International Journal of Electrical and Computer Engineering (IJECE)
Vol. 14, No. 3, June 2024, pp. 2937~2944
ISSN: 2088-8708, DOI: 10.11591/ijece.v14i3.pp2937-2944

k-dStHash tree for indexing big spatio-temporal datasets

Meenakshi Hooda, Sumeet Gill
Department of Mathematics, Maharshi Dayanand University, Rohtak, India

Article Info

Article history:

Received Feb 5, 2024
Revised Feb 24, 2024
Accepted Feb 25, 2024

Keywords:

Brute force
Hash table
Indexing structure
Insertion algorithm
k-d tree
Linked list
Spatio-temporal data

ABSTRACT

Today's era is witness of tremendous ever growing spatial, temporal and spatiotemporal data. The huge spatio-temporal data immensely pushes the need for design and development of novel methods tailored for indexing spatio-temporal data. In this research paper, we propose the design of a novel spatio-temporal data indexing method, named as *k-dStHash*. We have proposed the algorithm *k-dStHashInsertion* for inserting spatio-temporal objects and an algorithm *k-dStHashSrchPlaceTime* has been used to search for the objects at given location and time. It is able to handle datasets with duplicate keys which has been ignored in many research works. Though the algorithm *k-dStHashInsertion* takes 1.3-1.5 times longer time to insert data in *k-dStHash* data structure as it needs to find a specific location to organize data efficiently, but when it comes to search for required records it is even more than 90 times faster when analyzed in comparison to brute force method. It is generalized enough to organize any kind of k-dimensional data and time-based data also including object finding, fleet management, clustering, leader identification, nearest neighbor, human/animal tracking, path finding and many more.

Corresponding Author:

Meenakshi Hooda
Department of Mathematics, Maharshi Dayanand University
Rohtak, Haryana, India
Email: meenakshi.maths@mdurohtak.ac.in

1. INTRODUCTION

Different types of spatio-temporal indexing methods to organize big data introduced by researchers in country and abroad can be categorized on the basis of application background and distributed or centralized environment, and the main burning issues which needs attention in the near future, are proposed for addressing everchanging application requirements [1]. In past few years, many surveys, which have been made public, depict the progress in the field of research related to indexes for spatio-temporal records [2], [3]. Surveys highlight that many of the spatial and time based indexing structures aim at centralized indexing systems, i.e. where implementation is main memory based [4], [5]. Distributed computing systems are mostly comprised of stream data processing systems, hybrid processing systems, and batch processing systems [6]. Large spatio-temporal datasets are generated daily at never ever before rates [7], [8], because of fast emerging applications, like location based web search, social networks with geo-tagged content, surveillance systems. Prevailing NoSQL stores deliver restricted support for location based data and fail to provide inherent support for data based on both location and time [9]. The researchers observed that many times, in spatial datasets/databases, multiple entries exist for the same spatial location. Most of the research work either does not include such type of spatial data or remains silent on how to handle multiple records with same location-based key. The duplicate spatial keys are handled using the same method which is used to organize the location-based keys with smaller key values in comparison to current location-based key value, which means that the algorithm treats both equal to (=) and less than (<) relation among the keys in the same way [10]. In another research work, on finding a duplicate spatial key, the address of already existing node with same key is returned back to the calling module [11]. In one other approach, the researchers first remove duplicate spatial records and then rest of the data is considered for further processing [12]. Another method uses context dimension awareness for creating multi-level index. It selects a proper partitioning technique and splits the dataset into multiple balanced divisions [13]. STAQR algorithm takes altitude of a spatial location point and time into consideration to index the data. Following multi-level indexing, this technique indexes all unique codes got from four dimensional data [14]. Multi-scale spatio-temporal grid index (MSTGI) uses Hilbert curves to obtain grid after global geospatial subdivision and then linearizes them [15]. In comparison to generalized linear models based on classical linear, graph regression model for spatial and temporal environmental data results in more general regression relationships and flexibility [16]. In efficient querying and indexing of moving data objects, a new data type based on spatio-temporal predicates results in simple and easier queries [17]. In another technique, a globally coherent model for covariance was used. Also, for better predictions, fixed effects estimation was used, though the predictions were made on local nearest neighbors [18]. Another work proposed three-level spatial index on zone-grid-space for spatio-temporal database based on geographic conditions and, analyzed and tested it over massive land cover data [19]. Researchers divided spatial data by making use of six traditional spatial partitioning techniques and further used machine-learned search within every division to support distance, range, point and even spatial join queries [20]. Spatio-temporal meshing and coding method Hilbert-GeoSOT was proposed for efficient spatio-temporal range queries on big trajectory data [21]. Hadoop cluster can make use of cloud platform's dynamic expansion ability for better expansion of system [22]. The performance measures of Base 64, Base 32, Elias delta and Elias gamma codes on spatial temporal data and different encoding techniques have been illustrated from the time and space complexity point of view [23]. A spatio-temporal data processing system, distributed in nature, ST4ML was proposed to support scalable machine-learning applications [24]. Spatial data infrastructure system supports the use and management of geo-spatial data and resources related to it [25]. A new indexing tree, *k-dLst* to index the spatial data records having duplicate keys was implemented [26]. Researchers proposed a search algorithm based on *k-dSLst* tree for finding nearest neighbor [27]. Here, we have proposed a data structure *k-dStHash* which is capable to index big spatio-temporal datasets with duplicate keys which has been ignored by many researchers. The indexing structure is generalized enough to organize big datasets with k-dimensional duplicate data keys in any field.

