2 is the directory for the file D_1 and D_1 is the directory for the data file itself. There are three entries in the directory D_1 is shown in Figure 4. An entry (00, 0) in D_1 represents all the records whose (hash) values of the first attribute start with 00, and whose (hash) values of the second attribute start with 0. We should also note that the empty blocks (e.g. (0, 01), (1,1)) are not represented anywhere in this structure and each physical block can represent a region of varying size, although only one entry in the directory corresponds to that block. The MLGF structure employs a consistent representation at all directory levels. It stores hashed bit strings; thus, resulting in a compact directory. The directory size of MLGF grows in a linear manner. In case the domain of a particular dimension is completely represented, the '-' symbol is used.

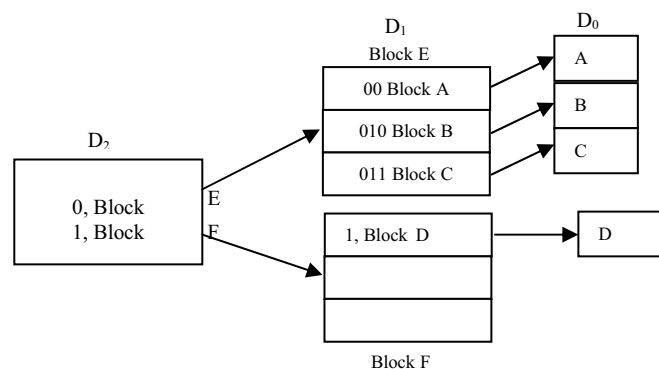


Figure 3: Directory structure in Multilevel Grid File Structure

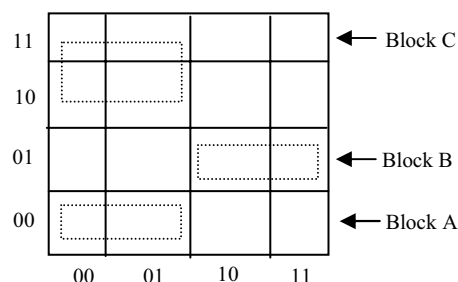


Figure 4: The Space partition to the MLGF

2) Representation of the Record and Access Structure for Fuzzy Databases in MLGF

The directory entry is the basic building block of the adapted MLGF structure that we used here as an access structure; for fuzzy databases on mobiles. It contains a composite bit pattern formed of bit patterns of organizing attributes. An array contains the number of bits used for each bit pattern of the organizing attributes. Each non-empty directory entry points to a directory block. If the directory entry is a leaf level directory entry, it points to a data block. A diagrammatic representation of the directory entry on fuzzy database for using on mobile databases can be shown in figure 5 as follows:

Length of Organizing Attributes	Number of records	Composite bit patterns of organizing attributes	Pointer to data block/ directory block
---------------------------------	-------------------	---	--

Figure 5: Representation of the record and access structure for Fuzzy Databases

For efficiency reasons in querying, a bit value is used to show whether the data in the record or in the crisp (a bit value of one (1) is used in this case) or fuzzy (a bit value of zero (0)). Thus crisp data is distinguished from fuzzy ones and this information is used in querying when they needed, especially for handling crisp exact match and range queries. Finally, the hit representation of the original data is appended at the end of by the similarity matrix shown in figure 6 of the fuzzy set short with the fuzzy sets medium and high must also be taken into account. For the fuzzy set medium, the similarity value "0.6" is determined from the similarity matrix. This value is multiplied by the membership degrees of the partition tree for the fuzzy set medium. Then, the following partition tree in Figure 6 is reformed for this fuzzy query.

