The KNDN Problem: A Quest for Unity in Diversity

Jayant R. Haritsa
Indian Institute of Science, Bangalore
`haritsa@dsl.serc.iisc.ernet.in`

Abstract

Given a query location Q in multi-dimensional space, the classical KNN problem is to return the K spatially closest answers in the database with respect to Q. The KNDN (K-Nearest Diverse Neighbor) problem is a semantic extension where the objective is to return the spatially closest result set such that each answer is sufficiently different, or diverse, from the rest of the answers. We review here the KNDN problem formulation, the associated technical challenges, and candidate solution strategies.

1 Introduction

Over the last decade, the issue of diversity in query results produced by information retrieval systems, such as search engines, has been investigated in great depth (see [11] for a recent survey). Curiously, however, there has been relatively little diversity-related work with regard to queries issued on traditional *database systems*. Even more surprisingly, this paucity has occurred in spite of extensive research, during the same period, on supporting vanilla distance-based Top-K (or equivalently, KNN) queries in a host of database environments including online business databases, multimedia databases, mobile databases, stream databases, etc. (see [2, 6, 7] for detailed surveys). The expectation that this prior work would lead on to a vigorous investigation of result-set *semantics*, including aspects such as diversity, has unfortunately not fully materialized. In this article, we review our 2004 study of database result diversity [9], in the hope that it may rekindle interest in the area. Specifically, we cover the KNDN (K-Nearest Diverse Neighbor) problem formulation, the associated technical challenges, and candidate solution strategies.

Motivation. The general model of a KNN query is that there is a database of multidimensional objects against which users submit point queries. Given a metric for measuring distances in the multidimensional space, the system is expected to find the K objects in the database that are spatially closest to the location of the user's query. Typical distance metrics include the Euclidean and Manhattan norms. An example KNN application scenario is given below.

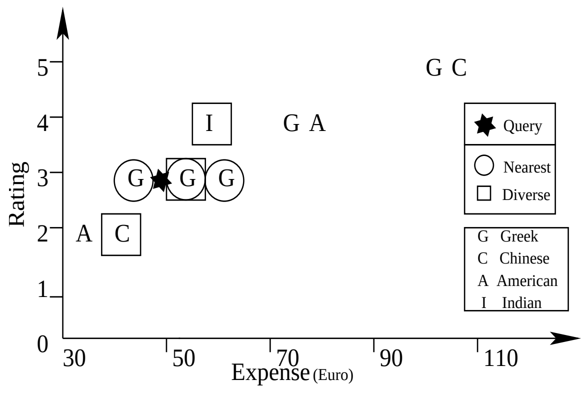
Example 1: Consider the situation where the London tourist office maintains the relation RESTAURANT (Name,Speciality,Rating,Expense), where Name is the name of the restaurant; Speciality indicates the food type (Greek, Chinese, Indian, etc.); Rating is an integer between 1 to 5 indicating restaurant quality; and, Expense is the typical expected expense per person. In this scenario, a visitor to London may wish to submit

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

the KNN query shown in Figure 1(a) to have a choice of three nearby mid-range restaurants where dinner is available for around 50 Euros. (The query is written using the SQL-like notation described in [2].)

```
SELECT * FROM RESTAURANT
WHERE Rating=3 and Expense=50
ORDER 3 BY Euclidean
```

(a) KNN Query



(b) Data Distribution

Figure 1: Motivational Example

For the sample distribution of data points in the RESTAURANT database shown in Figure 1(b), the above KNN query would return the answers shown by the circles. Note that these three answers are very similar: {*Parthenon-α,3,Greek,54*}, {*Parthenon-β,3,Greek,45*}, and {*Parthenon-γ,3,Greek,60*}. In fact, it is even possible that the database may contain exact *duplicates* (wrt the Speciality, Rating and Expense attributes).

Returning three very similar answers may not offer much value to the London tourist. Instead, she might be better served by being told, in addition to {*Parthenon-α,3,Greek,54*}, about a Chinese restaurant {*Hunan,2,Chinese,40*}, and an Indian restaurant {*Taj,4,Indian,60*}, which would provide a close but more heterogeneous set of choices to plan her dinner – these answers are shown by rectangles in Figure 1(b). In short, the user would like to have not just the closest set of answers, but the closest *diverse* set of answers (an oft-repeated quote from Montaigne, the sixteenth century French writer, is *"The most universal quality is diversity"* [15]).