2. BRUTE FORCE METHOD

Brute force algorithm explains a style of programming in which no shortcut is used for improving the performance of program. This method believes in absolute computing power and tries for every possibility to find a solution, if exists. Algorithm 1 shows the algorithm *bruteForceInsertion* which is the insertion algorithm to store spatio-temporal data using brute force method.

Algorithm 1. bruteForceInsertion

```
Algorithm prototype          char bruteForceInsertion (struct dataSet
                             *spatioTemporalDataRecord)
Inputs to the algorithm      spatioTemporalDataRecord [type: struct dataSet*]: dataset
                             record to be inserted
Output(s) of the algorithm   SUCCESS [type-char]: Successful insertion or FAILURE [type-
                             char]: Could not insert

Algorithm:
BEGIN
        IF HEAD is NULL
        THEN
                Create a node HEAD
                IF ERROR
                THEN
                        return FAILURE
                END IF
                Update the dataRecord pointer of node HEAD with spatioTemporalDataRecord
                Set HEAD → next ← NULL
                Set TAIL ← HEAD
                return SUCCESS
        ELSE
                Create a node NODE
                IF ERROR
                THEN
                        return FAILURE
                END IF
```

```
            Set NODE → spatioTemporalDataRecord ← spatioTemporalDataRecord
            NODE → next ← NULL
            TAIL → next ← NODE
            TAIL ← NODE
            return SUCCESS
        END IF
END
```

3. METHOD

The algorithm coded in language C and the experimental analysis has been done using "GNU compiler collection (GCC) compiler - version 6.4.3 on Operating System Ubuntu-10.04.1-Desktop-amd64" running on 2.0 GHz Intel (R) Core (TM) 2 Duo CPU T5750 processor with 5 GB installed memory. It is not only organization of spatio-temporal data but also visualization of output of different queries is the demand of the day. For graphical display of spatial locations of crime in spatio-temporal dataset and for the retrieved data points according to user's query, the authors have used "quantum geographic information system (QGIS) desktop 2.12.1". One additional layer with "Google Satellite option of Google Map OpenLayers plugin in QGIS" has been used to show output images realistically. For spatial and spatio-temporal datasets which do not include information in form of latitude and longitude coordinates, we need a geocoder to convert any street address data in form of longitude and latitude coordinates. The researchers have used the online freely available geo-coding sites to do this mapping, wherever required. In any dataset, if it contains a street address only and other details like city name or zip code are missing, the researchers use a default city name or fill the gap with city that is mostly used in that dataset.

3.1. k-dStHash: the proposed indexing structure

The researchers are introducing a novel indexing tree *k-dStHash* which is based on k-d tree, hash table and linked list. This proposed indexing structure is capable of indexing duplicate spatio-temporal key datasets efficiently. The n-dimensional spatial data has been indexed using k-d tree, hash table linked with each k-d tree node indexes spatio-temporal *spatioTemporalDataRecord* using epoch value of timestamp. A linked list is also attached with hash table for storing spatio-temporal records related to particular node for given hash table key. Figure 1 depicts the structure for proposed *k-dStHash* indexing tree.

