The Fractal Dimension Making Similarity Queries More Efficient

Adriano S. Arantes, Marcos R. Vieira, Agma J. M. Traina, Caetano Traina Jr.
Computer Science Department - ICMC
University of Sao Paulo at Sao Carlos
Avenida do Trabalhador Sao-Carlense, 400
13560-970 - Sao Carlos, SP - Brazil
Phone: +55 16-273-9693
{arantes | mrvieira | agma | caetano}@icmc.usp.br

ABSTRACT

This paper presents a new algorithm to answer k-nearest neighbor queries called the Fractal k-Nearest Neighbor (k-$NNF()$). This algorithm takes advantage of the fractal dimension of the dataset under scan to estimate a suitable radius to shrinks a query that retrieves the k-nearest neighbors of a query object. k-$NN()$ algorithms starts searching for elements at any distance from the query center, progressively reducing the allowed distance used to consider elements as worth to analyze. If a proper radius can be set to start the process, a significant reduction in the number of distance calculations can be achieved. The experiments performed with real and synthetic datasets over the access method Slim-tree, have shown that the efficiency of our approach makes the total processing time to drop up to 50%, while requires 25% less distance calculations.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*;
H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS, Image databases*

General Terms

Algorithms, Measurement

Keywords

fractals, intrinsic dimension, nearest neighbors queries, similarity queries

1. INTRODUCTION

Queries asking for equality in complex data domains, such as, image, video, spatial information, genomic sequences, time series, among others, are not useful, and the similarity between pairs of elements is the most important property in such domains [5]. Thus, a new class of queries, based on algorithms that search for similarity among elements emerged as more adequate to manipulate these data. To be able to apply similarity queries over elements of a data domain, a dissimilarity function, or a "distance function", must be defined on the domain. A distance function $\delta()$ takes pairs of elements of the domain, quantifying the dissimilarity among them. If $\delta()$ satisfies the following properties: for any elements x, y and z of the data domain, $\delta(x, x) = 0$ and $\delta(x, y) > 0, x \neq y$ (non-negativeness); $\delta(x, y) = \delta(y, x)$ (symmetry); $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$ (triangular inequality), it is said to be a metric. Metrics are fundamental to create access methods that can be used with complex data types, the so-called "Metric Access Methods - (MAM)". MAMs, such as Slim-tree [12], can accelerate similarity queries on complex data by orders of magnitude.

There are basically two types of similarity queries: the Range Query (RQ) and the k-Nearest Neighbor Query (k-NNQ) [8]. The range query algorithm, $Range(o_q, r_q)$, searches the data set recovering every element that is at the distance r_q or closer from the query center o_q. An example of a Range Query on a dataset $\mathbf{S}$ of genomic sequences is: "*choose the polypeptide chains which are different from the chain p by up to 5 codons*". This can be expressed in relational algebra as $\sigma_{(Range(p,5))}\mathbf{S}$. The k-nearest neighbor algorithm, k-$NN(o_q, k)$, searches the data set recovering the k elements nearest to the query center o_q. An example of a k-Nearest Neighbor Query on $\mathbf{S}$ is: "*Choose the 10 polypeptide chains closer to polypeptide chains p*"; which can also be expressed in relational algebra as $\sigma_{(k-NN(p,10))}\mathbf{S}$.

Due to the high computational cost to calculate the distance between a pair of elements in complex domains, similarity queries commonly uses an index structure to accelerate the processing. Index structures prune subtrees using the limiting radius. An algorithm $Range(o_q, r_q)$ executes range queries using r_q as the limiting radius, thus the pruning ability of the $Range$ algorithms using index structures is usually high. However, there is no limiting radius to perform a k-Nearest Neighbor Query.

