ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

A Unified Framework for Answering k Closest Pairs Queries and Variants using Grouping of Objects

Mintu Thomas, Neena Joseph

Student, Department of CSE, Mangalam College of Engineering, Ettumanoor, Kottayam, Kerala, India

Assistant Professor, Department of CSE, Mangalam College of Engineering, Ettumanoor, Kottayam, Kerala, India

ABSTRACT: We propose a unified framework for answering *k* closest pairs queries, *k* furthest pairs queries and their variants by using the concept of grouping of objects. The existing unified framework [1] used for the same purpose efficiently answer a broad class of top-*k* queries including top-*k* pairs queries on multi-valued objects and exclusive top-*k* pairs queries. Where, when giving a scoring function that computes the score of a pair of objects, a top-*k*pairs query returns *k* pairs with the smallest scores. But, its performance is poor when the dataset is very large. This is because it requires a large amount of time to pairing and scoring of large amounts of data objects and then to returning the output. So, in order to overcome that problem and thus to improving the efficiency, we introduce a new concept called grouping of data objects. Where, we try to discover a group of object pairs from the heap of all object pairs. The size of this group is depends on the value of *k* and the score of object pairs. For that purpose, we find an efficient equation also. This new concept can be applied in the case of non-chromatic and chromatic pairs queries. It can be used with the existing unified framework. In our proposed solution, all operations are similar to the existing one. The only difference is that here, all operations are applied to a particular group of objects only. So, the expected performance of our framework is much higher than the previous version and it requires less computation time. It can even be used with very large datasets of real world. In addition to this, like its previous version, it does not require any pre-built indexes, uses limited main memory and is easy to implement.

KEYWORDS: Closest Pairs Queries, Furthest Pairs Queries, Chromatic and Non-Chromatic Pairs Queries, Top-*k* Queries, Multi-Valued Objects, Exclusive Pairs Queries, Grouping of Objects.

I. INTRODUCTION

Answering *k* closest pairs queries and their variants have very significant role in real world applications. For example, at the time of course and seat allotment, posting of staffs to corresponding vacancies, exam center allocation, pair-trading, computational geometry applications, database applications and in the case of many other applications, there is a need of an efficient solution to the above problem. In order to solve that problem, we can see several methods in the area of knowledge and data engineering. The unified framework proposed in [1] and [2] are two efficient methods used for solving that problem. But, the performance of that framework is poor in the case of large datasets. So, in order to overcome that problem, we make an additional step to that existing framework. That is, we propose a unified framework for answering *k* closest pairs queries; *k* furthest pairs queries and their variants by using the concept of grouping of data objects. This technique is similar to the one that proposed in [1] except that in which all operations are performed in a particular group of objects only. So, the total time required for performing the same is much reduced in our framework. Thus, it becomes an efficient framework even in the case of very large dataset.

Here, by *k* closest pairs we mean the most similar pairs. That is, *k* closest pairs query returns *k* pairs with the smallest distances if we rank the pairs based on distance functions. Similarly, by *k* furthest pairs we mean most dissimilar pairs. Here, we consider its chromatic and non-chromatic variants also. That is, if each object in a database has assigned a color, then a chromatic query consider color of objects and a non-chromatic query does not consider color of objects. Chromatic queries are of two types: homochromatic and heterochromatic top-*k* pairs queries. A homochromatic top-*k*pairs query returns top-*k* pairs that contain two objects having the same color. But, a

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

heterochromatic top-*k*pairs query returns top-*k* pairs that contain two objects having different colors. Our framework also considers the case of multi-valued object pairs queries and exclusive object pairs queries. An object having multiple instances is called multi-valued object [1]. Let *S* be the set of top-*k* pairs, then we say that an object o_u satisfies the exclusiveness constraint if $o_u$appears at most once in the set *S*.

Top-*k* queries retrieve the top-*k* objects by using a user defined scoring function. Paper [7] is an example for this. Our proposed framework, as its previous version supports a generic ranking function that is based on both local and global scoring functions. A local scoring function computes the score of a pair of objects on a single attribute. But, a global scoring function computes the final score of a pair by combining all of its local scores.

