Spatio-Temporal Range Search using K-DSTH Indexing Structure

Sumeet Gill, Meenakshi

Abstract: ***A large amount of data which includes spatial and temporal information related to different fields like geography, satellite, medical or multimedia is generated and collected at an extraordinary scale. Such data is produced by satellites, mobile devices, emerging applications like social networking sites, photo sharing sites and many more. As the whole world is aware of the importance of such spatio-temporal data, a great amount of research work is evolving around efficient storage structures and algorithms to handle a variety of spatio-temporal queries. In this paper, the authors are introducing a novel spatio-temporal indexing structure k-dStH which is an extension of k-dSTHash indexing structure. It uses master hash table, local hash table and B-tree additionally which are based on timestamp values. The researchers also introduce an algorithm k-dStHSpaTempRangeSrch based on the proposed indexing structure to find spatio-temporal objects in given spatial range at particular temporal value. The performance analysis shows that the algorithm proposed by the authors is far more efficient as compared to brute force technique of searching for the spatio-temporal objects.***

Keywords: Brute force, B-Tree, Hash Table, Indexing, k-d Tree, Spatio-temporal Dataset, Spatio-temporal Range Search.

I. INTRODUCTION

Spatio-temporal range search is one of the essential and important areas in case of computational geometry. Spatial range search is to retrieve all data records from a dataset/database which fulfill the provided range restrictions for given set of dimensions. These might include report problem in which details of objects satisfying the query are reported or count problems where just the numbers of retrieved objects is reported. The spatio-temporal range search queries retrieve for the records based on both spatial and temporal range restrictions. Also, there can be queries to retrieve spatio-temporal objects within a given range at particular time. In this paper, the authors propose a novel indexing structure k-dStH to index spatio-temporal data and an algorithm k-dStHSpaTempRangeSrch to search for spatio-temporal objects within given range at given time. Experimental evaluation of the algorithm clearly demonstrates the efficiency of the proposed structure.

Revised Manuscript Received on October 15, 2019.

Sumeet Gill, Department of Mathematics, M. D. University, Rohtak, Haryana, India. Email: drsumeetgill@gmail.com

Meenakshi, Department of Mathematics, M. D. University, Rohtak, Haryana, India. Email: mshthebest@gmail.com

II. RELATED WORK

In this day and age, the indexing structures and techniques to index spatio-temporal data competently are attaining a vast amount of insight and attention of researchers, academicians and engineers [1]. Some of the well-accepted indexing structures to save and retrieve information about spatial objects are Grid files [2], R+ trees [3], K-D-B trees [4] and Quad-trees [5]. These structures are also used as base sturctures of lots of spatio-temporal indexing structures. [6] presents an efficient graph-based spatio-temporal indexing method for task-oriented multi-modal scene data organization. This research work proposed spatio-temporal index which is multi-modal and multi-level hybrid in design. It consists of two parts- a local index and a global index. There are many fine-grained indexes in local index. These fine grade indices include B+-tree, Quad-tree, R+ -tree, graphs and hashing. Main memory and external storage hold these indexes and used for proficient scheduling of data related to computing intensive and I/O-intensive tasks. The global index is based on graph and manages the time, semantics and links between data objects and features of multi-modal scene. [7] designs and presents a hierarchical information quadtree for efficient spatial temporal image search for multimedia stream. It is spatio-temporal search system for images. The system is composed of three components-preprocess module, update module, and query modules. The job of preprocess module is to get the incoming spatio-temporal image, to extract geo-temporal image's location, and then sends every geo-temporal image and its location to the update module. Location can be precise coordinates i.e. latitude and longitude or it can be center point of MBR i.e. Minimum Bounding Rectangle. The update module confirms timely insertion of every incoming spatio-temporal image in memory indexes. It also checks that every incoming spatio-temporal image query is answered correctly and efficiently using in-memory indexes. The query module employs spatio-temporal visual pruning methods which lessens the count of visited images for returning final result.

III. BRUTE FORCE METHOD

In this section, we are explaining the brute force method to search for spatio-temporal data in given spatial range at particular time. Brute force technique of searching for required objects is the algorithm usually used for its simplicity as no domain knowledge is required to implement it.