A k-$NN(o_q, k)$ algorithm starts collecting a list of k ele-

ments in the dataset, ordered by the distance of the element to the query center o_q. A "dynamic radius" keeps track of the largest distance from elements in this list to the query center o_q. The algorithm is executed computing the distance from o_q to all elements in the data set. Whenever a nearer element is found, it is included in the list, removing the object with the largest radius, and reducing the dynamic radius accordingly. The $k\text{-}NN(o_q, k)$ algorithm starts with a dynamic radius larger than the maximum distance between any pair of elements in the dataset, that is, the diameter of the dataset (or simply start with ∞ infinity). Therefore, until at least k elements are found, no pruning can be executed, and afterward the radius is progressively reduced, allowing that many unsuitable elements had been initially treated as candidates. This procedure increases the number of distance calculations required to find the correct answer. If a proper radius can be set to start the $k\text{-}NN()$ algorithm, a significant reduction in the number of distance calculations can be achieved. Therefore, the problem posed is the following: "*How an estimated radius r can be found to execute a RQ that returns the same elements of a k-NNQ?*"

In this work we present a new algorithm to perform k-nearest neighbor queries, called $k\text{-}NNF(o_q, k)$, which takes advantage of the Fractal Dimensionality of a dataset to estimate such a radius. It can decrease the number of distance calculations up to 25%, and speedup in 50% the total time demanded by the query processing.

The remainder of this paper is structured as follows. Section 2 comments on related work to this paper. Section 3 provides a short description of the concept of Correlation Fractal Dimension. Section 4 introduces the equations used to define the estimated radius r and the $k\text{-}NNF()$ algorithm. Section 5 presents the experiments on the metric tree Slim-Tree performed in order to evaluate the proposed method, showing that it significantly improves the performance of *k-NNQ*. Finally, section 6 gives the conclusions of this paper.

2. RELATED WORK

In the last years, algorithms to answer similarity queries has motivated many researches, the majority of them using index structures. A common approach that have been used is called the "branch-and-bound". In this approach, specific properties (heuristics) of the data domain must be considered to prune the branches of the tree. One of the most influential algorithms in this category was proposed by Roussopoulos et al. [10] to find the k-NN queries using R-trees [7] to index points in muldimensional spaces, and until now, the one with the best performance ratio for general k-NN queries. Other algorithms based on this approach are presented in [3], [1] and [4]. Regarding metric spaces, the algorithms to answer similarity queries also follow the branch-and-bound approach, as for example those proposed to work with the M-tree [4] and the Slim-tree [12].

Other approaches were also proposed. One of them uses incremental algorithms to answer similarity queries, like the one proposed by Park and Kim [9], which can partially prune worthless tuples that does not fulfill the remaining non-spatial predicates in a query. An alternative approach proposed, in the literature, by Berchtold et al. [2] indexes an approximation of the Voronoi diagram associated with the dataset.

3. BACKGROUND

Experimental evidences have shown that the distribution of distances between pairs of elements in the majority of real datasets does not follow any of the traditional statistical distributions, such as Gaussian or Poisson. Instead, it has been shown that they present a "fractal behavior". That is, for a usable range of scales, the distribution of distances between elements of a dataset follows power laws [11, 6].

For datasets resembling fractals, it has been shown that, given a set of N objects in a dataset with a distance function $\delta(x, y)$, the average number k of neighbors within a given distance r is proportional to r raised to $\mathcal{D}$. Thus, the pair-count $PC(r)$ of pairs of elements within distance r follows the power law:

$$PC(r) = K_p \cdot r^{\mathcal{D}} \tag{1}$$

where K_p is a proportionality constant, and $\mathcal{D}$ is the correlation fractal dimension of the dataset.

Whenever a dataset presents a metric between pairs of its elements, a graph depicting it can certainly be drawn, even if the dataset are not in a dimensional domain. Plotting this graph in log-log scales, for the majority of real datasets results in an almost straight line for a significant range of distances. This plot in log-log scale of the number of pairs within a radius r for each r is called the "Distance Plot" [11]. The slope of the line in the distance plot is the exponent in equation 1, so it is called the "Distance Exponent". It is interesting to note that $\mathcal{D}$ closely approximates the correlation fractal dimension of a dataset, and therefore its intrinsic dimensionality [13]. Hence, it can be seen as a measurement of how the elements in the dataset are distributed, even if the dataset has no spatial property.