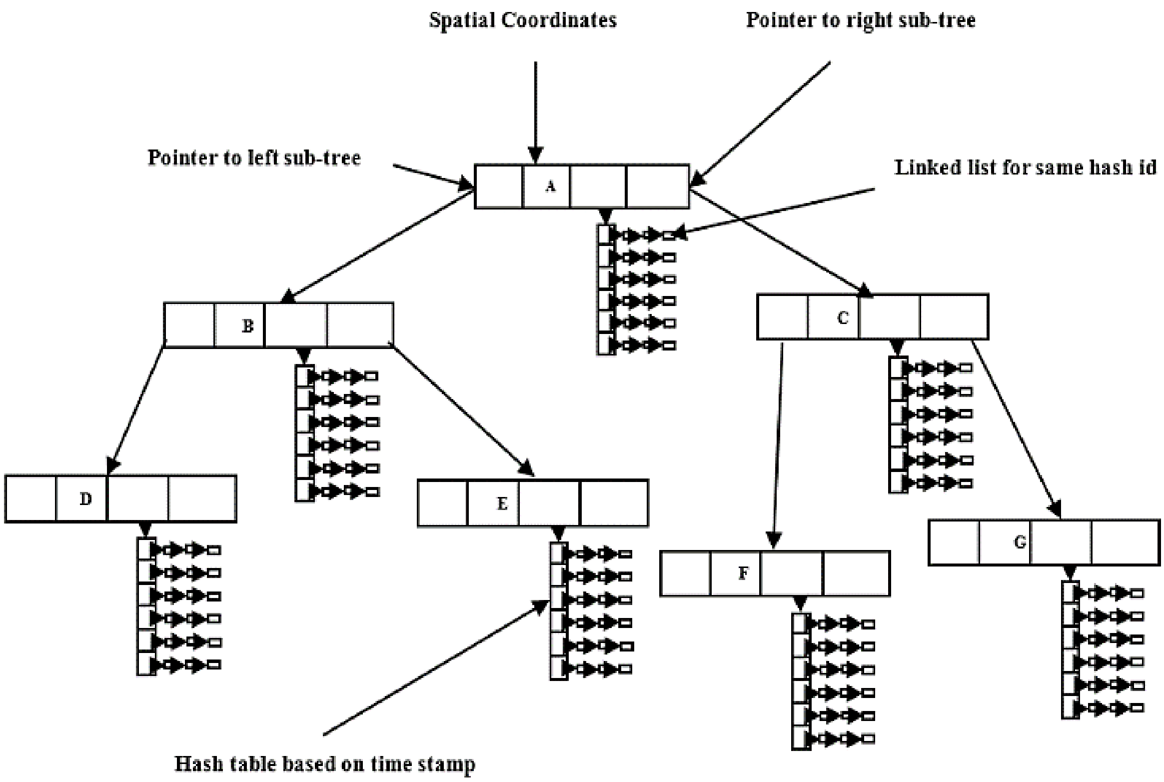


Figure 1. k-dStHash indexing structure

3.2. Creation of k-dStHash tree structure

The authors propose an algorithm to create *k-dStHash* indexing structure and insert spatio-temporal record in the same. A new record is read from spatio-temporal dataset and passed on to *k-dStHashInsertion* module. The algorithm receives pointer to the root node *k-dStHashRoot* which points to the root node of the *k-dStHash* indexing tree/sub-tree, *spatialCoordinates* pointer to spatial n-dimensional coordinates of the record to be inserted, *spatioTemporalDataRecord* pointer to the record read from spatio-temporal dataset and to be inserted, *currentDimension* to keep track of the dimension for the current level of spatio-temporal indexing structure *k-dStHash* and maximum number of dimensions *maxDimensions* required for dataset under consideration. Algorithm 2 *k-dStHashInsertion* gives the steps to create and insert a new node in *k-dStHash* indexing tree. If *k-dStHashRoot* is NULL, then a new tree is created else the existing tree structure is extended. Spatio-temporal data is organized in the *k-dStHash* tree on basis of *spatialCoordinates* and *currentDimension*. If the coordinate key of already existing node for *currentDimension* is less than the coordinate key of *spatialCoordinates* of next record to be inserted for same dimension, then it will traverse towards left sub-tree; else, it will traverse towards right *k-dStHash* sub-tree on recursive basis until the control reaches to a leaf node. The new node is inserted as left son of the leaf node coordinate value of already existing node for *currentDimension* is less than the coordinate key of *spatialCoordinates* to be inserted for same dimension else its inserted as right son of the leaf node. Also, if no node with equal key exists in *k-dStHash* indexing tree, a new *kdSTHashNode* will be created and inserted at proper position and *spatioTemporalDataRecord* will be inserted in hash table at index key generated by using epoch value of temporal attribute associated with it; otherwise, to handle duplicate spatial keys *spatioTemporalDataRecord* will be inserted in the list of matching *kdSTHashNode* i.e. node with equal n-dimensional keys at index key generated by using epoch value of temporal attribute associated with it such that the list remains in sorted order of epoch temporal values.