There is no focus on the improvement of performance, tries out all possible combinations and relies on the power of computing absolutely. Brute force method is extremely straightforward and can implement to search for any kind of data. We have implemented this method using linear linked list to store the data from spatio-temporal dataset under analysis. Algorithm I, **bFSpaTempRangeSrch** presents the method of searching for spatio-temporal objects in given spatial range at particular time using brute force method of searching.

Algorithm - I: bFSpaTempRangeSrch

(To search for spatio-temporal objects using Brute Force Method)

Algorithm	int **bFSpaTempRangeSrch** (struct bruteForceStr* HEAD, POINT coordinatesAboutWhichToSearch, RANGE withInRange, TIME timeStampToSearch)
Inputs to Algorithm	**HEAD [type-struct bruteForceStr*]:** Starting pointer of the Structure **coordinatesAboutWhichToSearch [type- POINT]:** N-dimensional queryPoint about which objects are to be found **withInRange [type-RANGE]:** Spatial range within which the objects are to be searched **timeStampToSearch [type- TIME]:** Time at which the objects are to be searched
Output from Algorithm	**NO_DATA [type-int]:** When structure is empty or **fndWithInRangeCount [type-int]:** Number of spatio-temporal objects which satisfy the conditions **resultList [type-rsltLinkLst]:** Details of spatio-temporal objects which satisfy the conditions

```
BEGIN

    IF HEAD is  NULL
                THEN
                     resultList ← NULL
                     return NO_DATA
                END IF

    SET TEMP ← HEAD
                COUNT ← 0

    LOOP WHILE TEMP != NULL
    DO
                    for every dimension dim repeat
          distanceSquare        +=      square      (record      [dim]      -
coordinatesAboutWhichToSearch [dim])
          end for

          IF (distanceSquare  < Square(withInRange))
                    THEN
                       IF timestamp epoch value of node TEMP match
                       with epoch value of timeStampToSearch
                              THEN
                        increment fndWithInRangeCount by 1
                        add record to resultList
                              END IF
          END IF

         distanceSquare = 0
         Update TEMP to point to next node of List with
spatioTemporalDataRecord
                END WHILE

                return fndWithInRangeCount
END
```

The algorithm **bFSpaTempRangeSrch** receives a pointer *HEAD* of type *struct bruteForceStr*, *coordinatesAboutWhichToSearch* of type *POINT*, *withInRange* of *RANGE* type and *timeStampToSearch* of *TIME* type. Also, a pointer to *resultList* of type *rsltLinkLst* is made available as a global variable to store details of retrieved spatio-temporal objects. It there is no record in the list, *NO_DATA* is returned back to notify the same. If records exist, then the whole list is traversed and distance of every traversed object from *coordinatesAboutWhichToSearch* will be calculated. Square of distances for each dimension is considered for calculating *distanceSquare*. If *distanceSquare* is less than the squared value of range then the temporal value of current object is compared with queried *timeStampToSearch*, and, if it matches then *fndWithInRangeCount* is incremented by 1 and record is inserted to *resultList*. When the whole dataset is processed, *fndWithInRangeCount* will contain the number of spatio-temporal objects found within given range at particular given time and *resultList* will be holding details of every spatio-temporal object satisfying the queried conditions. If no record satisfies the query conditions, *fndWithInRangeCount* is returned as zero and nothing is inserted in the *resultList*. The disadvantage of this technique is that the researchers need to compare each and every object's details with queried *coordinatesAboutWhichToSearch* and *timeStampToSearch*. It takes lot of time and resources and is not an efficient way to organize and search for data especially spatio-temporal data.

IV. PROPOSED SPATIO-TEMPORAL INDEXING STRUCTURE: K-DSTH

The researchers are proposing a spatio-temporal indexing structure **k-dStH** in this research work. This structure is a combination of k-d tree, B-Tree and Hash table. Hash table and B-Tree are maintained at two levels – Master i.e. global level and Local level. This indexing structure is a data structure which can index duplicate spatio-temporal key datasets in an efficient way. k-d tree is used to organize records on the basis of spatial data using n-dimensional spatial coordinates, local hash table at every k-d tree node is used to store spatio-temporal record at particular spatial coordinate on the basis of epoch value of