Figure 1 shows the distance plot of a dataset whose elements are the geographical coordinates of streets and roads of the Montgomery County. As can be seen, the plots are linear for the most sought after range of sizes in the queries. We are not interested in radius much smaller or larger than the typical distances involved in the dataset.

Using plots like that, the distance exponent $\mathcal{D}$ of any dataset can be calculated as the slope of the line that best fits the resulting curve in the distance plot. Therefore, considering Figure 1, equation 1 can be expressed as

$$log(PC(r)) = \mathcal{D} \cdot log(r) + K_d, \quad K_d = log(K_p) \tag{2}$$

The distance exponent has many interesting properties, derived from the Correlation Fractal Dimension ones. The main property used in this work is that the correlation fractal dimension $\mathcal{D}$ is invariant to the size of the dataset, provided a reasonable number of elements from a representative sample are used [6]. Therefore, the slopes of the lines corresponding to datasets taken from the same data domain will be the same. This enable one to keep the distance exponent measured for a dataset even after the dataset had been updated with insertions and deletions.

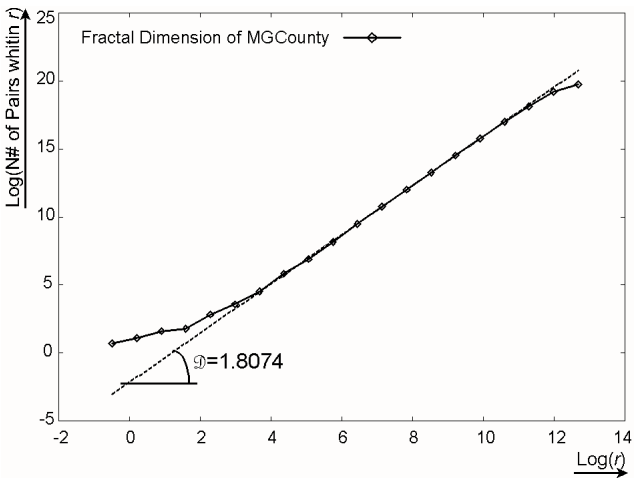

Figure 1: Distance Plot of the MGCounty dataset, showing its Correlation Fractal Dimension $\mathcal{D} \approx 1.81$.

4. USING FRACTALS TO ESTIMATE A SUITABLE RADIUS

In this section we show how to speedup the k-$NN(o_q, k)$ algorithm using the distance exponent of the searched dataset. The idea is to estimate a final radius r for the k-NN query, allowing the majority of elements unlikely to pertain to the answer set be pruned even before k elements have been found. The improved algorithm, that we call k-$NNF(o_q, k)$, performs three steps: first, it must estimate the final radius r for the *k-NNQ*, then it must perform the radius-limited k-$NN()$ algorithm, and finally, if the required number k of objects were not obtained, refine the search procedure.

The second step can be done easily, creating a new algorithm $kAndR(o_q, k, r)$, whose sole difference from the original k-$NN(o_q, k)$ algorithm is the input parameter r, used to initialize the dynamic query radius. This modification produces an algorithm able to answer a composite query, equivalent to $Range(o_q, r)$ and k-$NN(o_q, k)$ queries centered at the same query center o_q. That is, considering the dataset $\mathbf{S}$, we have that

$$\sigma_{(Range(o_q,r_q) \wedge k-NN(o_q,k))}(\mathbf{S}) \Leftrightarrow$$
$$\sigma_{Range(o_q,r_q)}(\mathbf{S}) \cap \sigma_{k-NN(o_q,k)}(\mathbf{S}) \Leftrightarrow$$
$$\sigma_{kAndR(o_q,k,r_q)}(\mathbf{S})$$

For example, it could answer queries like "*Select the 10 nearest restaurants not farther than a mile from here*" ($kAndR$(here, 10, 1 mile)). This is why we called it the *nearest and range* $kAndR()$ algorithm. Note that, the answer can recover less than 10 restaurants if the given range is not sufficiently large.