Algorithm 2. Proposed k-dStHashInsertion

```
Algorithm                k-dStHashInsertion
Inputs to Algorithm      - k-dStHashRoot [type - struct kdnode**]: Root node of k-dStHash
                           Indexing Tree
                         - spatialCoordinates [type - const double*]: N-dimensional
                           coordinates of current node
                         - spatioTemporalDataRecord [type - struct dataset*]: Data record
                           read from dataset under consideration
                         - currentDimension [type - int]: Dimension of current node
                         - maxDimensions [type - int]: Maximum number of dimensions of
                           spatio-temporal data
Output from Algorithm    SUCCESS [type-char]: Successful insertion or FAILURE [type-char]:
                         Could not insert

BEGIN
  IF k-dStHashRoot is NULL
  THEN
      Allocate memory for new record and assign the pointer to k-dStHashRoot
      Allocate memory for Hash Table for k-dStHashRoot
      Initialize k-dStHashRoot→timeHash→timeChain table with NULL
      Generate timeHashId by using timeHashFunction based on epoch value of timestamp in
         current spatioTemporalDataRecord
      Insert spatioTemporalDataRecord at generated timeHashId in k-dStHashRoot → timeHash
         → timeChain table
  SET k-dStHashRoot→left ← k-dStHashRoot→right ← NULL
      IF ERROR
      THEN
          return FAILURE
      ELSE
          return SUCCESS
      END IF
  END IF
  SET new_currentDimension ← (node → currentDimension + 1) mod maxDimensions
  IF (spatialCoordinates[node → currentDimension] < node → spatialCoordinates[node →
        currentDimension])
  THEN
        CALL k-dStHashInsertion with left pointer of current node and updated parameters
  END IF
     IF spatialCoordinates have duplicate keys
     THEN
         Generate timeHashId by using timeHashFunction based on epoch value of timestamp in
            current
     IF no record on generated hash id
```

```
     THEN
        Insert current record at generated hash id
     ELSE
          Insert current record in a linked chain at generated hash id in ascending order of
             epoch time
     END IF
     IF ERROR then
             return FAILURE
     ELSE
             return SUCCESS
     END IF
  END IF
CALL k-dStHashInsertion with right pointer of current node and updated parameters
END
```

4. RESULTS AND DISCUSSION

The researchers have analyzed the algorithms on different synthetic spatio-temporal datasets. Crime dataset contains the location and time of different types of crimes happened all over the earth in January, 2019. The format of crime dataset (source: *https://catalog.data.gov*) is as given in Table 1. For the dataset, the researchers have queried the data for both types of queries i.e. spatial and spatio-temporal. In spatial queries, the researches queried about the crimes at particular location i.e. at given latitude and longitude values, while in spatio-temporal queries information is retrieved about crimes at particular latitude, longitude and time as well.