In fact, one does not need to compute the complete partition tree of the fuzzy term medium. The algorithm is designed for generating the partition trees in a way that only a part of the partition tree satisfying the condition specified in the query is reformed more specifically. The level value at the top of the partition tree is computed and compared against the threshold value. If it qualifies, the search moves to the next levels of the partition tree and computes those values. For example, if the threshold value were 0.5 in our query. The search would go to the next

level of the partition tree. The right partition value 0.47 would not qualify, so it would be prune from the search tree and no more computation would be done on that part of the tree. The query would be translated to "retrieve the subscriber in the fuzzy set medium for the whole left partition and the middle partitions", which corresponds to the (0100, 0101) hit patterns. The same approach is followed when determining the partitions to search for the fuzzy set high. Note that, as the threshold gets lower, the number of partitions to him searched increases.

If a fuzzy query intermingled with a fuzzy modifier arrives, then the effect of the fuzzy modifier is taken into account for determining the partitions. For example, at least we take the case of the fuzzy modifier "very", this performs a concentration operation. If the query is "retrieve very short subscriber with a threshold of 0.7", then the partition tree for the fuzzy set short is reformed as shown in Figure 6.

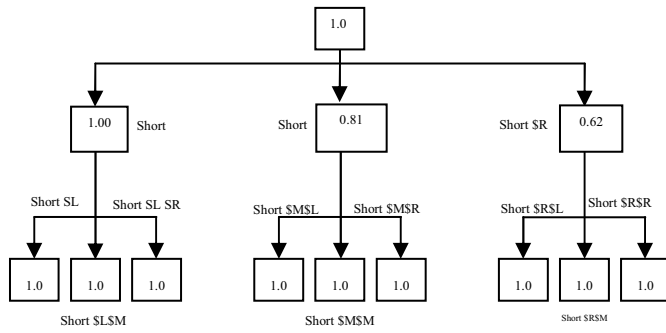


Figure 6: Fuzzy Membership partition tree

D. Evaluation of Queries on fuzzy databases

First of all, let us give a brief definition of fuzzy sets. Let Ω denote our universe of discourse. We can define a fuzzy set X by a membership function μ_X of the form:

$$\mu_X : \Omega \rightarrow [0,1]$$

In contrast to crisp sets, in which an element of Ω is either a member of a given set or not, fuzzy sets assign varying degrees of membership to elements of Ω . Membership functions, as defined above, can also be seen as possibility distribution functions. We discuss the relevance of these functions in a moment.

We assume n data items $(o_1, o_2, o_3, \dots, o_n)$ in our database, all of which have an attribute A with domain Ω whose value is described by a possibility distribution function $\pi_{A(o_i)}$. We also assume that Ω is discrete. If this is not the case for a given application, we have to discretize the membership function. The reason for this is that we want to apply access methods for discrete crisp sets when indexing fuzzy sets.

If at least one $\omega \in \Omega$ exists, such that possibility distributions function $\pi_{A(o_i)}(\omega)$ is equal to 1 ($\pi_{A(o_i)}(\omega)$ said to be normalized in this case), we can define a possibility measure on Ω as follows:

$$\forall X \in \rho(\Omega) \pi(x) = \max_{\omega \in X} \pi_{A(o_i)}(\omega) \quad \text{Prade and}$$

Testemale developed a model based on possibility and necessity measures for querying fuzzy data.

Our goal is to support queries involving attributes described by possibility distribution functions. We look at queries in which attributes are compared to constants. Both the comparison operator θ and the constant a are represented by membership functions μ_θ and μ_a , respectively.

The Comparisons operator for "approximately equal to" could be defined as follows:

$$\mu \approx (\omega, \omega') = \begin{cases} 1 - |\omega - \omega'| & \text{for } |\omega - \omega'| \leq 1 \\ 0 & \text{else} \end{cases}$$

In our example α is equal to $\mu_{\text{medium-sized collection}}$ and θ is equal to $\mu_{\approx}$.