The first step of the k-$NNF(o_q, k)$ algorithm corresponds to define a method to estimate the final radius of the k-NN query, so the corresponding value r can be used to call the $kAndR(o_q, k, r)$ algorithm. This value can be estimated using the distance plot of a dataset, and an adequate line with slope $\mathcal{D}$ to convert the number k of elements into the corresponding covering radius r.

For this, we use the property of the distance exponent to remain the same when the dataset is updated. However, even knowing its slope, one must define a particular line for a given dataset using a known point of the line. So, when a query must be answered, we propose to use the distance exponent as the slope of the line, the total number of objects N in the dataset, and the diameter R of the dataset at that time. The number N is easily obtained, and the diameter R can be estimated from the indexing structure, as the diameter of the root node of the indexing tree.

Equation 2 uses the number of pairs $PC(r)$ within a distance r, but we are interested in the number of elements k involved, so we must be able to convert numbers of elements into numbers of pairs within a distance. Having a subset of k elements from the dataset, the number of pairs that these elements generate is

$$Pairs(k) = \frac{k(k-1)}{2} \tag{3}$$

because each pair should be counted only once. Thus, given a dataset with cardinality N, the number of pairs separated by distances less than the diameter of the dataset is $PC(R) = Pairs(N) = N(N-1)/2$. Thus, a line specific for the dataset when a query is issued can be found considering the point $< log(R), log(Pairs(N)) >$ in the distance plot of the dataset. Using this line, the number of elements k that form pairs at a distance less or equal r can be estimated using $PC(r) = Pairs(k)$.

Figure 2 illustrates this idea. In this Figure, 'Line 0' is the line used to calculate the intrinsic dimension $\mathcal{D}$ of the data domain of a dataset. This line approximates the average number of points (in log scale) over the full range of distances that occur in the dataset. Notice that real datasets usually have fewer distances with values near the diameter of the dataset, so the counting of the larger distances increases at a slower pace than the counting of medium or small distances. This is the reason of the flattening of typical pair-counting plots at large radius of real datasets, as shown by the Montgomery County dataset in Figures 1 and 2.

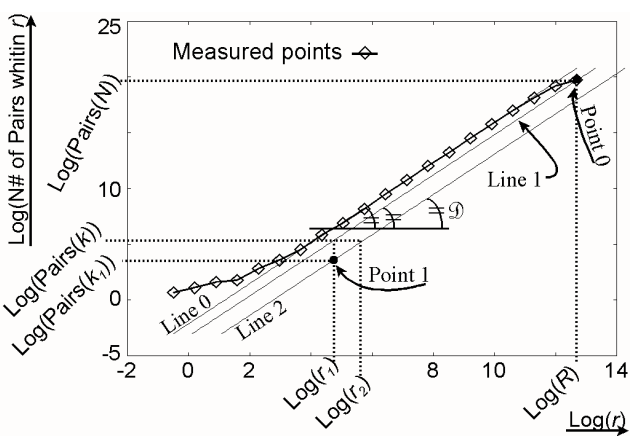

Figure 2: How to use the distance plot of a dataset to estimate the radius r for a *k-NNQ*.

Now consider the line with slope $\mathcal{D}$ passing at the point defined as $< log(R), log(Pairs(N)) >$, identified as 'Point

0' in Figure 2. This line, identified as 'Line 1' in the Figure, represents the relationship between the radii and the number of pairs within each radius. Therefore, it is adequate to be used to estimate the final radius r of a k-NN query. The constant K_d of equation 2 of 'Line 1' can be calculated as:

$$\begin{aligned} K_d &= log(Pairs(N)) - \mathcal{D} \cdot log(R) \\ &= log\left(\frac{N(N-1)}{2}\right) - \mathcal{D} \cdot log(R) \end{aligned} \quad (4)$$

Combining equation 4 with equation 2 we obtain

$$\begin{aligned} log(r) &= \frac{log(PC(r)) - K_d}{\mathcal{D}} \\ &= \frac{log\left(\frac{k(k-1)}{2}\right) - log\left(\frac{N(N-1)}{2}\right) + \mathcal{D} \cdot log(R)}{\mathcal{D}} \Rightarrow \\ r &= R \cdot \exp\left(\frac{log\left(k(k-1)\right) - log\left(N(N-1)\right)}{\mathcal{D}}\right) \end{aligned} \quad (5)$$