2 The KNDN Problem

Based on the above motivation, we introduced five years ago the **KNDN** (K-Nearest Diverse Neighbor) problem in [9]. The objective here is to identify the spatially closest set of answers to the user's query location such that each answer is sufficiently diverse from all the others in the result set.

We began by modeling the database as composed of N tuples over a D-dimensional space with each tuple representing a point in this space. The user specifies a point query Q over an M-sized subset of these attributes $S(Q) : (s_1, s_2, \ldots, s_M)$, referred to as "spatial attributes"; and lists a L-sized subset of attributes $V(Q) : (v_1, v_2, \ldots, v_L)$, on which she would like to have result diversity – these attributes are referred to as "diversity attributes". The space formed by the diversity attributes is referred to as *diversity-space*. Note that the choice of the diversity attributes is *orthogonal* to the choice of the spatial attributes. Finally, the user also specifies K, the number of desired answers, and the expected result is a set of K diverse tuples from the database. Referring back to the London tourist example, the associated settings in this framework are $D = 4, M = 2, s_1 = Rating, s_2 = Expense, L = 1, v_1 = Speciality, K = 3$.

For ease of exposition, we will hereafter assume that the domains of all attributes are numeric and normalized to the range $[0, 1]$, and that all diversity dimensions are equivalent in the user's perspective. The relaxation of these assumptions is discussed in [8].

2.1 Diversity Measure

The next, and perhaps most important, question we faced was how to define diversity and quantify it as a measure. Here, there were a variety of options, such as hashing-based schemes [1], or taxonomy-based schemes [10], but all these seem primarily suited for the largely unstructured domain of IR applications. In the structured database world, however, we felt that diversity should have a direct *physical association* with the data tuples for it to be meaningful to users. Accordingly, we chose our definition of diversity based on the classical *Gower coefficient* [3]. Here, the difference between two points is defined as a weighted average of the respective attribute value differences. Specifically, we first compute the (absolute) differences between the attribute values of these two points in *diversity-space*, sequence these differences in *decreasing* order of their values, and then label them as $(\delta_1, \delta_2, \ldots, \delta_L)$. Next, we calculate $divdist$, the diversity distance between points P_1 and P_2 with respect to diversity attributes $V(Q)$ as

$$divdist(P_1, P_2, V(Q)) = \sum_{j=1}^{L} (W_j \times \delta_j) \quad (1)$$

where the W_j's are weighting factors for the differences. Since all δ_j's are in the range $[0, 1]$ (recall that the values on all dimensions are normalized to $[0, 1]$), and by virtue of the W_j assignment policy discussed below, diversity distances are also bounded in the range $[0, 1]$.

The assignment of the weights is based on the principle that *larger weights* should be assigned to the *larger differences*. That is, in Equation 1, we need to ensure that $W_i \geq W_j$ if $i < j$. The rationale for this assignment is as follows: Consider the case where point P_1 has values $(0.2, 0.2, 0.3)$, point P_2 has values $(0.19, 0.19, 0.29)$ and point P_3 has values $(0.2, 0.2, 0.27)$. Consider the diversity of P_1 with respect to P_2 and P_3. While the aggregate difference is the same in both cases, yet intuitively we can see that the pair (P_1, P_2) is more homogeneous as compared to the pair (P_1, P_3). This is because P_1 and P_3 differ considerably on the third attribute as compared to the corresponding differences between P_1 and P_2. In a nutshell, a pair of points that have higher variance in their attribute differences are taken to be more diverse than a pair with lower variance.

Now consider the case where P_3 has values $(0.2, 0.2, 0.28)$. Here, although the aggregate difference is higher for the pair (P_1, P_2), yet again it is pair (P_1, P_3) that appears more diverse since its difference on the third attribute is larger than any of the individual differences in pair (P_1, P_2).

Based on the above observations, we decided that the weighting function should have the following properties: First, all weights should be positive, since having differences in any dimension should never decrease the diversity. Second, the sum of the weights should add up to 1 (i.e., $\sum_{j=1}^{L} W_j = 1$) to ensure that $divdist$ values are normalized to the $[0, 1]$ range. Third, the weights should be *monotonically decaying* ($W_i \geq W_j$ if $i < j$) to reflect the preference given to larger differences. Finally, the weighting function should ideally be tunable using a single parameter.