II. Related work

M. A. Cheema, in [1] proposed a unified framework for answering *k* closest pairs queries and variants. This is the just previous version of our proposed framework. This paper proposes different algorithms for creating and maintaining the sources in the case of both single valued and multi-valued objects. Both of these algorithms are the variations of one of the top-*k* algorithm called Threshold algorithm (TA). TA supports both sorted and random accesses. This framework supports a generic ranking function for finding the rank of object pairs. This ranking function is based on both local and global scoring functions. That is, here, the main concept is that let *d* be the number of local scoring functions found in our query, then each source $S_i$incrementally returns a pair with the best score according to the *i*th local scoring function. Then the existing top-*k* algorithm can be used to return the top-*k* pairs by combining these ranked inputs. This paper also proposes methods for handling multi-valued object pairs queries. Where, we define top-*k* pairs based on $\emptyset$-quantile scores. This paper also discusses about the solution for finding exclusive top-*k* pairs queries. All of these are efficient solutions but their performance is degraded as the size of the dataset increases.

The unified framework proposed in [2] is used to compute top-k pairs in multi-dimensional space. This is the just previous version of the framework proposed in [1]. This paper proposes efficient internal and external memory algorithms for returning top-*k* pairs. The different query processing algorithms discussed in this paper are Score-based Top-*k* Pairs Queries, Skyline Pairs Queries and Rank-based Top-*k* Pairs Queries. The Score-based Top-*k* Pairs Queries uses threshold algorithm (TA) to combine the scores of a pair from different sources and return the top-*k* pairs. A skyline pairs query returns every pair that is not dominated by any other pair. And, the Rank-based Top-*k* Pairs Queries is used when it is difficult to define a global scoring function on the attributes that are incompatible. The expected performance of those algorithms is optimal when two or less attributes are involved. But, the unified framework proposed in [1] overcomes this limitation.

The paper [3] proposes Incremental Distance Join Algorithms for Spatial Databases. This is the first paper that discusses about closest pair problem. Here, two new spatial join operations, distance join and distance semi join, are introduced. In these operations, the join output is ordered by the distance between the spatial attribute values of the joined tuples. All of these operations are performed in a pipelined fashion by using the incremental algorithms andin the context of the R-tree. R-trees are one of the spatial data structures. Here, the incremental distance join algorithm works as simultaneously applying an incremental nearest neighbor algorithm to the two spatial data structures.These spatial data structures are corresponding to the spatial attributes of the joined relations. The distance semi-join is a special case of distance join in which the resulting pairs is limited to a range.

The second paper that discusses about closest pair problem is [4] in which spatial queries that combine join and nearest neighbor query is examined. It is called *k* closest pairs query (K-CPQ). It finds *k*-closest pairs between two spatial datasets. Here, the case of 1-CPQ is considered as a special case and each dataset is stored in a structure belonging to the R-tree family. In order to solve the *k* closest pairs problem, an extra structure called *k*- heap is used. Which is organized as a max-heap and it holds pairs of objects according to their distance. So, the pair of points with the largest distance occupies on the top of the heap.

The paper [5] discusses about on spatial range closest pair queries. That is, it addresses the problem of closest pairs within the same dataset or inside a given range. So, the problem of this paper is also called as Self Range Closest Pair problem (SRCP problem). Its entire previous works discusses only about closest pairs of objects among two datasets. This paper proposes two versions of the SRCP-tree and a new index structure to solve that problem. The SRCP-tree

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

version one is an R-tree augmented with “local” closest pair information. But, the SRCP-tree version two is an R-tree augmented with “local-parent” closest pair information along with each index entry.

III. PROPOSED METHOD

A. *Preliminaries:*

Atop-k query retrieves top-k objects based on a user defined scoring function. This scoring function is based on both local and global scoring functions. The local scoring function is loose monotonic and the global scoring function is monotonic. A loose monotonic function is more general than the monotonic function. Now, we can see some useful definitions.

Top-k pairs query: Let Obe the set of objects, a non-chromatic top-k pairs query returns a set of pairs $P \subseteq O$ x O that contains k pairs such that for any pair $(a, b) \in P$ and any pair $(a', b') \notin P$, SCORE $(a, b) \leq$SCORE (a', b') [1].

A function f is called a *monotonic* function if it satisfies $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ where,$x_i \leq y_i$ for every $1 \leq i \leq n$. Similarly, a function $s(..,..)$ is called a *loose monotonic* function if for every value x_i both of the following are true: i) for a fixed x_i and every $x_j > x_i$, $s(x_i, x_j)$ either monotonically increases or monotonically decreases as x_j increases, and ii) for a fixed x_i and every $x_k < x_i$, $s(x_i, x_k)$ either monotonically increases or monotonically decreases as x_k decreases [1].