Equation 5 can be used to estimate the final radius r_1 of a k-NN query. Obviously r_1 is an approximation, as the exact radius of a query depends on the local density of elements around the query center. Therefore, if the *k-NNQ* is centered where the density of elements is similar to or higher than the average density of the whole dataset, the estimated radius r_1 can be used to call the $kAndR(o_q, k, r)$ algorithm at the same center, answering the *k-NNQ* with a better performance. If more than the required number k of elements are returned, the dynamic radius of the algorithm shrinks, retrieving just those k elements nearest to the query center.

However, if r_1 is under-estimated, the $kAndR(o_q, k, r)$ algorithm will return fewer elements than required (quantity k_1 of elements), and another call to the $kAndR()$ algorithm is required. As calling this algorithm a second time reduces the overall performance gain of this technique, it is worth to augment slightly the estimated r_1 radius to minimize the odds of having to call the $kAndR()$ algorithm more than once. In fact this augment is already considered in the method described above, when we proposed to estimate the size of the dataset looking at the indexing structure: as the covering radius of the nodes of metric trees needs to guarantee that every element stored at each subtree is covered by that radius, they are always equal to or even larger than the minimum required radius.

The third step of the proposed $k\text{-}NNF(o_q, k)$ occurs when r_1 reveals to be under-estimated despite the inflated diameter obtained from the root node of the metric tree, then another search is required with a larger value $r_2 > r_1$. At this time, an incremental search can be performed by another algorithm called $kRingRange(o_q, r_1, r_2, k - k_1)$, which returns the $k-k_1$ elements in the ring between the inner r_1 and outer r_2 radius closest to the query center o_q. The $kRingRange()$ algorithm is basically a $Range()$ algorithm modified to prune also nodes and elements that are inside the ball defined by the inner radius, and with an external dynamic radius that can be reduced whenever the required number of objects were already found nearer to the original outer radius.

The value for r_2 can be calculated using the distance plot again, as follows (see Figure 2). The first calling to the $kAndR()$ algorithm using radius r_1 returned a quantity k_1 of elements that is less than the required k elements asked by the original *k-NNQ*. The number k_1 indicates the local density of the dataset around the query center. Therefore, the point defined by $< log(r_1), log(Pairs(k_1)) >$ defines another line in the distance plot, indicated as 'Line 2' in figure 2, that is useful to estimate the final radius of *k-NNQ* centered at regions whose local density is similar to that where the current query were posed. Therefore, the radius r_2 can be estimated through equation 5 considering the line passing through the point $< log(r_1), log(Pairs(k_1)) >$ as:

$$r = r_1 \cdot \exp\left(\frac{log\left(k(k-1)\right) - log\left(k_1(k_1-1)\right)}{\mathcal{D}}\right) \quad (6)$$

In the unlikely event of a k-NN query where the first call to the $kRingRange()$ algorithm does not retrieve the required number k of elements, the point $< log(r_2), log(Pairs(k_2)) >$, where k_2 is the total number of elements retrieved by the previous callings of the $kAndR()$ and $kRingRange()$ algorithms, it can be used to estimate another radius. This new value is used in another call to the $kRingRange()$ algorithm, repeating this last step until the desired number k of elements are retrieved.

The $k\text{-}NNF()$ algorithm, shown as algorithm 1, uses the $kAndR()$ and $kRingRange()$ algorithms to perform the same duty of a usual $k\text{-}NN()$ algorithm, so it receives the same parameters and produces the same results but with a superior performance (see Section 5).