Example 2: A candidate weighting function that materially satisfies the above requirements is the following:

$$W_j = \frac{a^{j-1} \times (1 - a)}{1 - a^L} \quad (1 \leq j \leq L) \quad (2)$$

where a is a tunable parameter over the range $(0, 1)$. Note that this function implements a *geometric* decay, with the parameter 'a' determining the rate of decay. Values of a that are close to 0 result in faster decay, whereas values close to 1 result in slow decay. When the value of a is nearly 0, almost all weight is given to the maximum difference i.e., $W_1 \simeq 1$, modeling the L_∞ (i.e., Max) distance metric, and when a is nearly 1, all attributes are given similar weights, modeling a L_1 (i.e., Manhattan) distance metric.

Figure 2 shows, for different values of the parameter 'a' in Equation 2, the locus of points which have a diversity distance of 0.1 with respect to the origin $(0, 0)$ in two-dimensional diversity-space.

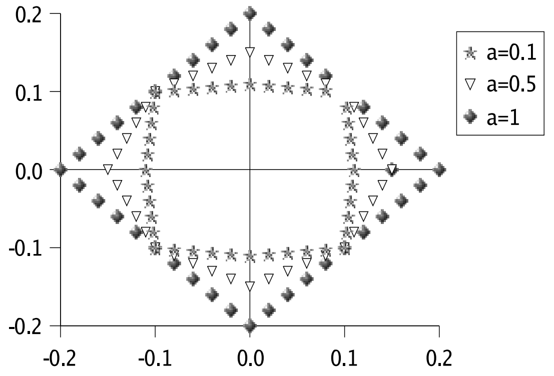


Figure 2: Locus of Diversity=0.1 Points

```
SELECT * FROM RESTAURANT
WHERE Rating=3 and Expense=50
ORDER 3 BY Euclidean
WITH MinDiv=0.1 ON Speciality
```

Figure 3: KNDN Query Example

2.1.1 Minimum Diversity Threshold

The next issue we faced was deciding the mechanism by which the user specified the extent to which diversity was required in the result set. While complicated functions could be constructed for this purpose, we felt that typical users of database systems would be only willing, or knowledgeable enough, to provide a single value characterizing their requirements. Accordingly, we supported a single threshold parameter *MinDiv*, ranging between $[0, 1]$. Given this threshold setting, two points are diverse if the diversity distance between them is greater than or equal to *MinDiv*. That is,

$$DIV(P_1, P_2, V(Q)) = true \;\; iff \;\; divdist(P_1, P_2, V(Q)) \geq MinDiv \tag{3}$$

The advantage of the above simple definition is that it is amenable to a *physical* interpretation that can guide the user in determining the appropriate setting of *MinDiv*. Specifically, the interpretation is that if a pair of points is deemed to be diverse, then these two points have a difference of *MinDiv* or more on atleast one diversity dimension. For example, a *MinDiv* of 0.1 means that any pair of diverse points differ in atleast one diversity dimension by atleast 10% of the associated domain size. In practice, we would expect that *MinDiv* settings would be on the low side, typically not more than 0.2. Finally, note that the DIV function is commutative but not transitive.

The notion of pair-wise diversity is easily extended to the final user requirement that each point in the result set must be diverse with respect to *all* other points in the set. That is, given a result set $\mathcal{A}$ with answer points $A_1, A_2, \ldots, A_K$, we require $DIV(A_i, A_j, V(Q)) =$ true $\forall\ i, j$ such that $i \neq j$ and $1 \leq i, j \leq K$. We call such a result set to be *fully-diverse*.

With the above framework, setting *MinDiv* to zero results in the traditional KNN query, whereas higher values provide more and more importance to diversity at the expense of distance. The London tourist query can now be stated as shown in Figure 3, where *Speciality* is the attribute on which diversity is calculated, and the goal is to produce the spatially closest fully-diverse result set.

2.1.2 Integrating Diversity and Distance

Given a user query location and a diversity requirement, there may be *multiple* fully-diverse answer sets available in the database. Ideally, the set that should be returned is the one that has the closest spatial characteristics. To identify this set, we need to first define a single value that characterizes the aggregate spatial distances of a set of points, and this is done in the following manner: Let function $SpatialDist(P, Q)$ calculate the spatial distance of point P from query point Q. The choice of *SpatialDist* function is based on the user specification and could be any monotonically increasing distance function such as Euclidean, Manhattan, etc. We combine distances of all points in a set into a single value using an aggregate function Agg which captures the overall distance of the set from Q. While a variety of aggregate functions are possible, the choice is constrained by the fact that the aggregate function should ensure that as the individual points in the set move farther away from the query, the distance of the set should also increase correspondingly. Sample aggregate functions which obey this constraint include the Arithmetic, Geometric, and Harmonic Means. Finally, we use the reciprocal of the aggregate to determine the "closeness score" of the (fully-diverse) result set. Putting all these formulations together, given a query Q and a candidate fully-diverse result set $\mathcal{A}$ with points $A_1, A_2, \ldots, A_K$, the closeness score of $\mathcal{A}$ with respect to Q is computed as

$$CloseScore(\mathcal{A}, Q) = \frac{1}{Agg(SpatialDist(Q, A_1), \ldots, SpatialDist(Q, A_K))} \tag{4}$$