The possibility that the value of attribute A of data item o_i belongs to the set of values determined by θ and α is equal to

$$\pi(\alpha \theta | A(o_i)) = \max_{\omega \in \Omega} \min(\mu_{\alpha \theta}(\omega), \mu_{A(o_i)}(\omega))$$

With

$$\mu_{\alpha \theta}(\omega) = \max_{\omega' \in \Omega} \min(\mu_\theta(\omega, \omega'), \mu_\alpha(\omega'))$$

The necessity of belonging to this set is equal to

$$N(\alpha \theta | A(o_i)) = \min_{\omega \in \Omega} \max(\mu_{\alpha \theta}(\omega), 1 - \pi_{A(o_i)}(\omega))$$

1) Query Evaluation

Using above expressions is very unwieldy in terms of calculating them efficiently. Bosc and Galibourg show in [7] how to simplify the evaluation of these expressions. A data item o_i belongs to the set of data items possibly satisfying the query, iff

$$\pi(\alpha \theta | A(o_i)) > 0 \Leftrightarrow L_{>0}(\pi_{A(o_i)}) \cap L_{>0}(\mu_{\alpha \theta}) \neq \emptyset$$

Where $L_{>0}$ are α -cuts of fuzzy sets. An α -cut of a fuzzy set F is defined as ($0 \leq \alpha \leq 1$):

$$L_\alpha(\mu_F) = \{\omega \in \Omega \mid \mu_F(\omega) \geq \alpha\}$$

We talk of strict α -cuts whenever

$$L_{>\alpha}(\mu_F) = \{\omega \in \Omega \mid \mu_F(\omega) > \alpha\}$$

There are two special α -cuts, the core $L_1(\mu_F)$ and the support $L_{>0}(\mu_F)$ of a fuzzy set F .

For more selective queries an acceptance threshold α can be provided by the user. Determining qualifying data items then boils down to

$$\begin{aligned} \pi(\alpha \theta | A(o_i)) > \alpha &\Leftrightarrow L_\alpha(\pi_{A(o_i)}) \cap L_\alpha(\mu_{\alpha \theta}) \neq \emptyset \\ \Rightarrow L_{>\alpha}(\pi_{A(o_i)}) \cap L_\alpha(\mu_{\alpha \theta}) &\neq \emptyset \end{aligned}$$

In both cases the appropriate α -cut of $\mu_{\alpha\theta}$ can be calculated beforehand and the supports of the fuzzy sets describing the attributes values of the data items can be used in filtering data items during query evaluation.

The case for the necessity measure is handled similarly. A data item O_i belong to the set of data items necessarily satisfying the query, iff

$$N(\alpha\theta \mid A(O_i)) > \alpha \Leftrightarrow L_1(\pi_{A(O_i)}) \subseteq L_{>\alpha}(\mu_{\alpha\theta})$$

For an acceptance threshold of α we get

$$N(\alpha\theta \mid A(O_i)) > \alpha \Leftrightarrow L_{1-\alpha}(\pi_{A(O_i)}) \subseteq L_{>\alpha}(\mu_{\alpha\theta})$$

$$\Rightarrow L_1(\pi_{A(O_i)}) \subseteq L_{\alpha}(\mu_{\alpha\theta})$$

E. Performance of the MLGF Access Structure for Fuzzy Queries

The software design includes several modules designed so that the overhead in function calls is held at minimum. These modules are:

- User Input/ Output Module,
- MLGF Operations Module
- Transform Operations Module
- Query Module
- Partitioning free Module
- Database Operations Module

In this paper, we used algorithms proposed by Bosc for searching database, updating and deleting the database records on mobile data.

In order to measure the performance of the file organization of MLGF access structure for mobile database studied here, the complexity analysis using the asymptotic notation for various queries on thus access structure shall be stated. Thus, the asymptotic growth of the directory is $O(n)$. The estimated number of levels in the adapted MLGF is as $(\log_b n)$ where b is the number of directory entries in a physical block (directory blocking factor).