Algorithm 1 $k\text{-}NNF(o_q, k)$

Input: o_q, k
Output: The *Answer* List of k pairs $< OId, \delta(o_{k_i}, o_q) >$.

```
1: Obtain N as the number of elements in dataset
2: Obtain R as the diameter of the dataset indexed
3: Clear the Answer list
4: Set r1 through Equation 5
5: Execute KAndR(oq, k, r1),
   store the answer in Answer,
   and set k1 to the number of elements retrieved
6: while k1 ≤ k do
7:    Set r2 through Equation 6
8:    Execute kRingRange(oq, r1, r2, k − k1),
      store the answer in Answer,
      and set k1 to the number of elements in Answer
9:    set r1=r2
```

5. EXPERIMENTAL RESULTS

To evaluate the effectiveness of the presented concepts, we worked on a variety of datasets, both synthetic and from the real world. As we cannot discuss all of them here, we selected two real datasets that have representative experimental results to present the efficiency of $k\text{-}NNF()$ algorithm.

The MGC dataset is a dataset with the geographical positions of 27,032 road intersections in Montgomery County, Maryland and has a $\mathcal{D}$ equal to 1.81. The CUR dataset represents the normalized exchange rate of currencies from six countries, using the Canadian Dollar as the reference, obtained daily over a period of 10 years. It have 2,561 values

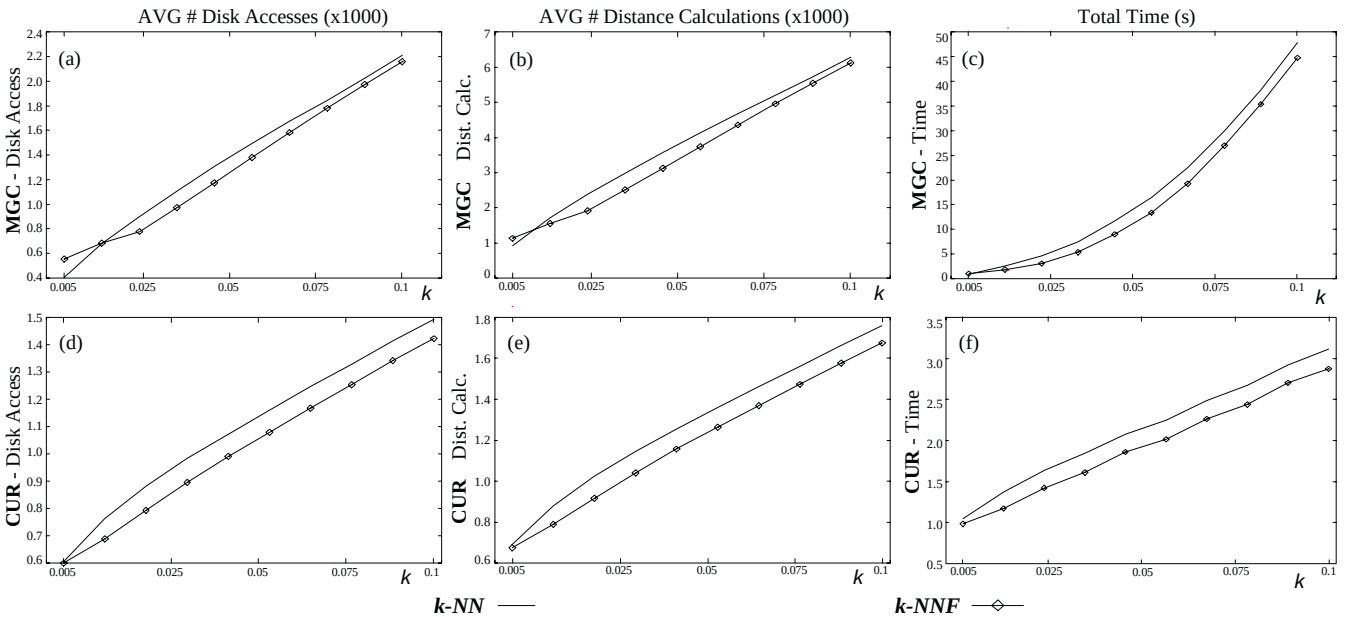

Figure 3: Comparison between k-$NNF()$ and traditional k-$NN()$ algorithm from MGC and CUR datasets, regarding: (a) the average number disk accesses for MGC dataset; (b) the average number distance calculations for MGC dataset; (c) the total time for MCG dataset; (d) the average number disk accesses for CUR dataset; (e) the average number distance calculations for CUR dataset; (f) the total time for CUR dataset.

and presents a $\mathcal{D}$ equal to 2.60. Table 1 summary the real datasets.