2.1.3 Problem Formulation

The final KNDN problem formulation is as follows: *Given a point query Q on a D-dimensional database, a* MinDiv *diversity threshold on a set of diversity dimensions V(Q), and a desired diverse-result cardinality of K, the goal of the K-Nearest Diverse Neighbor (KNDN) problem is to find the set of K mutually diverse tuples in the database, whose CloseScore is the maximum, after including the spatially nearest tuple to Q in the result set.*

The requirement that the nearest point to the user's query should *always* form part of the result set is because this point, in a sense, *best fits* the user's query. Further, the nearest point A_1 serves to seed the result set since the diversity function is meaningful only for a *pair* of points. Since point A_1 of the result is fixed, the result sets are differentiated based on their remaining $(K - 1)$ choices.

3 Solution Techniques

Having described the KNDN problem formulation, we now move on to discussing the issues related to developing solutions for this problem.

Not surprisingly, finding the *optimal* (wrt CloseScore) fully-diverse result set for the KNDN problem turns out to be computationally hard. We can establish this by mapping KNDN to the well known *independent set problem* [4] which is NP-complete. The mapping is achieved by forming a graph corresponding to the dataset in the following manner: Each tuple in the dataset forms a node in the graph and a edge is added between two nodes if the diversity between the associated tuples is less than *MinDiv*. Now any independent set of nodes, that is, a subgraph in which no two nodes are connected, of size K in this graph represents a fully-diverse set of K tuples. But finding *any* independent set, let alone the optimal independent set, is itself computationally hard. Tractable solutions to the independent set problem have been proposed [4], but they require the graph to be sparse and all nodes to have a bounded small degree. In our world, this translates to requiring that all the clusters in diversity-space should be small in size. But, this may not be typically true for the datasets encountered in practice, rendering these solutions not generally applicable, and forcing us to go in for heuristic solutions instead.

3.1 Inadequacy of Clustering-based Solutions

At first glance, it may appear that a simple heuristic solution to the KNDN problem would be to initially group the data into clusters using algorithms such as BIRCH [13], replace all clusters by their representatives, and then apply the traditional KNN approach on this summary database. There are two problems here: First, since the clusters are pre-determined, there is no way to dynamically specify the desired diversity, which may vary from one user to another or may be based on the specific application that is invoking the KNDN search. Second, since the query attributes are not known in advance, we potentially need to identify clusters in all possible subspaces, which may become infeasible due to the exponential number of such subspaces. Finally, this approach cannot provide the traditional KNN results.

Yet another approach to produce a diverse result set could be to run the standard KNN algorithm, cluster its results, replace the clusters by their representatives, and then output these representatives as the diverse set. The problem with this approach is that it is not clear how to apriori determine the original number of required answers such that there are finally K diverse representatives. If the original number is set too low, then the search process has to be restarted, whereas if the original number is set too high, a lot of wasted work ensues.

3.2 The MOTLEY Algorithm

In light of the above, we designed a diversity-conscious greedy algorithm called **MOTLEY**[1] for solving the KNDN problem, which is described in the remainder of this section.

3.2.1 Database Navigation

A major design issue in Motley was the mechanism chosen to navigate through the database. Two primary approaches for database navigation have been proposed in the extensive literaure on the vanilla KNN problem – the first is based on using standard database statistics (e.g., [2]), while the other is based on distance browsing using spatial indices such as the R-tree (e.g., [6, 12]). We opted to use the latter approach since the detailed experimental evaluation of [6] had indicated that distance browsing outperforms the alternative approaches – further, distance browsing can be seamlessly used for both KNN and KNDN problems.

In the distance browsing approach, the database tuples are processed incrementally in increasing order of their distance from the query location. The approach is predicated on having a containment-based index structure, such as the R-tree[5], built collectively on all dimensions of the database (more precisely, the index needs to cover only those attributes on which spatial predicates may appear in the query workload). This assumption appears practical since most current database systems natively support R-trees.