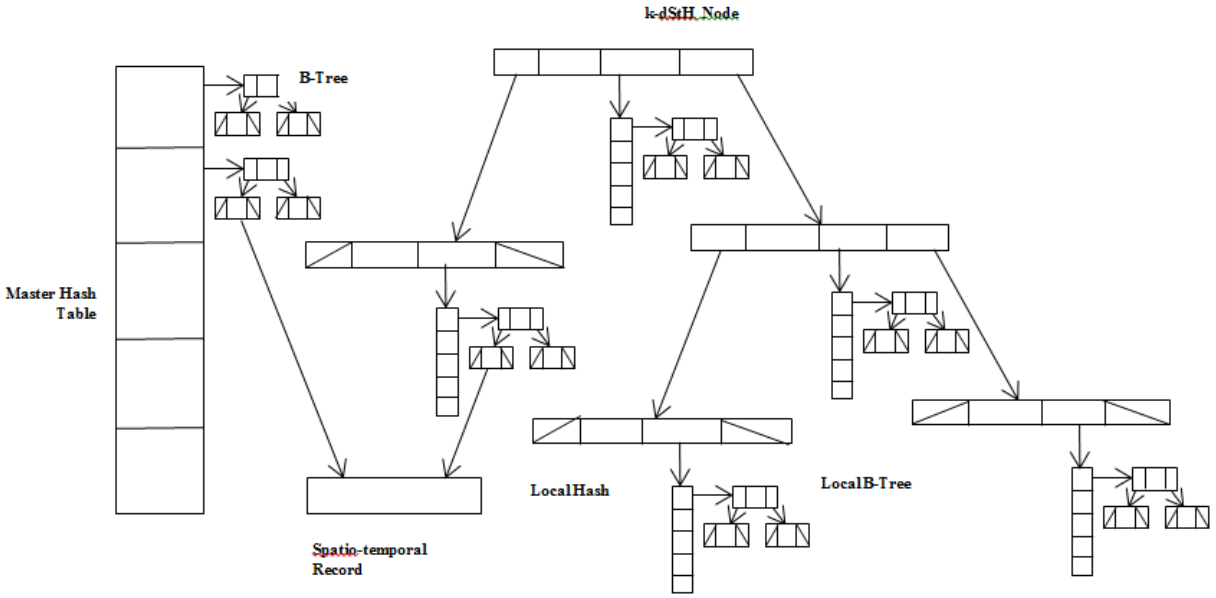

Fig. 01: k-dStH Indexing Structure

timestamp and a local B-Tree is attached with local hash table to store spatio-temporal objects at that particular spatial node for holding records with hash keys based on epoch value of timestamp.

The Master Hash Table keeps record of every spatio-temporal record according to hash keys based on timestamp values without worrying about spatial attributes. It is used to retrieve the results of queries based on temporal values only. To retrieve the results related to spatio-temporal queries, first k-d tree is consulted and then hash key for temporal value is generated to find the required objects using Local Hash Table. B-Tree linked with Local Hash Table organizes records with same temporal keys but at more refined level. While organizing spatio-temporal records in **k-dStH** indexing structure, the pointers to the spatio-temporal records are inserted at proper places in B-Trees in both Master Hash Table and Local Hash Table.

V.SPATIO-TEMPORAL RANGE SEARCH USING K-DSTH INDEXING STRUCTURE

In this section, the authors are proposing an algorithm to search for the spatio-temporal objects within given range about given spatial location at particular temporal value in **k-dStH** indexing structure. The algorithm starts traversing at the root of **k-dStH** indexing structure and keeps on pruning the sub-tree if bounding box of the sub-tree doesn't intersect with the requirements of range query. If bounding box of sub-tree falls entirely within requirements of the range query, it will save all objects of sub-tree in the result list. If bounding box of sub-tree overlaps range query requirements then the algorithm recurses left and right.

The algorithm is divided into two sub-algorithms where first sub-algorithm is the wrapper for second sub-algorithm. The algorithm II (a) **k-dStHSpaTempRangeSearch** receives a pointer *kdS_RootNode* to the root node of **k-dStH** indexing structure of *type struct k-dStH*, *rangeWithinWhichToSearch* of *Distance* type and pointer *timeStampToSearch* of type *timestamp*. It is a wrapper algorithm to initialize *withInRangeResultList*, call another algorithm **findWithInRangeSpaTemporal** and return *withInRangeiResultLst*.