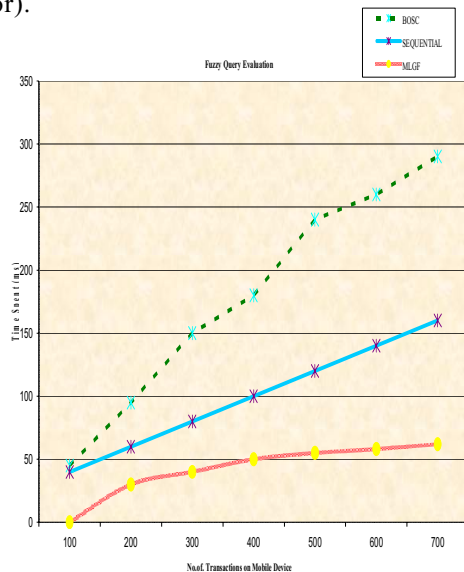


Figure 7: Comparison Graph for Index approaches

If the number of accesses, c are included to the main data blocks, then the number of block access for a fuzzy query in worst-case would be $c_1(\Gamma \log n) + c_2$ where c_1 and c_2 are constant. This is, in terms of the asymptotic growth is $O(\log_b n)$.

We compared the performances in terms of time (msec) and member of I/O blocks retrieved for fuzzy queries from fuzzy databases. The tests are performed to compare the performances of fuzzy querying in MLGF partitioning schema with the sequential method as well as the Bosc indexing method, resultant as we shown in figure7. We found that for poor performance of the Bosc index is the existence of more duplicated data block address values in this fuzzy index structure. For profiling the performance of the adapted MLGF as a fuzzy access structure in fuzzy databases on mobile databases, a series of tests have been performed. The tests are performed on MATLAB with a light load. All tests are performed with a data block set of records.

CONCLUSIONS

In this paper, we are introducing a record and a file organization schemes for fuzzy databases along with a fuzzy access structure by adopting the Multi Level Grid File structures, grid file and fuzzy indexing, a multidimensional data structure for mobile databases. The proposed method is effectively uses fuzzy queries as well as crisp query for mobile databases. The storage cost of Multi Level Grid File fuzzy access structure is advancement towards efficient querying in fuzzy databases.

REFERENCES

- [1] <http://www.dcs.gla.ac.uk/Keith/Chapter.4/Ch.4.html>
- [2] Li Ye-bai, Zhang Bin and Hai-bin, "Study and Design on data management Model for SQL server CE for mobile application", IEEE, 2010 International Conference on e-Education, e-Business and e-Learning.
- [3] Bentley J.L., "Multidimensional Binary Search Trees in database application", IEEE Trans. On Software engineering V.SE-5, no 4, 333-340
- [4] Sven Helmer, "Indexing Fuzzy data", IEEE, page no: 2120-2124, 2001
- [5] Kyu-Young Whang, "The Multilevel Grid file - A dynamic hierarchical dimensional file structure", Database systems for advanced applications, World scientific publishing & Co.
- [6] Glagowski, Terry G., and Kent L. Jones, A new method for implementing Fuzzy Retrieval from a Spatial Database, School of Electrical Engineering and Computer Science, Washington state University, Pullman, Washington ,1995, pending publications in Information sciences.
- [7] Bosc P. M. Galibourg and H Hanson, "Fuzzy querying with SQL: Extensions and implementations aspects", Fuzzy Sets and Systems, 28, 333-349
- [8] Bosc P and O Pivert, "Fuzzy Querying in conventional database ", Fuzzy logic for management of uncertainty , edited by LA Zadeh and J Kacprzyk Johan Wiley and sons inc 645-671
- [9] Yazici A and D Cibiceli , "An index structure for fuzzy database s", The proceedings of the 5th IEEE international conference on fuzzy systems, 1375-1381 New Orleans, USA
- [10] Rotem D and Segar A, "Algorithms for multi dimensional partitioning of static files", IEEE Trans on software engineering 16(11) 1700-1710.