Table 1. Format of crime dataset

Crime Id	Country	Area	Longitude	Latitude	Crime details	Date of crime	Time of crime
C74098	Afghanistan	Kabul	34.516667	69.183334	BURGLARY FROM VEHICLE	1-Jan-19	12:00
C74099	Afghanistan	Kandahar	31.61	65.699997	BUNCO, GRAND THEFT	3-Jan-19	11:42
C74100	Afghanistan	Mazar-e Sharif	36.706944	67.112221	ROBBERY	3-Jan-19	21:00

First, the researchers have analyzed the performance of both insertion algorithms *i.e. bruteForceInsertion* and *k-dStHashInsertion*. As, algorithm *bruteForceInsertion* simply inserts the record in linked list without any comparison, it takes less time in insertion as compared to *k-dStHashInsertion*, in which lot of comparisons and calculations are required to organize spatio-temporal data efficiently. But, as we need to insert the data only once and retrieve it frequently, the time taken to insert the spatio-temporal data in *k-dStHash* tree structure can be compromised against its fast retrieval time. Table 2 shows the performance comparison with respect to time taken to insert 23,602 spatio-temporal data records of Crime Dataset. The researchers executed both insertion algorithms 250 times in iteration using a script and picked 05 random iterations for analysis. It shows a comparative analysis if time taken to insert spatio-temporal data records of Dataset using algorithms *bruteForceInsertion* and *k-dStHashInsertion* for iteration number 01, 99, 187, 203, 250. For every iteration, the analysis shows that time taken by algorithm *k-dStHashInsertion* is more when compared with that of the other algorithm bruteForceInsertion. Figure 2 shows the performance analysis of algorithms bruteForceInsertion and k-dStHashInsertion graphically as per Table 2. Figure 3 shows data records related to given query location along with the information retrieved with respect to both spatial and spatio-temporal queries.

Table 2. Performance analysis of insertion algorithms

Crime Dataset-Number of records: 23602		
Algorithm	bruteForceInsertion	k-dStHashInsertion
Randomly picked iterations (out of total 250 iterations)	Time taken (ms)	
Iteration-01 (SET-A)	77,084	111,727
Iteration-99 (SET-B)	76,790	104,964
Iteration-187 (SET-C)	77,083	105,246
Iteration-203 (SET-D)	77,542	106,310
Iteration-250 (SET-E)	77,058	105,527

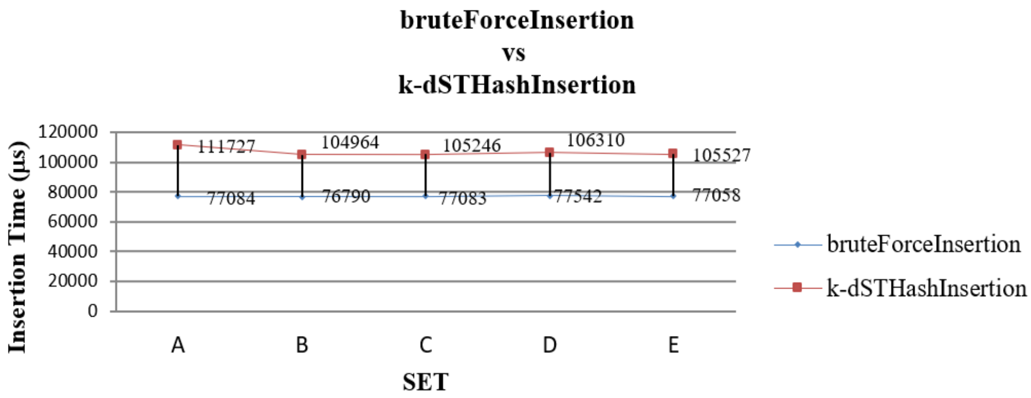
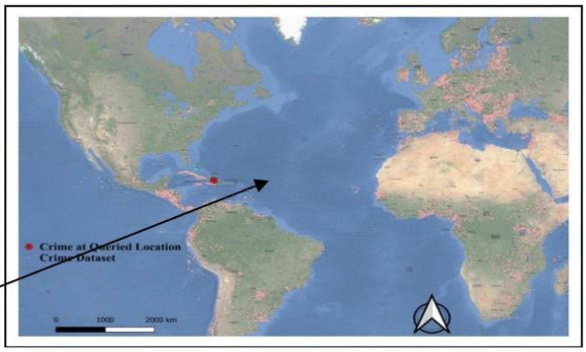


Figure 2. Performance analysis of algorithms *bruteForceInsertion* and *k-dStHashInsertion*