Algorithm - II (a): k-dStHSpaTempRangeSearch (To search for spatio-temporal objects in k-dStH Indexing Structure)

Algorithm - II (a)	struct kdSLstRgSrResult* **k-dStHSpaTempRangeSearch (**struct k-dStH * **kdS_RootNode**, Location queryPoint, Distance rangeWithinWhichToSearch, timeStamp* timeStampToSearch)
Inputs to the Algorithm	**kdS_RootNode [type-struct k-dStH*]:** Root node of k-dStH index Structure. **queryPoint [type-Location]:** N-dimensional queryPoint about which objects are to be found. **rangeWithinWhichToSearch [type-Distance]:** Range within which objects are to be searched. **timeStampToSearch [type-timeStamp*]:** Temporal value on which the objects are to be searched.
Output from Algorithm	**withInRangeResultList [type-struct kdSLstRgSrResult*]:** List to store the result i.e. object details which qualify the query.
Algorithm	struct kdSLstRgSrResult* **k-dStHSpaTempRangeSearch** (struct k-dStH *kdS_RootNode, Location coordinatesAboutWhichToSearch, Distance rangeWithinWhichToSearch, timeStamp* timeStampToSearch) BEGIN Initialize withInRangeResultList Call findWithInRangeSpaTemporal(kdS_RootNode, coordinatesAboutWhichToSearch, rangeWithinWhichToSearch, withInRangeResultList, kdS_dimension, timeStampToSearch) return withInRangeResultList END

Algorithm - II (b): findWithInRangeSpaTemporal (To search for spatio-temporal objects in k-dStH Indexing Structure)

Algorithm - II (b)	int **findWithInRangeSpaTemporal**(struct k-dStH *k-dStH_node, Location coordinatesAboutWhichToSearch, Distance range, struct resultList *withInRangeResultList, int currentDimension, timeStamp* timeStampToSearch)
Inputs to the Algorithm	**k-dStH [type- struct *k-dStH_node]:** Tree/Sub-tree root node. **coordinatesAboutWhichToSearch [type- Location]:** N-dimensional queryPoint about which objects are to be found. **rangeWithinWhichToSearch [type- Distance]:** Range within which objects are to be searched. **resultList [type - struct *withInRangeResultList]:** List to store the result i.e. object details which qualify the query. **currentDimension [type – int]:** Dimension to be considered for current node **timeStampToSearch [type-timeStamp*]:** Temporal value on which the objects are to be searched.
Outputs from Algorithm	**foundWithInRangeCount [type - int]:** Number of objects found within given Range of queried Location. **withInRangeResultList [type - struct resultList *]:** List to store the result i.e. object details which qualify the query.

Algorithm	int **findWithInRangeSpaTemporal**(struct k-dStH *k-dStH_node, Location coordinatesAboutWhichToSearch, Distance rangeWithinWhichToSearch, struct resultList *withInRangeResultList, int currentDimension, timeStamp* timeStampToSearch)
	
	BEGIN
	
	if k-dStH_node is NULL
	then
	return 0
	end if
	
	distanceSquare = 0
	for every dimension 0 to maxDimensions repeat
	distanceSquare += square of difference in coordinatesAboutWhichToSearch and k-dStH_nodePoint for currentDimension
	end for
	
	if distanceSquare is less than or equal to square of rangeWithinWhichToSearch
	then
	if records exist at hash key index in local hash table based on epoch value of timeStampToSearch
	then
	Traverse local B-Tree to search for exact epoch value of timeStampToSearch
	if record is for queried timeStampToSearch
	then
	add k-dStH_nodePoint to withInRangeResultList
	increment foundWithInRangeCount by 1
	end if
	end if
	end if
	
	distanceDiff = difference in coordinatesAboutWhichToSearch and k-dStH_nodePoint for currentDimension
	
	if distanceDiff is less than or equal to 0
	then
	returnedCount = CALL findWithInRangeSpaTemporal(k-dStH_nodeLeft, coordinatesAboutWhichToSearch, rangeWithinWhichToSearch, withInRangeResultList, currentDimension)
	else
	returnedCount = CALL findWithInRangeSpaTemporal(k-dStH_nodeRight, coordinatesAboutWhichToSearch, rangeWithinWhichToSearch, withInRangeResultList, currentDimension)
	end if
	