In addition to the above, a loose monotonic functionis called right increasing (resp. decreasing) function iffor every $x_j > x_i$ for the fixed x_i,$s(x_i, x_j)$ monotonicallyincreases (resp. decreases) as x_j increases.A loosemonotonic function is called left increasing (resp. decreasing)function if for every $x_k < x_i$for the fixed x_i, $s(x_i, x_k)$monotonically increases (resp. decreases) as k decreases [1].

B. *Unified framework for answering k closest pairs queries and variants using grouping of objects:*

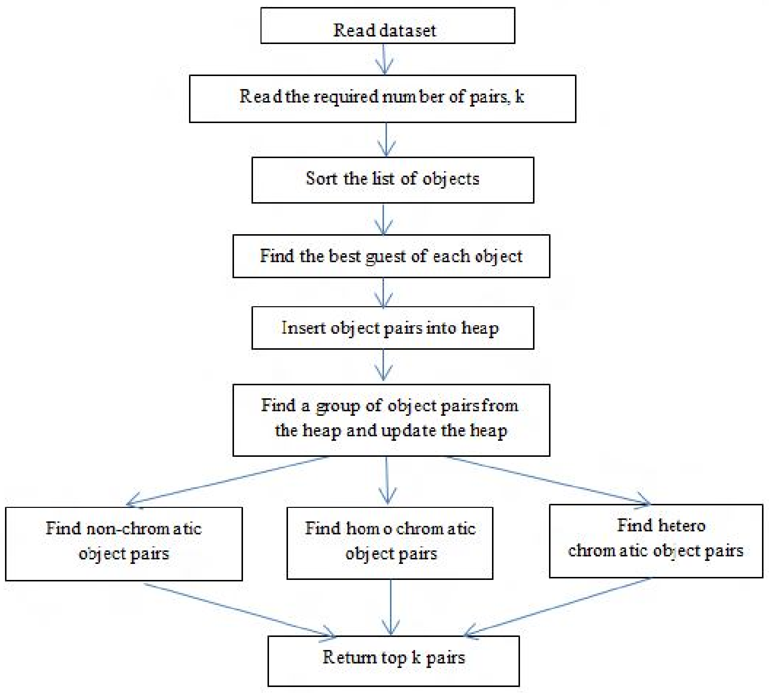
Read dataset
Read the required number of pairs, k
Sort the list of objects
Find the best guest of each object
Insert object pairs into heap
Find a group of object pairs from the heap and update the heap
Find non-chromatic object pairs
Find homo chromatic object pairs
Find hetero chromatic object pairs
Return top k pairs

Fig. 1: System architecture

Our proposed framework is similar to the one that defined in [1] except that it works by using the concept of grouping of data objects. In the existing framework, for top-*k*pairs computation we first sort all objects in ascending order of their attribute values. Then, we find different combinations of these object pairs and calculated its local scores on the *i*th attribute by using a local scoring function. Let *d* be the number of local scoring functions, then each source S_i incrementally returns its best pair according to the *i*th local scoring function. Finally a top-*k* algorithm is used to retrieve top-*k* pairs by combining all of the *d* local scores.

We are using a new concept because, in the existing framework, the top-*k*pairs computation requires a large amount of time when the dataset is very large. This is because if the dataset is very large, then the pairing of objects and scoring of all combinations of pairs require large computations. For example, if there are 100 data objects in a dataset, then the pairing of that data objects leads to the generation of $100c_2$ combinations of pairs. The computational cost for processing all of these pairs is very high. In addition to this, their computational cost also increases as the *k* value increases. So, in order to overcome that problem, we find a group of object pairs from the set of all combinations of object pairs by using an efficient equation. Then apply all of the pre-mentioned operations in that particular group only. We also find an algorithm for creating and maintaining the sources. This will reduce the total computations and thus the time required for completing the computations. The accuracy is also maintained. Our solution is better in the case of non-chromatic and chromatic object pairs queries. Fig.1 shows the architecture of our system.

From this architecture, we can see that the heap size is reduced to the group size after finding the group. Now, we can find the top *k* pairs by processing those single group elements only. The more detailed explanation of the algorithm is given in the next section.