To implement distance browsing, a priority queue, *pqueue*, is maintained which is initialized with the root node of the R-Tree. The *pqueue* maintains the R-Tree nodes and data tuples in increasing order of their distance from the query location. While the distance between a data point and the query Q is computed in the standard manner, the distance between a R-tree node and Q is computed as the minimum of the distances between Q and all points in the region enclosed by the MBR (Minimum Bounding Rectangle) of the R-tree node. The distance of a node from Q is zero if Q is within the MBR of that node, otherwise it is the distance of the closest point on the MBR periphery. For this, we first need to compute the distances between the MBR and Q along each query dimension – if Q is inside the MBR on a specific dimension, the distance is zero, whereas if Q is outside the MBR on this dimension, it is the distance from Q to either the low end or the high end of the MBR, whichever is nearer. Once the distances along all dimensions are available, they are combined (based on the distance metric in operation) to get the effective distance.

To return the next nearest neighbor, we pick up the first element of the *pqueue*. If it is a tuple, it is immediately returned as the sought for neighbor. However, if the element is an R-tree node, all the children of that node are inserted in the *pqueue*. During this insertion process, the *spatial* distance of the object from the query point is calculated and used as the insertion key. This process is repeated until we find a tuple to be the first element of the queue, which is then returned.

The above distance browsing process continues until either the diverse result set is found, or until all points in the database are exhausted, signaled by the *pqueue* becoming empty.

[1] Motley: A collection containing a variety of sorts of things [14].

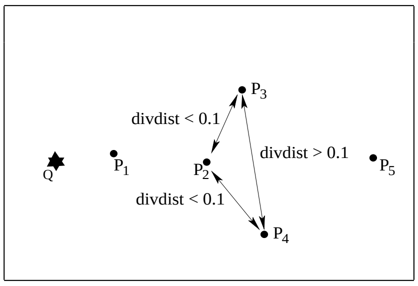


Figure 4: Immediate Greedy

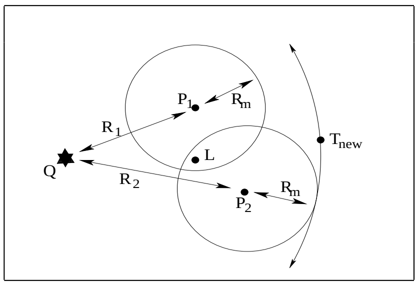


Figure 5: Buffered Greedy

3.2.2 Immediate Greedy

The first solution we attempted is called ImmediateGreedy (IG), wherein distance browsing is used to simply access database tuples in increasing der of their spatial distances from the query point, as discussed above. The first tuple is always inserted into the result set, $\mathcal{A}$, to satisfy the requirement that the closest tuple to the query point must figure in the result set. Subsequently, each new tuple is added to $\mathcal{A}$ if it is diverse with respect to *all* tuples currently in $\mathcal{A}$; otherwise, it is discarded. This process continues until $\mathcal{A}$ grows to contain K tuples.

While the IG approach is straightforward and easy to implement, there are cases where it may make poor choices as shown in Figure 4. Here, Q is the query point, and P_1 through P_5 are the tuples in the database. Let us assume that the goal is to report 3 diverse tuples with *MinDiv* of 0.1. By inspection, we observe that the overall best choice could be $\{P_1,\ P_3,\ P_4\}$. But since $DIV(P_1, P_2, V(Q)) = true$, IG would include P_2 and this would then disqualify the candidatures of P_3 and P_4 as both $DIV(P_2, P_3, V(Q))$ and $DIV(P_2, P_4, V(Q))$ are false. Eventually, Immediate Greedy would give the solution as $\{P_1,\ P_2,\ P_5\}$. Worse, if point P_5 happens to be not present in the database, then this approach will fail to return a fully-diverse set even though such a set, namely $\{P_1,\ P_3,\ P_4\}$, is available.

3.2.3 Buffered Greedy

The above problems are addressed in the BufferedGreedy (BG) method by recognizing that IG gets into difficulty because, at all times, only the diverse points (or "leaders") in the result set, retained. To counter this, BG maintains with each leader a bounded buffered set of "dedicated followers" – a dedicated follower is a point that is not diverse with respect to a specific leader but is diverse with respect to *all remaining* leaders.