	if returnedCount is greater than or equal to 0 and distanceDiff less than range
	then
	increment foundWithInRangeCount by returnedCount
	if distanceDiff less than or equal to 0
	then
	returnedCount = CALL findWithInRangeSpaTemporal(k-dStH_nodeLeft, coordinatesAboutWhichToSearch, rangeWithinWhichToSearch, withInRangeResultList, currentDimension)
	else
	returnedCount = CALL findWithInRangeSpaTemporal(k-dStH_nodeRight, coordinatesAboutWhichToSearch, rangeWithinWhichToSearch, withInRangeResultList, currentDimension)
	end if
	end if
	
	if returnedCount is -1
	then
	return -1
	end if
	
	increment foundWithInRangeCount by returnedCount
	return foundWithInRangeCount
	
	END

The algorithm II (b) *findWithInRangeSpaTemporal* receives root node *k-dStH_node* of the proposed structure, spatial n-dimensional location *coordinatesAboutWhichToSearch* about which to search for the objects, *rangeWithinWhichToSearch* range within which objects are to be searched, pointer to result list *withInRangeResultList* to hold details of spatio-temporal objects retrieved, *currentDimension* which keeps the track of dimension for current level of the indexing structure **k-dStH** and temporal dimension *timeStampToSearch* which mentions the time with which data should belong to. The algorithm is capable of indexing n-dimensional spatio-temporal data where dimensions of current dataset are initialized using *maxDimensions*. If *k-dStH_node* is *NULL*, then *NO_DATA* is returned back. Otherwise, if spatial coordinate value of current node for *currentDimension* is less than the coordinate value of *coordinatesAboutWhichToSearch* to be searched for same dimension, then control is passed to left sub-tree; otherwise, the control is passed to right sub-tree. It continues on recursive basis until a leaf node is reached or spatial n-dimensional coordinates are matched for every dimension. If the leaf node is reached and still the node with matching *coordinatesAboutWhichToSearch* is not found, then *NO_DATA* is returned, else next, search for temporal attribute by traversing through B-Tree in local hash table at index *epochHashidSearch* is started. Hash key *epochHashidSearch* is generated using another module **GenerateEpochValue (timeStampToSearch)** which receives *timeStampToSearch* and generates epoch value and then hash id fir it. If no entry exists at calculated hash key index *epochHashidSearch* at current node, then return *NO_DATA*, else traverse through B-Tree to add all details in *resultList* and increment the *foundWithInRangeCount* for every record in B-Tree. Before adding to *resultList*, a final filter is applied to match with epoch value of *timeStampToSearch* to filter data out with different timestamps but same hash-id key, as there might be one to many relationships in hash-id key and timestamp value. Also, if any query is time related only without inclusion of nearest neighbor or range search, results can be retrieved using only Master Hash Table which will result in lot of time and resource saving.

VI. EXPERIMENTAL ANALYSIS

The authors have implemented the algorithms using **language** C and GNU Compiler Collection (GCC)

compiler - version 6.4.3 on Operating System Ubuntu-10.04.1-Desktop-amd64. For visualization of spatio-temporal datasets and output of different queries using proposed algorithm, Quantum Geographic Information System (QGIS) Desktop 3.4.7 has been used. The authors have used Google Satellite and Google Map images using XYZ Tiles available in the QGIS browser.

For analysis of our algorithm, the authors have created two synthetic datasets related to trees in Haryana and nearby boundary locations. The coordinates i.e. latitude and longitude of Haryana have been downloaded from online sources and modified using shell scripts in Linux to make it suitable for our research work. First dataset holds the trees those were in Haryana and nearby areas in 2010. Each entry in spatio-temporal dataset represents approx. 5K trees. The second dataset contains the same for the year 2018. The authors have searched for the trees in given range of a location in Haryana at particular time in 2010 and 2018. The retrieved data has been shown pictorially and it can be concluded very easily that there is great fall in the number of trees in queried range during the timespan under study. Also, the performance analysis proves that the proposed indexing structure and algorithm outperform the algorithm based on brute force method of searching. Fig. 01 and Fig. 02 show all the data points (representing approx. 5K trees per data point) of spatio-temporal datasets for the year 2010 and 2018 respectively. Fig. 03 and Fig. 04 show the output of queries to find the trees within range of .002 about *coordinatesAboutWhichToSearch* (Latitude: 30.1, Longitude: 76.8) on Jan 30, 2010 and Jan 30, 2018 respectively. Fig. 05 and Fig. 06 show the output of queries to find the trees within range of .002 about *coordinatesAboutWhichToSearch* (Latitude: 30, Longitude: 76) on Dec 30, 2010 and Dec 30, 2018 respectively. From the images itself, it can be observed clearly that there is drastic fall in number of trees in the area under study.