Table 1: Summary of the real datasets for experiments

Name	$\mathcal{D}$	#Attrib.	# Objs	Description
MGC	1.81	2	27,032	Road intersections in Montgomery County
CUR	2.60	6	2,561	Normalized exchange rate of currencies

For both datasets we have used the Euclidean metric and a computer with an Intel Pentium-4 1.6GHz processor, with 256 MB of RAM memory. To perform the tests, we computed the average number of distance calculations, the average number of disk accesses and total time (in seconds). Each measured point in a plot corresponds to 500 queries with the same number of neighbors using different query centers. The set of 500 query objects are samples extracted from the respective datasets.

5.1 Performance improvement

In this Section we compare the k-$NNF()$ algorithm and the k-$NN()$ algorithm with a priority queue, as described in [10], the one considered to have the better performance for general k-NN queries. Both algorithms were implemented using the Slim-tree as the indexing method. In this experiment, the number of neighbors in the k-NN queries varies between 0.5% to 10% of the dataset.

Figure 3 shows plots of the two datasets. Figure 3(a), (b) and (c) corresponds respectively to the average number of disk accesses, the average number of distance calculations and the total time, for the MGC dataset. As it can be seen, the k-$NNF()$ algorithm reduces up to 25% the required number of distance calculations, up to 50% the total time, and up to 15% the number of disk accesses as compared with the k-$NN()$ algorithm. Figure 3 (d), (e) and (f) show the same set of experiments for the CUR dataset. With this dataset, the k-$NNF()$ algorithm required 12% less distance calculations, 11% less disk accesses, and was up to 17% faster than the k-$NN()$ algorithm.

It is interesting to note the behavior of the k-$NNF()$ algorithm searching the MGC dataset. For queries asking for a quantity of elements k fewer than 0.5% of the dataset, the k-$NNF()$ algorithm is losing in both number of distance calculations and number of disk accesses, as can be seen in Figure 3 (a) and (b). Comparing these curves with the distance plot of the MGC dataset presented in Figure 2, we can see that the distance plot does not follows the line with the measured $\mathcal{D}$ for this dataset at the distance ranges required to answer k-NN queries with these low number of elements. Therefore, this behavior of the k-$NNF()$ algorithm, at those small distance ranges, indicates that the proposed way to estimate the final range radius of the k-NN queries is useful, whenever the dataset behavior at the required range follows $\mathcal{D}$ at the required radius.

Regarding total time, the k-$NNF()$ algorithm presents major improvements when compared with k-$NN()$ algorithm, as seen in Figure 3(c) and (f). This happens because the k-$NN()$ algorithm uses a priority queue whose management is complex and consumes time, mainly, when the number of neighbors grows. On the other side, the complexity of the proposed k-$NNF()$ algorithm is much lower, reducing the

processing time accordingly.

5.2 Number of kRingRange() calls

An interesting point to check is determining when the *kRingRange*() algorithm needs to be called. To verify this point, we measured the number of times it is called at each set of 500 queries using the same number k of required elements, generating an histogram of number of distance calls for 0 calls of *kRingRange*(), 1 call, 2 calls and so on. It turned out that the *kRingRange*() algorithm was never called more than twice, and even the second call was only performed once in both datasets. Figure 4 shows the plot of those histograms for no calls and for one call for the CUR dataset.