C. *Creating and Maintaining the Sources*

In our solution, we require that each source should return its top-*k* pairs in a sorted order. So, we need an efficient technique to creating and maintaining the sources. Here, we present an internal memory optimal algorithm for that purpose. In this algorithm, we first sort all objects in ascending order of their attribute values such that $o_1 \leq o_2 \leq \ldots \leq o_M$. We assume that the local scoring function which satisfies $s(o_u, o_v) = s(o_v, o_u)$ and we consider only the pairs (o_u, o_v) where u <v. For any pair (o_u, o_v), we called o_u as host and o_v as guest of o_u. If we assume that the queries are non-chromatic and the scoring function is absolute difference, then the object o_v is called the best guest of o_u if for every other guest $o_{v'}$of o_u, $s(o_u, o_v) \leq s(o_u, o_{v'})$. Where, $s(o_u, o_v)$ means the score of the pair (o_u, o_v). In our proposed solution, we find scores of object pairs based on a distance function called absolute difference.

Here, for creating and maintaining the sources, we use an improved algorithm called Algorithm New. This algorithm is shown below. Here, first we sort all objects in ascending order of their attribute values. Then we find the best guest of each object o_u in the sorted list and denoted it by a pair (o_u, o_v) where,o_v is the best guest of o_u. Then, we insert all such pairs into heap1 with its score. The size of this heap will increase as the size of the dataset increases. So, in order to reduce its size and thus to reduce the computations, we find a group of object pairs from this heap. That is, if the value of *k* is 1, then the algorithm simply returns the top pair of heap 1 as result. But, if the value of *k* is greater than 1, then the algorithm retrieves the top pair of heap 1 and find its next best guest upto*k*-1 times. The resulting pair $(o_u, o_{v'})$ is inserted into another heap called heap 2 with its score, $s(o_u, o_{v'})$. Then scan heap 1 upto finding an object pair (o_u, o_v) such that $s(o_u, o_v) \geq s(o_u, o_{v'})$. If such a pair exists, then we remove that pair (pairs) and its subtree (subtrees) from heap1. Now, we get a reduced heap. This reduced heap will form the new group of object pairs. Then we continue all of the pre mentioned operations in that reduced heap until the heap become empty. As the algorithm proceeds, the heap size (group size) may be reduced. This is because of the formation of new heap 2 pairs whose score is less than the existing ones. Finally, we retrieve the top pair of heap 2 and return all of the path pairs between its objects as the result of the query or top *k* pairs.

The same technique can be applied in the case of all of its variants such as homochromatic and heterochromatic object pairs queries. In the case of chromatic queries, the best guest of an object o_u is the right adjacent object of o_u that meet the color requirement. That is, in the case of homochromatic queries, the best guest of o_u is the right adjacent object of o_u with the same color as that of o_u. Similarly, in the case of heterochromatic queries, the best guest of o_u is

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

the right adjacent object of $o_u$with a different color. After finding the adjacent objects and best guest of each chromatic object, we can answer its top *k* pairs query by following the same procedure as that of chromatic object pairs. Here also the use of the reduced heap or the group of object pairs improves the efficiency of our algorithm.

Algorithm New: Creating and Maintaining a Source

```
        InitializeSource()
1.  {
2.  sort all objects in ascending order of their values;
3.  for each object $o_u$ do
4.  $o_v$ ← the best guest of $o_u$;
5.      insert the pair ($o_u$, $o_v$)  into heap 1 with score
    $s(o_u, o_v)$;
6.   }
7.  read the value of $k$;
8.  if $k$ = 1, then
9.      return the top pair of heap 1 as result;
10. else
11.     while (heap 1 is non-empty)
12.     {
13.         for each pair ($o_u$, $o_v$) in heap 1 do
14.             call getNextBestPair ();
15.             scan heap 1 until to obtain a pair ($o_u$, $o_v$)
such that $s(o_u, o_v) \geq s(o_u, o_{v'})$;
16.             if such a pair or pairs exists, then
17.                 remove ($o_u$, $o_v$) and its subtree from
    heap 1;
18.             return ($o_u$, $o_v$);
19.             update heap1;
20.     }
21. retrieve the top pair ($o_u$, $o_{v'}$) from heap 2;
22. return all of the path pairs between $o_u$and$o_{v'}$ as top $k$ pairs;

    getNextBestPair ()
1.  get the top pair ($o_u$, $o_v$) from the heap1;
2.  for( i=2; i ≤ k; i++)
3.  {
4.      if the next best guest of $o_u$ exists, then
5.  $o_{v'}$ ← the next best guest of $o_u$;
6.          i = i+1;
7.      else go to step 18;
8.   }
9.   insert the pair ($o_u$, $o_{v'}$) into heap 2 with score
    $s(o_u, o_{v'})$;
10. return ($o_u$, $o_{v'}$);
```

The same technique can be applied in the case of all of its variants such as homochromatic and heterochromatic object pairs queries. In the case of chromatic queries, the best guest of an object o_u is the right adjacent object of o_u that meet the color requirement. That is, in the case of homochromatic queries, the best guest of o_u is the right adjacent object of o_u with the same color as that of o_u. Similarly, in the case of heterochromatic queries, the best guest of o_u is the right adjacent object of $o_u$with a different color.