Table 01 gives the comparision of Algorithms bFSpaTempRangeSrch and k-dStHSpaTempRangeSearch. It shows that both algorithms search for same number of datapoints for every given range, but the time taken by algoritms is noticably different. When location to search about is *coordinatesAboutWhichToSearch* (Latitude: 30.1, Longitude: 76.8) and range is 0.002 radius unit, the algorithm bFSpaTempRangeSrch takes 493 µs and 312 µs for reporting the results related to Jan. 30, 2010 and Dec. 30, 2018 respectively, while algorithm k-dStHSpaTempRangeSearch takes only 09 µs and 04 µs for searching the same. Also, when the location to search about is *coordinatesAboutWhichToSearch* (Latitude: 30, Longitude: 76) and range is 0.002 radius unit, the algorithm bFSpaTempRangeSrch takes 2031 µs and 1273 µs for reporting the results related to Jan. 30, 2010 and Dec. 30, 2018 respectively, while algorithm k-dStHSpaTempRangeSearch takes only 12 µs and 07 µs for searching the same, which is remarkably far less as compared to previous algorithm. It proves that, for given case, time taken by algorithm bFSpaTempRangeSrch is more than 11 times as compared to time taken by algorithm k-StHSpaTempRangeSearch. Similarly, for test cases for diiferent range radius units, our proposed algorithm has been proved far better as compared to brute force method.

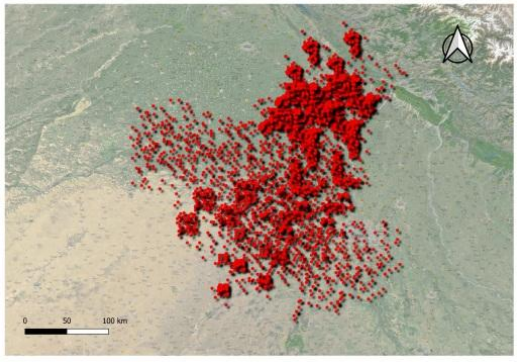

Fig. 01: Trees as per Dataset 01 (Jan. 30, 2010)

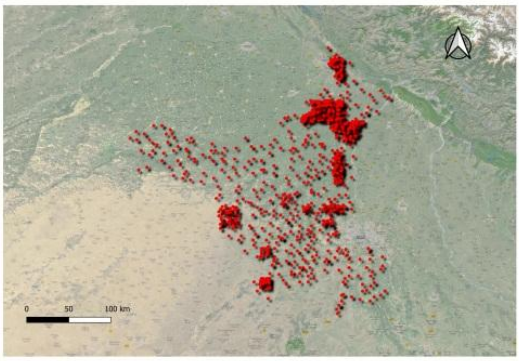

Fig. 02: Trees as per Dataset 02 (Dec. 30, 2018)

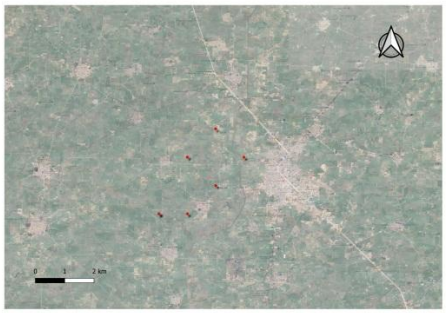

Fig. 03: Trees found within range of .002 about coordinate (Latitude: 30.1, Longitude: 76.8) (Jan. 30, 2010)

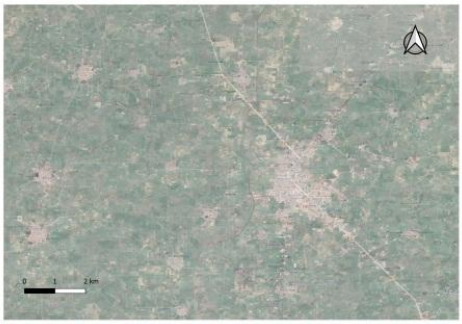

Fig. 04: Trees found within range of .002 about coordinate (Latitude: 30.1, Longitude: 76.8) (Dec. 30, 2018)

