Providing Diversity in K-Nearest Neighbor Query Results

Anoop Jain, Parag Sarda, and Jayant R. Haritsa*

Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore 560012, INDIA.

Abstract. Given a point query Q in multi-dimensional space, K-Nearest Neighbor (KNN) queries return the K closest answers in the database with respect to Q. In this scenario, it is possible that a majority of the answers may be very similar to one or more of the other answers, especially when the data has clusters. For a variety of applications, such homogeneous result sets may not add value to the user. In this paper, we consider the problem of providing diversity in the results of KNN queries, that is, to produce the closest result set such that each answer is sufficiently different from the rest. We first propose a user-tunable definition of diversity, and then present an algorithm, called MOTLEY, for producing a diverse result set as per this definition. Through a detailed experimental evaluation, we show that MOTLEY can produce diverse result sets by reading only a small fraction of the tuples in the database. Further, it imposes no additional overhead on the evaluation of traditional KNN queries, thereby providing a seamless interface between diversity and distance.

Keywords: Nearest Neighbor, Distance Browsing, Result Diversity

1 Introduction

Over the last few years, there has been considerable interest in the database community with regard to supporting K-Nearest Neighbor (KNN) queries [8]. The general model of a KNN query is that the user gives a point query in multidimensional space and a distance metric for measuring distances between points in this space. The system is then expected to find, with regard to this metric, the K closest answers in the database from the query point. Typical distance metrics include Euclidean distance, Manhattan distance, etc.

It is possible that a majority of the answers to a KNN query may be very similar to one or more of the other answers, especially when the data has clusters. In fact, there may even be *duplicates* w.r.t. the attributes of the multidimensional space. For a variety of applications, such as online restaurant selection [6], providing homogeneous result sets may not add value to the user. It is our contention in this paper that, for such applications, the user would like to have not just the closest set of answers, but the closest *diverse* set of answers.

* Contact Author: haritsa@dsl.serc.iisc.ernet.in

Based on the above motivation, we consider here the problem of providing diversity in the results of KNN queries, that is, to produce the closest result set such that each answer is sufficiently diverse from the rest. We hereafter refer to this problem as the *K-Nearest Diverse Neighbor (KNDN)* problem, which to the best of our knowledge has not been previously investigated in the literature, and cannot be handled by traditional clustering techniques (details in [6]).

An immediate question that arises is how to define diversity. This is obviously a user-dependent choice, so we address the issue by providing a *tunable* definition, which can be set with a single parameter, *MinDiv*, by the user. *MinDiv* values range over [0,1] and specify the minimum diversity that should exist between *any pair* of answers in the result set. Setting *MinDiv* to zero results in the traditional KNN query, whereas higher values give more and more importance to diversity at the expense of distance.

Finding the optimal result set for KNDN queries is an *NP-complete problem* in general, and is computationally extremely expensive even for fixed K, making it infeasible in practice. Therefore, we present here an online algorithm, called **MOTLEY**[1], for producing a sufficiently diverse and close result set. MOTLEY adopts a greedy heuristic and leverages the existence of a spatial-containment-based multidimensional index, such as the R-tree, which is natively available in today's commercial database systems [7]. The R-tree index supports a "distance browsing" mechanism proposed in [5] through which database points can be efficiently accessed in increasing order of their distance from the query point. A pruning technique is incorporated in MOTLEY to minimize the R-tree processing and the number of database tuples that are examined.

Through a detailed experimental evaluation on real and synthetic data, we have found that MOTLEY can produce a diverse result set by reading only a small fraction of the tuples in the database. Further, the quality of its result set is very close to that provided by an off-line brute-force optimal algorithm. Finally, it can also evaluate traditional KNN queries without any added cost, thereby providing a *seamless interface between the orthogonal concepts of diversity and distance.*

2 Basic Concepts and Problem Formulation

In the following discussion, for ease of exposition and due to space limitations, we focus on a *restricted* instance of the KNDN problem – a significantly more general formulation is available in the full version of the paper [6].

We model the database as composed of N tuples over a D-dimensional space with each tuple representing a point in this space[2]. The domains of all attributes are numeric and normalized to the range [0,1]. The user specifies a point query Q over an M-sized subset of these attributes ($M \leq D$). We refer to these attributes as "point attributes". The user also specifies K, the number of desired answers, and a L-sized subset of attributes on which she would like to have diversity ($L \leq D$). We refer to these attributes as "diversity attributes" and the space formed by diversity attributes as *diversity-space*. Note that the *choice* of the diversity attributes is *orthogonal* to the *choice* of the point attributes. Finally, the result is a set of K database points.