Top-k pairs queries on multi-valued objects: A multi-valued object means an object that having multiple instances. The procedure for answering Top-*k*pairs queries on multi-valued objects is given in [1]. In the case of multi-valued objects, an object is represented by using the set of its instances (for eg: let U be an object, then it is defined as $U = \{u_1,\ldots,u_n\}$) and top-*k* pairs are defined by using ∅-quantile scores (see [1]).

Exclusive top-k pairs queries: Let, S be the set of top-*k* pairs, then we say that an object o_u satisfies the exclusiveness constraint if $o_u$appears at most once in the set S. An exclusive top-*k*pairs query retrieves top-*k* pairs in which every object satisfies the exclusiveness constraint. Its procedure also discussed in paper [1]. In the case of exclusive object pairs queries, the situation is different and our proposed solution require slight modifications.

IV. PERFORMANCE EVALUATION

We implement our proposed framework in a real world dataset of location data. For that purpose we use two datasets of different size. Here, we use *n* to denote the size of the dataset or the number of objects in a dataset and we use *k*to denote the required number of pairs. One dataset contain 100 residential blocks and another dataset contain 300 residential blocks from USA. Where, each residential block corresponds to the location of a town. Our both datasets contain 10 different towns numbered from 0 to 9. In first dataset each town contain 10 residential blocks and in second dataset each town contain 30 residential blocks. Residential blocks that found within the same town have assigned the same color and residential blocks that found within different towns have assigned different colors. Here, each location has four attributes. They are two location coordinates, population and average rent of the properties in the location. Then we run different queries on this dataset and evaluate the performance of our algorithm.

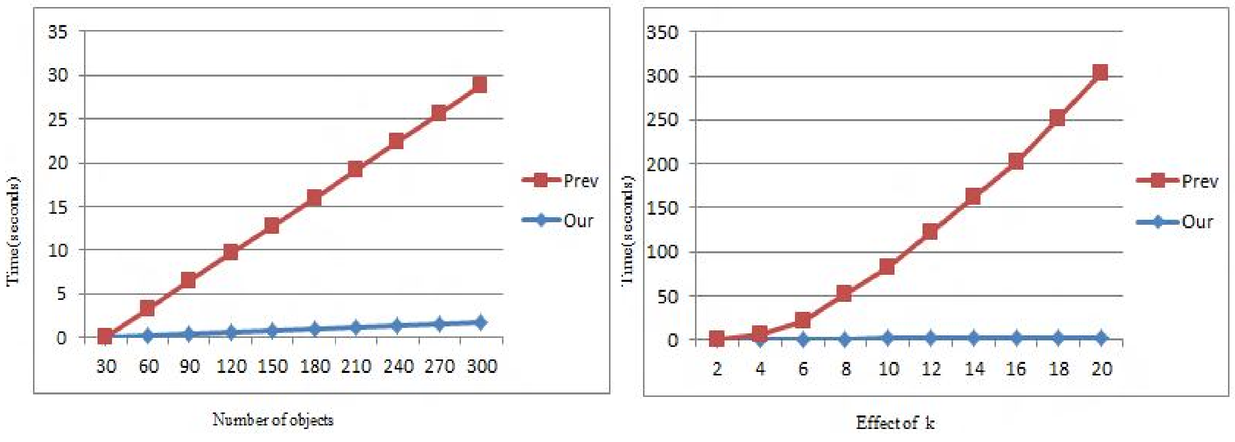

Fig. 2(a) shows Computation time versus the number of objectswhen *k* is a constant and Fig. 2(b) shows Computation time versus the effect of*k* value when *n* is a constant

The computation time required for the existing system increases as the number of data objects in the dataset increases as well as the *k* values increases. But, the performance of our proposed method is better than its previous version [1] because it requires only fewer computations even if the dataset is very large. This is shown in above figures. Fig. 2(a)shows the effect of the size of the dataset on computation time in the case of existing system and in the case of our proposed system when *k* is a constant such as 5.But, Fig. 2(b) shows the effect of *k* values on computation time in the case of existing system and in the case of our proposed system when *n* is a constant such as 300. In these figures, ‘Our’ denotes our proposed solution and ‘Prev’ denotes the previously existing solution.