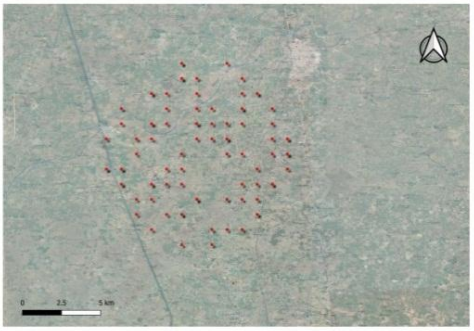

Fig. 05: Trees found within range of .002 about coordinate (Latitude: 30, Longitude: 76) (Jan. 30, 2010)

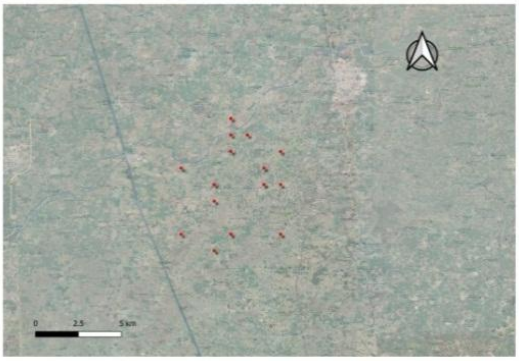

Fig. 06: Trees found within range of .002 about coordinate (Latitude: 30, Longitude: 76) (Dec. 30, 2018)

Table 01: Comparison of Algorithms bFSpaTempRangeSrch and k-dStHSpaTempRangeSearch

Location (Range: radius units)	Count of Neighbors found in given range		Algorithm 01 bFSpaTempRangeSrch (in micro secs.)		Algorithm 02 k-dStHSpaTempRangeSearch (in micro secs.)	
Latitude: 30.1, Longitude: 76.8 (0.002)	(Jan. 30, 2010)	(Dec. 30, 2018)	(Jan. 30, 2010)	(Dec. 30, 2018)	(Jan. 30, 2010)	(Dec. 30, 2018)
	06	00	493	312	9	4
Latitude: 30, Longitude: 76 (0.002)	85	35	2031	1273	12	7

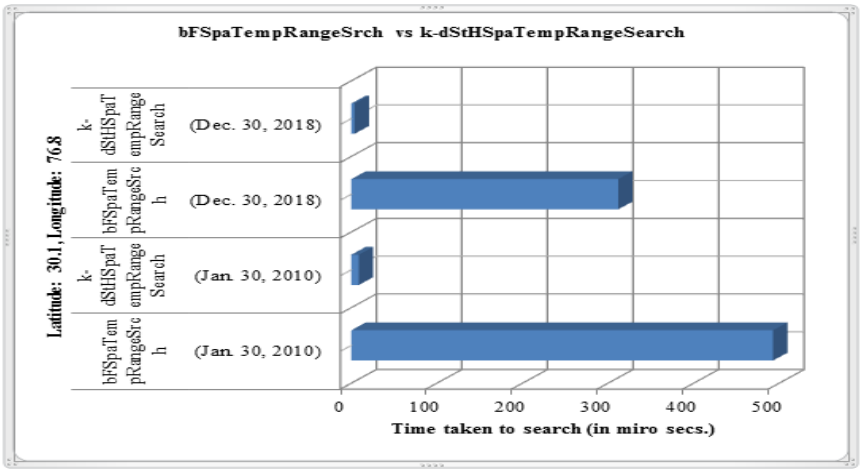

Fig. 08: Performance evaluation of Algorithms bFSpaTempRangeSrch and k-dStHSpaTempRangeSearch (Latitude: 30.1, Longitude: 76.8)