[1] Motley: A collection containing a variety of sorts of things [9].

[2] We use point and tuple interchangeably in the remainder of this paper

Given that there are N points in the database and that we need to select K points for the result set, there are ${}^{N}C_{K}$ possible choices. We apply the diversity constraints first to determine the feasible sets and then bring in the notion of distance from the query point to make a selection from these sets. Viewed abstractly, we have a *two-level* scoring function: The first level chooses candidate result sets based on diversity constraints, and the second level selects the result set that is spatially closest to the query point.

Result Diversity We begin by defining *point diversity* and then, since the result is viewed as a set, extend the definition to *set diversity*. Point diversity is defined with regard to a *pair* of points and is evaluated with respect to the diversity attributes, $V(Q)$, mentioned in the query. Specifically, given points P_1, P_2 and $V(Q)$, the function $DIV(P_1, P_2, V(Q))$ returns true if P_1 and P_2 are diverse with respect to each other on the specified diversity attributes. A sample DIV function is described later in this section.

For a set to be *fully diverse*, all the points in the set should be mutually diverse. That is, given a result set $\mathcal{R}$ with points $R_1, R_2, \ldots, R_K$, we require $DIV(R_i, R_j, V(Q)) =$ true $\forall\, i, j$ such that $i \neq j$ and $1 \leq i, j \leq K$. For the restricted scenario considered here, we assume that at least one fully diverse result set is always available for the user query.

Diversity Function Our computation of the diversity between two points P_1 and P_2, is based on the classical *Gower coefficient* [2], wherein the difference between two points is defined as a weighted average of the respective attribute differences. Specifically, we first compute the differences between the attributed values of these two points in *diversity-space*, sequence these differences in *decreasing* order of their values, and then label them as $(\delta_1, \delta_2, \ldots, \delta_L)$. Now, we calculate $divdist$, the diversity distance between points P_1 and P_2 with respect to diversity attributes $V(Q)$ as

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

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 heuristic 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 $i < j$ (recall that δ_j's are sorted in decreasing order.) 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.

Now consider the case where P_3 has value (0.2, 0.2, 0.28). Here, although the aggregate δ_j 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 discussion, the weighting function should have the following properties: Firstly, all weights should be positive, since having difference 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. Finally, the weights should be *monotonically decaying* ($W_i \geq W_j$ if $i < j$) to reflect the preference given to larger differences.

Example 1. A candidate weighting function that obeys the above requirements is the following:

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

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 maximum difference i.e., $W_1 \simeq 1$, modeling (in the language of vector *p-norms*) 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.

Minimum Diversity Threshold We expect that the user provides a quantitative notion of the minimum diversity distance that she expects in the result set through a threshold parameter *MinDiv* that ranges between [0,1][3]. 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 $divdist(P_1, P_2, V(Q)) \geq MinDiv$.

The *physical* interpretation of the *MinDiv* value is that if a pair of points are 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. This physical interpretation can guide the user in determining the appropriate setting of *MinDiv*. In practice, we would expect that *MinDiv* settings would be on the low side, typically not more than 0.2. As a final point, note that with the above formulation, the DIV function is *symmetric* with respect to the point pair $\{P_1, P_2\}$. However, it is *not transitive* in that even if $DIV(P_1, P_2, V(Q))$ and $DIV(P_2, P_3, V(Q))$ are both true, it does not imply that $DIV(P_1, P_3, V(Q))$ is true.

Integrating Diversity and Distance Let function $SpatialDist(P, Q)$ calculate the spatial distance of point P from query point Q (this distance is computed with regard to the point attributes specified in 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

[3] This is similar to the user specifying minimum support and minimum confidence in association rule mining to determine what constitutes interesting correlations.