V. CONCLUSION AND FUTURE WORK

This paper proposes a new unified framework for answering *k* closest pairs queries and their variants by using the concept of grouping of data objects. This grouping of data objects is used to reduce the number of computations and thus to saving the computational time. In the existing system, the algorithm for returning top-*k* pairs is applied to the whole dataset. So, when the dataset is very large, this procedure requires a large amount of time. Therefore, in order to overcome that problem, we introduce the concept of grouping of objects. According to this concept, after sorting all the objects in a dataset, we find pairs such that each pair contains a data object and its best guest. Then we insert all the pairs into the heap and then to reduce the heap size we use an efficient algorithm and an equation for score comparison. Then we apply all of the operations for finding top *k* pairs in that reduced heap only. In future, we can find efficient methods for answering k-closest groups queries and variants by using grouping of objects and other methods that work efficiently. Like its previous version, our proposed solution requires only limited main memory, it does not require any pre-built indexes and is easy to implement.

REFERENCES.

1. M.A. Cheema, X. Lin, H. Wang, J. Wang, and W. Zhang, "A Unified Framework for Answering k Closest Pairs Queries and Variants", IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 11, November 2014.
2. M.A. Cheema, X. Lin, H. Wang, J. Wang, and W. Zhang, "A Unified Approach for Computing Top-K Pairs in Multidimensional Space," Proc. 26th Int'l Conf. Data Eng. (ICDE), pp. 1031-1042, 2011.
3. G.R. Hjaltason and H. Samet, "Incremental Distance Join Algorithms for Spatial Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data, 1998.
4. A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest Pair Queries in Spatial Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data, 2000.
5. J. Shan, D. Zhang, and B. Salzberg, "On Spatial-Range Closest-Pair Query," Proc. Symp. Advances in Spatial and Temporal Databases (SSTD), pp. 252-269, 2003.
6. H. Shin, B. Moon, and S. Lee, "Adaptive Multi-Stage Distance Join Processing," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 343-354, 2000.
7. K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous Monitoring of Top-K Queries over Sliding Windows," Proc. ACM SIGMOD Int'l Conf. Management of Data, 2006.
8. K. Mehlhorn, "Computational Geometry," Data Structures and Algorithms, vol. 3, pp. 79-268, Springer, 1984.
9. M. Smid, "Closest-Point Problems in Computational Geometry, "Handbook on Computational Geometry, Elsevier Science, 1997.
10. A. Yang and K.-I. Lin, "An Index Structure for Improving Nearest Closest Pairs and Related Join Queries in Spatial Databases," Proc. Int'l Symp. Database Eng. & Applications (IDEAS), 2002.
11. Z. Shen, M.A. Cheema, X. Lin, W. Zhang, and H. Wang, "Efficiently Monitoring Top-K Pairs over Sliding Windows," Proc. IEEE 28th Int'l Conf. Data Eng. (ICDE), 2012.
12. W. Zhang, L. Zhan, Y. Zhang, M.A. Cheema, and X. Lin, "Efficient Top-K Similarity Join Processing over Multi-Valued Objects, "World Wide Web, vol. 17, no. 3, pp. 285-309, 2014.
13. Z. Shen, M.A. Cheema, X. Lin, W. Zhang, and H. Wang, "A Generic Framework for Top-K Pairs and Top-K Objects Queries over Sliding Windows," IEEE Trans. Knowledge and Data Eng, preprint, 19 Sept. 2012, doi:10.1109/TKDE.2012.181.
14. I.F. Ilyas, G. Beskales, and M.A. Soliman, "A Survey of Top-k Query Processing Techniques in Relational Database Systems, "ACM Computing Surveys, vol. 40, no. 4, article 11, 2008.
15. R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," J. Computer and System Sciences, vol. 66, no. 4, pp. 614-656, 2003.

BIOGRAPHY

Mintu Thomas is an M-Tech student in the Computer Science and Engineering Department, Mangalam College of Engineering, M. G. University. She received B-Tech degree in 2012 from College of Engineering, Kidangoor, CUSAT University, India. Her research interests are Knowledge and Data Management, Data Mining, Databases etc.

Neena Joseph is Associate Professor in the Department of Computer Science and Engineering, M.G. University, Mangalam college of Engineering, Ettumanoor, Kerala, India.