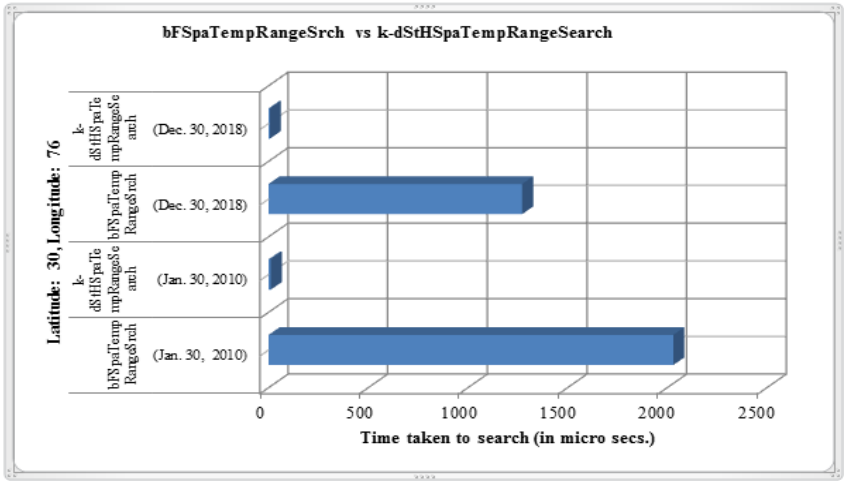

Fig.08: Performance evaluation of Algorithms bFSpaTempRangeSrch and k-dStHSpaTempRangeSearch (Latitude: 30, Longitude: 76)

Fig. 07 and Fig. 08 show the performance evaluation graphically for both query points. It is clear after analysis that algorithm k-dStHSpaTempRangeSearch is far better as compared to algorithm bFSpaTempRangeSrch. The indexing structure and algorithm, the authors proposed, is well efficient to organize spatio-temporal data with duplicate keys. It stores all instances found at same location and even at same time, instead of saving only one latest data record.

VII. Conclusion and Future Scope

With advancement in technologies and availability of more and more spatiotemporal data, it is becoming mandatory in every field to organize and process the received data in an efficient way. In this paper, we proposed a novel indexing structure k-dStH to organize spatio-temporal data. It is modified version of k-dStH indexing structure, which the authors proposed in their earlier work and is composed of k-d tree, B-trees, global and local hash tables. Also, an algorithm for range query on spatio-temporal data is designed and introduced. The authors carried out detailed experimental evaluation to verify the correctness and efficiency of proposed index and algorithm. Experimental analysis show that the proposed structure is appreciable and the range search algorithm k-dStHSpaTempRangeSearch takes remarkably less time to search for objects in given range at particular time as compared to brute force method of range search. In future work, the authors will implement the structure to different areas of research and also extend the structure from security point of view.

REFERENCES

1 Lu and J. Han, "Distance-Associated Join Indices for Spatial Range Search", 1992, pp. 284-292.

2 J. Nievergelt, H. Hinterberger and K. C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure", vol. 9, 1984, pp. 38-71.

3 T. Sellis, N. Roussopoulos and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects", 1987, pp. 3-11.

4 J. T. Robinson, "The K-D-B tree: A Search Structure for Large Multidimensional Dynamic Indexes", 1981, pp. 10-18.

5 H. Samet, "The Design and Analysis of Spatial Data Structures", 1990.

6 Z. B. Feng, L. Qing, L. Mingwei, Y. Zhang, F. Junxiao, Z. Xiao, Y. Li, H. Maosu, Y. Huagui and W. , "An Efficient Graph-Based Spatio-Temporal Indexing Method for Task-Oriented Multi-Modal Scene Data Organization", 2018.

7 C. Zhang, R. Chen, L. Zhu, F. Huang, L. Yunwu and A. Liu, "Hierarchical Information Quadtree: Efficient Spatial Temporal Image Search for Multimedia Stream", 2018.

AUTHORS PROFILE

Sumeet Gill has done Ph. D in Computer Science. He has taught in many reputed technical institutes and has more than 16 years of experience in the field of System Security and Artificial Intelligence. His research papers have been published in different Journals of International/National repute and the proceedings of the National/International Conferences. He has delivered invited talks and chaired sessions in various conferences. Presently, he is working with Maharshi Dayanand University, Rohtak, Haryana as an Associate Professor.

Meenakshi got Master Degree in Computer Applications and M.Tech in Computer Science. Then she completed her M.Phil in computer science. She has worked with Bharti Telesoft (Comviva Technologies), Okhla, Delhi for approx. 3.5 years as software developer and now working with Maharshi Dayanand University, Rohtak, Haryana as an Assistant Professor. She is also pursuing her Ph. D. in Computer Science from the same university.