Given this additional set of dedicated followers, we adopt the heuristic that a current leader, L_i, is *replaced* in the result set by its dedicated followers $F_i^1, F_i^2, \ldots, F_i^j (j > 1)$ as leaders if (a) these dedicated followers are *all* mutually diverse, and (b) incorporation of these followers as leaders does not result in the premature disqualification of future leaders. The first condition is necessary to ensure that the result set contains only diverse points, while the second is necessary to ensure that we do not produce solutions that are worse than Immediate Greedy. For example, if in Figure 4, point P_5 had happened to be only a little farther than point P_4 such that $DIV(P_2, P_5, V(Q)) = true$, then the replacement of P_2 by P_3 and P_4 could be the wrong choice since $\{P_1, P_2, P_5\}$ may turn out to be the best solution.

To implement the second condition, we must know when it is "safe" to go ahead with a replacement i.e., when it is certain that all future leaders will be diverse from the current set of followers. To achieve this, we do the following: For each point, we consider a hypothetical sphere that contains all points in the domain space that may be non-diverse with respect to it. That is, we set the radius R_m of the sphere equal to the distance of the farthest non-diverse point in the domain space. Note that this sphere may contain some diverse points as well, but our aim is to take a conservative approach. Now, the replacement of a leader by selected dedicated followers can be done as soon as we have reached a distance greater than R_m with respect to the farthest follower from the query – this is because all future leaders will be diverse with respect to selected dedicated followers and there is no possibility of disqualification beyond this point. To clarify this technique, consider the following example:

Example 3: In Figure 5, the circles around P_1 and P_2 show the areas that contain all points that are not diverse with respect to P_1 and P_2, respectively. During the distance browsing process, when we access the point T_{new} (Figure 5), we know that all future points will be diverse from P_1 and P_2. At this time, if P_1 and P_2 are dedicated followers of L and mutually diverse, then we can replace L by $\{P_1,\ P_2\}$.

Our experimental results (detailed in [9, 8]) on synthetic and real data-sets, indicate that BufferedGreedy can provide high-quality diverse solutions at a low cost in terms of both result distance and processing time.

4 Future Directions

In this article, we reviewed the KNDN problem of finding the closest set of answers such that the user will find each answer sufficiently different from the rest, thereby adding value to the result set. This topic has been largely ignored by the database community, but appears to be a fecund source of interesting and challenging problems. For example, handling diversity in high-dimensional data, and efficient maintenance and usage of spatial index structures for arbitrary subsets of attributes, are promising immediate probems to be addressed. In the long term, modeling the KNDN operations within the query optimizer and integrating the solution with the query processing engine would be of practical import.

References

[1] A. Broder, M. Charikar, A. Frieze and M. Mitzenmacher, *Min-wise independent permutations*, Journal of Computer and System Sciences, 60(3), 2000.

[2] N. Bruno, S. Chaudhuri and L. Gravano, *Top-K Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation*, ACM Trans. on Database Systems, 27(2), 2002.

[3] J. Gower, *A general coefficient of similarity and some of its properties*, Biometrics 27, 1971.

[4] M. Grohe, *Parameterized Complexity for Database Theorists*, SIGMOD Record 31(4), December 2002.

[5] A. Guttman, *R-trees: A dynamic index structure for spatial searching*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1984.

[6] G. Hjaltason and H. Samet, *Distance Browsing in Spatial Databases*, ACM Trans. on Database Systems, 24(2), 1999.

[7] I. Ilyas, G. Beskales and M. Soliman, *A Survey of Top-k Query Processing Techniques in Relational Database Systems*, ACM Computing Surveys, 40(4), October 2008.

[8] A. Jain, P. Sarda and J. Haritsa, *Providing Diversity in K-Nearest Neighbor Query Results*, Tech. Report TR-2003-04, DSL/SERC, Indian Institute of Science, 2003.

[9] A. Jain, P. Sarda, and J. Haritsa, *Providing Diversity in K-Nearest Neighbor Query Results*, Proc. of 8th Pacific Asia Conf. on Knowledge Discovery and Data Mining, 2004.

[10] S. Gollapudi and A. Sharma, *An Axiomatic Approach for Result Diversification*, Proc. of World Wide Web Conf., 2009.

[11] E. Minack, G. Demartini and W. Nejdl, *Current Approaches to Search Result Diversification*, Proc. of 1st Intl. Workshop on Living Web, 2009.

[12] N. Roussopoulos, S. Kelley and F.Vincent, *Nearest Neighbor Queries*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1995.

[13] T. Zhang, R. Ramakrishnan and M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1996.

[14] `www.thefreedictionarity.com`

[15] `www.elibronquotations.com`