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 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 of the spatial distances of the result points from the query point to determine the score of the (fully diverse) result set. (Note that the *MinDiv* threshold only determines the identities of the fully diverse result sets, but not their scores.) Putting all these formulations together, given a query Q and a candidate fully diverse result set $\mathcal{R}$ with points $R_1, R_2, \ldots, R_K$, the score of $\mathcal{R}$ with respect to Q is computed as

$$Score(\mathcal{R}, Q) = \frac{1}{Agg(SpatialDist(Q, R_1), \ldots, SpatialDist(Q, R_K))} \quad (3)$$

Problem Formulation In summary, our problem formulation is as follows:

Given a point query Q on a D-dimensional database, a desired result cardinality of K, and a MinDiv threshold, the goal of the K-Nearest Diverse Neighbor (KNDN) problem is to find the set of K mutually diverse tuples in the database, whose score, as per Equation 3, is the maximum, after including the 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 R_1 serves to seed the result set since the diversity function is meaningful only for a *pair* of points. Since point R_1 of the result is fixed, the result sets are differentiated based on their remaining $K - 1$ choices.

An important point to note here is that when *MinDiv* is set to zero, all points (including duplicates) are *diverse* with respect to each other and hence the KNDN problem reduces to the traditional KNN problem.

3 The MOTLEY Algorithm

Finding the optimal result set for the KNDN problem is computationally hard. We can establish this (proof in [6]) by mapping KNDN to the well known *independent set problem* [3], which is NP-complete. Therefore, we present an alternative algorithm here called MOTLEY, which employs a greedy selection strategy in combination with a distance-browsing-based accessing of points; our experimental evaluation, presented later in Section 4, shows that the result sets obtained are extremely close to the optimal solution.

3.1 Distance Browsing

In order to process database tuples (i.e., points) incrementally, we adopt the "distance browsing" approach proposed in [5], through which it is possible to efficiently access data points in increasing order of their distance from the query point. This approach is predicated on having a containment-based index structure such as the R-Tree[4], built

collectively on all dimensions of the database (more precisely, the index needs to cover only those dimensions on which point predicates may appear in the query workload.)

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 point. 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.

Example 2. Consider an MBR, M, specified by ((1,1,1),(3,3,3)) in a 3-D space. Let $P_1(2,2,2)$ and $P_2(4,2,0)$ be two data points in this space. Then, $SpatialDist(M, P_1) = \sqrt{0^2+0^2+0^2} = 0$ and $SpatialDist(M, P_2) = \sqrt{(4-3)^2+0^2+(0-1)^2} = 1.414$.

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 next nearest neighbor. However, if the element is an R-tree node, all the children of that node are inserted in the *pqueue*. Note that during this insertion process, the *spatial* distance of the object from the query point is calculated and used as the insertion key. The insertion process is repeated until we get a tuple as 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.

3.2 Finding Diverse Results

We first present a simple greedy approach, called *Immediate Greedy*, and then its extension *Buffered Greedy*, for efficiently finding result sets that are both close and diverse.

Immediate Greedy Approach In the ImmediateGreedy (IG) method, tuples are sent in increasing order of their spatial distance from the query point using distance browsing, as discussed above. The first tuple is always inserted into the result set, $\mathcal{R}$, 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 $\mathcal{R}$ if it is diverse with respect to *all* tuples currently in $\mathcal{R}$; otherwise, it is discarded. This process continues until $\mathcal{R}$ grows to contain K tuples. Note that the result set obtained by this approach has following property: Let $\mathcal{B} = b_1, \ldots, b_K$ be the sequence formed by any other fully diverse set such that elements are listed in increasing order of spatial distance from Q. Now if i is the smallest index such that $b_i \neq R_i (R_i \epsilon \mathcal{R})$, then $SpatialDist(b_i, Q) \geq SpatialDist(R_i, Q)$.