Crime Id	Longitude	Latitude	Crime Details	Date of Crime	Time of Crime	Query Type
C75093	19.6908333	-72.0161133	BURGLARY FROM VEHICLE	19-Jan-19	9:30	Spatial and Spatio-temporal
C94090	19.6908333	-72.0161133	BUNCO, GRAND THEFT	19-Jan-19	9:30	Spatial and Spatio-temporal
C94100	19.6908333	-72.0161133	ROBBERY	19-Jan-19	9:30	Spatial and Spatio-temporal
C97198	19.6908333	-72.0161133	BURGLARY FROM VEHICLE	19-Jan-19	9:30	Spatial and Spatio-temporal
C98102	19.6908333	-72.0161133	VIOLATION OF RESTRAINING ORDER	19-Jan-19	9:30	Spatial and Spatio-temporal
C99006	19.6908333	-72.0161133	THREATENING PHONE	19-Jan-19	9:30	Spatial and Spatio-temporal
C99910	19.6908333	-72.0161133	THEFT FROM MOTOR	22-Feb-19	0:50	Spatial
C10814	19.6908333	-72.0161133	THEFT OF IDENTITY	9-Jan-19	1:32	Spatial
C10171	19.6908333	-72.0161133	DISCHARGE FIREARMS/SHOTS FIRED	19-Jan-19	3:30	Spatial

Figure 3. Crimes at location (Latitude: -72.0161133, Longitude: 19.6908333)

The performance analysis of algorithms *bruteForceSearch* and *k-dStHashSrchPlaceTime* in terms of search time (in μs) is illustrated in Table 3. Algorithm *k-dStHashSrchPlaceTime* takes extremely lesser time to search for any object at any particular time. For example, for a query to search for object at location with latitude 72.0161133 and longitude 19.6908333 on 1/19/2019 at 9:30 am, when *bruteForceSearch* algorithm takes 516.8 µs, *k-dStHashSrchPlaceTime* algorithm takes only 5.6 µs which is approx. 92 times faster. It depends not only on number of objects found but also the location of record at which spatio-temporal record is saved in the indexing structure. Similarly, the table depicts comparison among different SET(s) A-E, and, in every case *k-dStHashSrchPlaceTime* algorithm outperforms *bruteForceSearch* algorithm. Figure 4 shows the comparison of search time taken by both algorithms graphically as per Table 3.

Table 3. Time performance analysis of algorithms *bruteForceSearch* and *k-dStHashSrchPlaceTime*

Search time comparison (in µs) crime dataset (Number of records: 23602) randomly picked iterations (out of total 250 iterations)					
Latitude	72.0161133	50.8644447	50.8644447	70.987952	76.853736
Longitude	19.6908333	27.2038889	27.2038889	20.273855	23.216667
Time	1/19/2019 9:30 AM	1/20/2019 12:34 PM	1/29/2019 12:34 PM	1/25/2019 8:53 AM	1/28/2019 5:50 AM
SET	A	B	C	D	E
bruteForceSearch (ms)	516.8	173.4	169.6	181	164.8
k-dStHashSrchPlaceTime (ms)	5.6	2.2	48.4	4	2.4

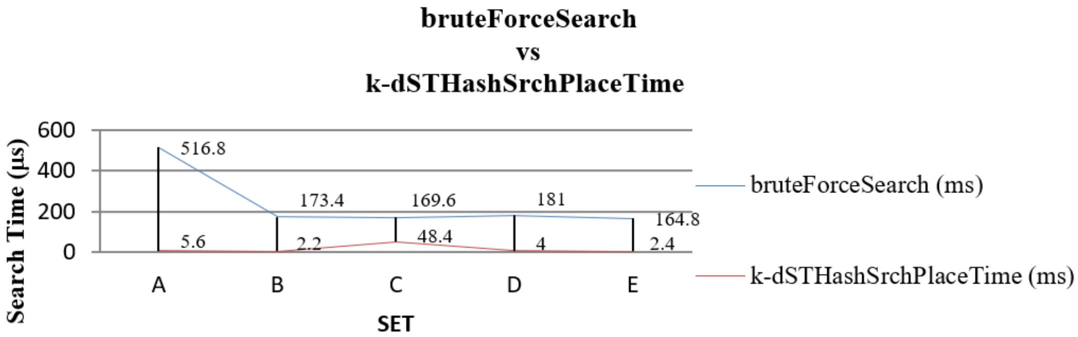


Figure 4. Performance analysis of *bruteForceSearch* and *k-dStHashSrchPlaceTime* (crime dataset)