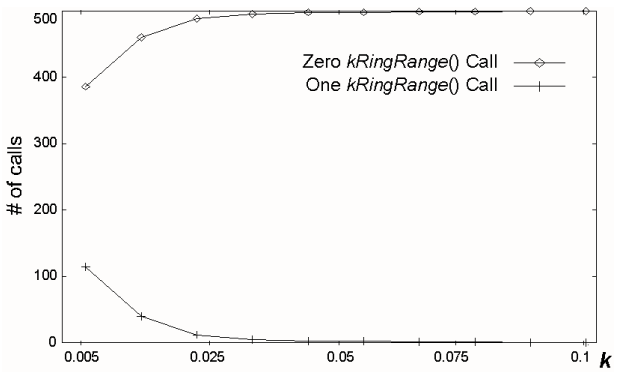

Figure 4: k vs. Number of Calls for the CUR dataset.

6. CONCLUSIONS

In this paper we presented how to explore the distance exponent of a dataset, measured as its correlation fractal dimension, to estimate the final radius of k-nearest neighbor query, as a way to improve its performance. Therefore we proposed a new algorithm, called the k-NNF(), which can be used to execute k-NN queries more efficiently. Our approach can be applied to both spatial and metric datasets. The experiments conducted to evaluate the algorithms show that the k-NNF() algorithm can reduce the time needed to answer a query in up to 50%, requiring up to 25% less distance calculations, and up to 15% less disk accesses.

Besides presenting a real improvement in answering k-NN queries, the proposed algorithm is simple to be implemented, requiring less computational power than the competing algorithms in the literature. Moreover, to the best of the authors' knowledge, this algorithm is the first application of the fractal theory to improve the algorithms to answer queries, bringing the fractal concepts to the inner core of data retrieval algorithms of database management systems.

Acknowledgement

This work has been supported by FAPESP (São Paulo State Research Foundation) under grants number 01/02426-8, 01/11987-3 and 02/07318-1, and CNPq (Brazilian National Council for Supporting Research) under grants 52.1685/98-6, 52.1267/96-0 and 860.068/00-7. The authors are grateful for the insightful contributions provided by Fabio J. T. Chino.

7. REFERENCES

[1] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *Intl. Database Engineering and Applications Symposium (IDEAS)*, pages 44–53, Canada, 2002. IEEE Computer Society.

[2] S. Berchtold, B. Ertl, D. A. Keim, H.-P. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional space. In *Intl. Conference on Data Engineering (ICDE)*, pages 209–218, USA, 1998.

[3] K. L. Cheung and A. W.-C. Fu. Enhanced nearest neighbour search on the R-tree. *ACM SIGMOD Records*, 27(3):16–21, 1998.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Intl. Conference on Very Large Databases (VLDB)*, pages 426–435, Greece, 1997. Morgan Kaufmann Publishers.

[5] C. Faloutsos. Indexing of multimedia data. In *Multimedia Databases in Perspective*, pages 219–245. Springer Verlag, 1997.

[6] C. Faloutsos, B. Seeger, A. J. M. Traina, and C. Traina Jr. Spatial join selectivity using power laws. In *ACM Intl. Conference on Data Management (SIGMOD)*, pages 177–188, USA, 2000.

[7] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *ACM Intl. Conference on Data Management (SIGMOD)*, pages 47–57, USA, 1984.

[8] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Intl. Conference on Very Large Databases (VLDB)*, pages 215–226, India, 1996. Morgan Kaufmann.

[9] D.-J. Park and H.-J. Kim. An enhanced technique for k-nearest neighbor queries with non-spatial selection predicates. *Multimedia Tools and Applications*, 19(1):79–103, 2003.

[10] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *ACM Intl. Conference on Data Management (SIGMOD)*, pages 71–79, USA, 1995.

[11] C. Traina Jr., A. J. M. Traina, and C. Faloutsos. Distance exponent: a new concept for selectivity estimation in metric trees. Research Paper CMU-CS-99-110, Carnegie Mellon University - School of Computer Science, March 1999.

[12] C. Traina Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *Intl. Conference on Extending Database Technology (EDBT)*, volume 1777 of *Lecture Notes in Computer Science*, pages 51–65, Germany, 2000. Springer.

[13] C. Traina Jr., A. J. M. Traina, L. Wu, and C. Faloutsos. Fast feature selection using fractal dimension. In *XV Brazilian Database Symposium*, pages 158–171, Brazil, 2000.