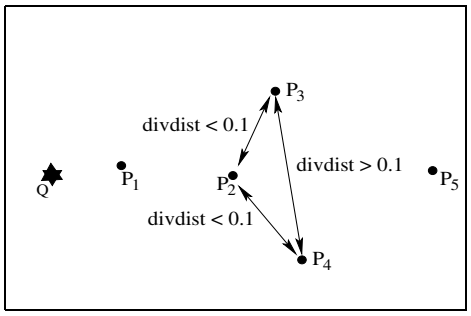

Fig. 1. Poor Choice by Immediate Greedy

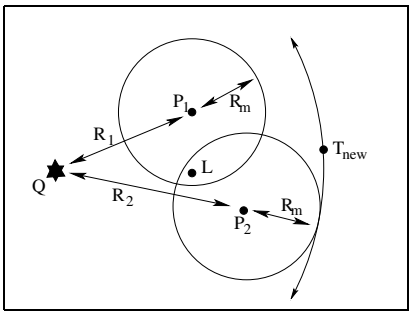

Fig. 2. Heuristic in Buffered Greedy Approach

While the IG approach is straight forward and easy to implement, there are cases where it may make poor choices as shown in Figure 1. 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. Clearly, $\{P_1,\ P_3,\ P_4\}$ satisfies the diversity requirement. Also $DIV(P_1, P_2, V(Q)) = true$. But inclusion of P_2 disqualifies the candidatures of P_3 and P_4 as both $DIV(P_2, P_3, V(Q)) = false$ and $DIV(P_2, P_4, V(Q)) = false$. By inspection, we observe that the overall best choice could be $\{P_1,\ P_3,\ P_4\}$, but Immediate Greedy would give the solution as $\{P_1,\ P_2,\ P_5\}$. Moreover, if point P_5 is 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.

Buffered Greedy Approach The above problems are addressed in the BufferedGreedy (BG) method by recognizing that in IG, only the diverse points (hereafter called "leaders") in the result set, are retained at all times. Specifically, 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. Our empirical results show that a buffer of capacity K points (where K is the desired result size) for each leader, is sufficient to produce a near-optimal solution. The additional memory requirement for the buffers is small for typical values of K and D (e.g., for K=10 and D=10, and using 8 bytes to store each attribute value, we need only 8K bytes of additional storage).

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 1, 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 need to 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 take the following approach: 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 to be 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 objective 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 2, 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. Due to distance browsing technique, when we access the point T_{new} (Figure 2), 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 $\{P_1,\ P_2\}$.

The integration of Buffered Greedy with distance browsing as well as pruning optimizations for minimizing the database processing are discussed in [6]. Further, the complexity of Buffered Greedy is shown to be $O(NK^2)$ in [6].

4 Experiments

We conducted a detailed suite of experiments to evaluate the quality and efficiency of the MOTLEY algorithm with regard to producing a diverse set of answers. While a variety of datasets were used in our experiments (see [6] for details), we report on only one dataset here, namely Forest Cover [10], a real dataset containing 581,012 tuples and 4 attributes representing *Elevation*, *Aspect*, *Slope*, and *Distance*.

Our experiments involve uniformly distributed point queries across the whole data space, with the attribute domains normalised to the range [0, 1]. The default value of K, the desired number of answers, was 10, unless mentioned otherwise, and *MinDiv* was varied across [0, 1]. In practice, we expect that *MinDiv* settings would be on the low side, typically not more than 0.2, and we therefore focus on this range in our experiments. The decay rate (a) of the weights (Equation 2) was set to 0.1, Harmonic Mean was used for the Agg function (Equation 3), and spatial distances were computed using the Euclidean metric. The R-tree (specifically, the R* variant [1]) was created with a fill factor of 0.7 and branching factor 64.

Result-set Quality We begin by characterizing the quality of the result set provided by MOTLEY, which is a greedy online algorithm, against an off-line brute-force optimal algorithm. This performance perspective is shown in Figure 3, which presents the average and worst case ratio of the result set scores. As can be seen in the figure, the average case is almost optimal (note that the Y-axis of the graph begins from 0.8), indicating that MOTLEY typically produces a *close-to-optimal* solution. Moreover, even in

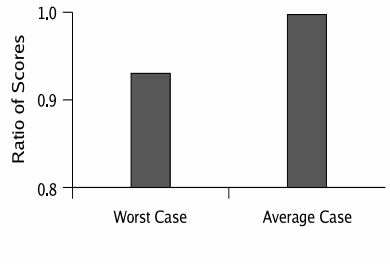

Fig. 3. MOTLEY vs. Optimal

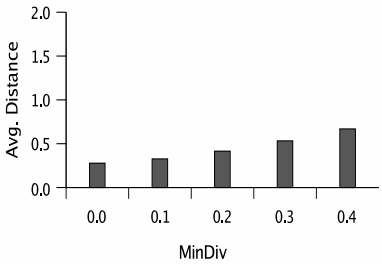

Fig. 4. MOTLEY vs. KNN

the worst-case, the difference is only around 10 percent. Importantly, even in the situations where MOTLEY did not provide the complete optimal result set, the errors were mostly restricted to only *one or two* points, out of the total of ten answers.