5. CONCLUSION

The research work introduced a new structure *k-dStHash* tree to index location and time-based data. Though the insertion algorithm takes more time to organize the records according to both location and time, but, when it comes to retrieval of required data, which is even more frequent, it outperforms the other indexing method based on brute search. As illustrated in experimental analysis, when insertion time of k-*dStHashInsertion* algorithm is 1.45, 1.37, 1.37, 1.37, 1.36 times more than time taken by *bruteForceInsertion* algorithm for SET(s) A-E respectively, while retrieving is 92.28, 78.82, 3.50, 45.25 and 68.67 times faster for same SET(s) A-E respectively. The experimental analysis demonstrates that the introduced indexing structure can proficiently organize, store and maintain spatio-temporal records and retrieve the required records speedly. The structure can be implemented for any research area with spatio-temporal data and even for datasets with k-dimensional duplicate keys. Further, the work can be enhanced to retrieve spatio-temporal objects within a given range and given time window. The time window can be static or sliding to suit real time analysis.

REFERENCES

[1] R. Tian, H. Zhai, W. Zhang, F. Wang, and Y. Guan, “A survey of spatio-temporal big data indexing methods in distributed environment,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 4132–4155, 2022, doi: 10.1109/JSTARS.2022.3175657.

[2] A. R. Mahmood, S. Punni, and W. G. Aref, “Spatio-temporal access methods: a survey (2010 - 2017),” *GeoInformatica*, vol. 23, no. 1, pp. 1–36, Oct. 2018, doi: 10.1007/s10707-018-0329-2.

[3] K. Jitkajornwanich, N. Pant, M. Fouladgar, and R. Elmasri, “A survey on spatial, temporal, and spatio-temporal database research and an original example of relevant applications using SQL ecosystem and deep learning,” *Journal of Information and Telecommunication*, vol. 4, no. 4, pp. 524–559, Sep. 2020, doi: 10.1080/24751839.2020.1774153.

[4] P. Li, H. Lu, Q. Zheng, L. Yang, and G. Pan, “LISA: a learned index structure for spatial data,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, May 2020, pp. 2119–2133, doi: 10.1145/3318464.3389703.

[5] E. Carneiro, A. V. de Carvalho, and M. A. Oliveira, “I2B+tree: interval B+ tree variant towards fast indexing of time-dependent data,” *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, Seville, Spain, 2020, pp. 1-7, doi: 10.23919/cisti49556.2020.9140897.

[6] W. P. Guo, Y. H. Zhao, G. R. Wang, and L. G. Wei, “Efficient fault-tolerant processing technology for Flink iterative computing,” *Jisuanji Xuebao/Chinese Journal of Computers*, vol. 43, no. 11, pp. 2101–2118, 2020, doi: 10.11897/SP.J.1016.2020.02101.

[7] G. M. Santipantakis *et al.*, “SPARTAN: semantic integration of big spatio-temporal data from streaming and archival sources,” *Future Generation Computer Systems*, vol. 110, pp. 540–555, Sep. 2020, doi: 10.1016/j.future.2018.07.007.

[8] C. Yang, K. Clarke, S. Shekhar, and C. V. Tao, “Big spatiotemporal data analytics: a research and innovation frontier,” *International Journal of Geographical Information Science*, vol. 34, no. 6, pp. 1075–1088, Jun. 2020, doi: 10.1080/13658816.2019.1698743.

[9] N. Koutroumanis and C. Doulkeridis, “Scalable spatio-temporal indexing and querying over a document-oriented NoSQL store,” *Advances in Database Technology - EDBT*, pp. 611–622, 2021, doi: 10.5441/002/edbt.2021.71.

[10] R. A. Brown, “Building a balanced k-d tree in O(kn log n) time,” *arXiv:1410.5420*, Oct. 2014.

[11] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975, doi: 10.1145/361002.361007.

[12] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, Sep. 1977, doi: 10.1145/355744.355745.

[13] R. Tian, W. Zhang, F. Wang, and J. Xiong, “A context-aware method for indexing large-scale spatiotemporal data,” in *2022 IEEE International Conference on Big Data (Big Data)*, Dec. 2022, pp. 6057–6065, doi: 10.1109/BigData55660.2022.10020916.

[14] P. Madhavi and K. P. Supreethi, “STAQR tree indexing for spatial temporal data with altitude,” *GIS Science Journal*, Nov. 2022, doi: 10.21203/rs.3.rs-2238587/v1.

[15] H. Liu *et al.*, “MSTGI: a multi-scale spatio-temporal grid index model for remote-sensing big data retrieval,” *Remote Sensing Letters*, vol. 15, no. 1, pp. 44–54, Dec. 2023, doi: 10.1080/2150704x.2023.2293474.

[16] R. Tayewo, F. Septier, I. Nevat, and G. W. Peters, “Graph regression model for spatial and temporal environmental data—case of carbon dioxide emissions in the United States,” *Entropy*, vol. 25, no. 9, Aug. 2023, doi: 10.3390/e25091272.

[17] S. Chaturvedi and T. Nagpal, “Efficient querying and indexing of moving data objects,” *2022 International Conference on Futuristic Technologies (INCOFT)*, Belgaum, India, 2022, pp. 1-6, doi: 10.1109/incoft55651.2022.10094348.
[18] J. M. Ver Hoef, M. Dumelle, M. Higham, E. E. Peterson, and D. J. Isaak, “Indexing and partitioning the spatial linear model for large data sets,” *PLOS ONE*, vol. 18, no. 11, Nov. 2023, doi: 10.1371/journal.pone.0291906.
[19] Y. Gao, H. Duo, J. Che, S. Zhao, and B. Zhao, “Research on efficient indexing of large-scale geospatial data based on multi-level geographic grid,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 73–80, Dec. 2023, doi: 10.5194/isprs-annals-x-1-w1-2023-73-2023.
[20] V. Pandey *et al.*, “Enhancing in-memory spatial indexing with learned search,” *arXiv:2309.06354*, Sep. 2023.
[21] H. Liu, J. Yan, J. Wang, B. Chen, M. Chen, and X. Huang, “HGST: a Hilbert-GeoSOT spatio-temporal meshing and coding method for efficient spatio-temporal range query on massive trajectory data,” *ISPRS International Journal of Geo-Information*, vol. 12, no. 3, Mar. 2023, doi: 10.3390/ijgi12030113.
[22] J. Xu, B. Chen, and L. Sun, “Big data storage index mechanism based on spatiotemporal information cloud platform,” *Security and Communication Networks*, vol. 2022, pp. 1–8, Aug. 2022, doi: 10.1155/2022/6774821.
[23] P. Madhavi and K. P. Supreethi, “A metaphorical analysis of different encoding techniques for spatial temporal data,” *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 11s, pp. 302–308, 2023.
[24] K. Liu, P. Tong, M. Li, Y. Wu, and J. Huang, “ST4ML: machine learning oriented spatio-temporal data processing at scale,” *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–28, May 2023, doi: 10.1145/3588941.
[25] K. Shahi, “Volunteered geographic information (VGI) in spatial data infrastructure (SDI) continuum,” *EAI Endorsed Transactions on Internet of Things*, vol. 9, no. 1, May 2023, doi: 10.4108/eetiot.v9i1.2979.
[26] Meenakshi and S. Gill, “k-dLst tree: k-d Tree with linked list to handle duplicate keys,” in *Emerging Trends in Expert Applications and Security*, Springer Singapore, 2019, pp. 167–175.
[27] M. Hooda and S. Gill, “Nearest neighbour search in k-dSLst tree,” *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 4, pp. 160–166, Jul. 2020, doi: 10.25046/aj050419.

BIOGRAPHIES OF AUTHORS

Meenakshi Hooda did master degree in computer applications and M.Tech in computer science. Then she completed her M.Phil. and Ph.D. 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 for last 10 years. Her research areas are indexing, sptio-temporal indexing, image processing, fuzzy logic, steganography and cryptography. She can be contacted at email: meenakshi.maths@mdurohtak.ac.in.

Sumeet Gill has done Ph.D in computer science. He has taught in many reputed technical institutes and has more than 25 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 professor. He can be contacted at email: drSumeetGill@mdurohtak.ac.in.