Figure 4 shows, as a function of *MinDiv*, the average *distance* of the points in MOTLEY's result set – this metric effectively captures the cost to be paid in terms of distance in order to obtain result diversity. The important point to note here is that for values of *MinDiv* up to 0.2, the distance increase is *marginal*, with respect to the traditional KNN query (*MinDiv* = 0). Since, as mentioned earlier, we expect that users will typically use *MinDiv* values between 0 and 0.2, it means that *diversity can be obtained at relatively little cost in terms of distance*.

Execution Efficiency Having established the high-quality of MOTLEY answers, we now move on to evaluating its execution efficiency. In Figure 5, we show the average fraction of tuples read to produce the result set as a function of *MinDiv*. Note firstly that the tuples scanned are always less than 15% of the complete dataset. Secondly, at lower values of *MinDiv*, the number of tuples read are small because we obtain K diverse tuples after processing only a small number of points, whereas at higher values of *MinDiv*, pruning is more effective and hence the number of tuples processed continues to be small in comparison to the database size.

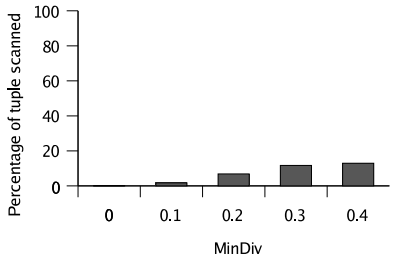

Fig. 5. Execution Efficiency of MOTLEY

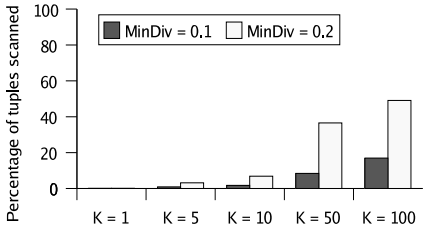

Fig. 6. Effect of K

Effect of K We also evaluated the effect of K, the number of answers, on the algorithmic performance. Figure 6 shows the percentage of tuples read as a function of *MinDiv* for different values of K ranging from 1 to 100. For $K = 1$, it is equivalent to the traditional NN search, irrespective of *MinDiv*, due to requiring the closest point to form part of the result set. As the value of K increases, the number of tuples read also increases, especially for higher values of *MinDiv*. However, we can expect that users will specify lower values of *MinDiv* for large K settings.

5 Conclusions

In this paper, we introduced the problem of finding the K Nearest Diverse Neighbors (KNDN), where the goal is to find 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. We provided a quantitative notion of diversity that ensured that two tuples were diverse if they differed in at least one dimension by a sufficient distance, and presented a two-level scoring function to combine the orthogonal notions of distance and diversity.

We described MOTLEY, an online algorithm for addressing the KNDN problem, based on a buffered greedy approach integrated with a distance browsing technique. Pruning optimizations were incorporated to improve the runtime efficiency. Our experimental results demonstrated that MOTLEY can provide high-quality diverse solutions at a low cost in terms of both result distance and processing time. In fact, MOTLEY's performance was close to the optimal in the average case and only off by around ten percent in the worst case.

Acknowledgements This work was supported in part by a Swarnajayanti Fellowship from the Dept. of Science & Technology, Govt. of India.

References

1. N. Beckmann, H. Kriegel, R. Schneider and B. Seeger, *The R*-tree: An efficient and robust access method for points and rectangles*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1990.
2. J. Gower, *A general coefficient of similarity and some of its properties*, Biometrics 27, 1971.
3. M. Grohe, *Parameterized Complexity for Database Theorists*, SIGMOD Record 31(4), December 2002.
4. A. Guttman, *R-trees: A dynamic index structure for spatial searching*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1984.
5. G. Hjaltason and H. Samet, *Distance Browsing in Spatial Databases*, ACM Trans. on Database Systems, 24(2), 1999.
6. 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.
7. R. Kothuri, S. Ravada and D. Abugov, *Quadtree and R-tree indexes in Oracle Spatial: A comparison using GIS data*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 2002.
8. N. Roussopoulos, S. Kelley and F.Vincent, *Nearest Neighbor Queries*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1995.
9. *www.thefreedictionarity.com*.
10. ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype