Similarity Search without Tears: the OMNI-Family of All-Purpose Access Methods

Roberto Figueira Santos Filho[1] Agma Traina[1] Caetano Traina Jr.[1] Christos Faloutsos[2]
[1] *Department of Computer Science and Statistics - University of São Paulo at São Carlos - Brazil*
[2] *Department of Computer Science - Carnegie Mellon University - USA*
{*figueira | agma | caetano* }*@icmc.sc.usp.br christos@cs.cmu.edu*

Abstract

Designing a new access method inside a commercial DBMS is cumbersome and expensive. We propose a family of metric access methods that are fast and easy to implement on top of existing access methods, such as sequential scan, R-trees and Slim-trees.

The idea is to elect a set of objects as foci, and gauge all other object with their distances from this set. We show how to define the foci set cardinality, how to choose appropriate foci, and how to perform range and nearest-neighbor queries using them, without false dismissals. The foci increase the pruning of distance calculations during the query processing. Furthermore we index the distances from each object to the foci to reduce even triangular inequality comparisons.

Experiments on real and synthetic datasets show that our methods match or outperform existing methods. They are up to 10 times faster, and perform up to 10 times fewer distance calculations and disk accesses. In addition, it scale up well, exhibiting sub-linear performance with growing database size.

1. Introduction

The growth of multimedia applications in recent years has pushed database management systems to support more diverse and complex data, such as dynamic and static images, time series, fingerprints, protein sequences, etc. Usually, this sort of data is searched by looking for information similar to that which the user has at hand. We assume that the dis-similarity (or distance) function used in comparisons is a metric function (see Section 2). As the similarity evaluation of multimedia objects can be expensive and time consuming, we consider the number of distance calculations as a factor to be reduced.

Our technique choose a number of objects from the dataset as "foci", from which distance calculations to answer queries can be pruned. We show that this technique can be used with other existing indexing structures, improving their performance in terms of distance calculations. We call this "*Omni-concept*", because the foci can both be used as global reference points to any object in the database, and be used to improve any underlaying index structure. Its use with an existing index structure generates a new index structure, leading to a whole new class of index methods, that we call the "*Omni-family*".

This paper makes the following contributions:

1. It provides a complete and practical technique to reduce the number of distance calculations needed to answer similarity queries, using a set of a few selected objects from the dataset. When the cost of distance calculation is high, this technique gives good performance even if the query retrieves much more than ten percent of the dataset.
2. It explains how to define an adequate number of objects to be used as Omni-foci, with the best tradeoff between increasing memory requirements and decreasing distance calculations.
3. It depicts an inexpensive algorithm to select adequate Omni-foci.
4. It shows the results of applying the Omni-concept with sequential scan and R-trees, achieving up to ten times fewer distance calculations, disk accesses and overall time.

The remainder of the paper is structured as follows. The next section summarizes some related concepts. Section 3 surveys the main current index structures and shows how they use triangular inequality to increase performance. There is also a description of the datasets used in our experiments. Section 4 introduces the Omni-concept and presents both a way to choose the number of foci and an algorithm to choose them. Section 5 describes how to exploit the Omni-concept with existing access methods, namely the sequential scan, B^+-trees and R-trees. Section 6 presents experiments showing the behavior of two members of the Omni-family and a comparison with existing access methods. Section 7 gives the conclusions of this paper.

2. Background

In this section we present the basic definitions which delimit the scope of this work.

[1] On leave at Carnegie Mellon University. This research has been funded by FAPESP (São Paulo State Foundation for Research Support - Brazil, under Grants 98/05556-5, 98/05559-7 and 98/15731-9).

[2] This material is based upon work supported by the National Science Foundation under Grants No. IRI-9625428, DMS-9873442, IIS-9817496, and IIS-9910606. Additional funding was provided by donations from NEC and Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

2.1. Metric spaces and similarity queries

Definition 1 (metric distance function): Given a set of objects $S = \{s_1, s_2, ..., s_n\}$ of a domain S, a function $d()$ that has the following properties:

1. Symmetry: $d(s_1,s_2) = d(s_2, s_1)$
2. Non-negativity: $0<d(s_1,s_2) < \infty$, $s_1 \neq s_2$ *and* $d(s_1,s_1)=0$
3. Triangle inequality: $d(s_1,s_3) \leq d(s_1,s_2) + d(s_2,s_3)$

is called a metric distance function. A metric space is a pair $M=<S, d()>$ which follows the aforementioned properties.

Spatial datasets following an L_p-metric distance function (such as Euclidean distances) are special cases of metric spaces. We considered two classes of similarity queries: range queries and nearest neighbors queries, which are defined as follows.

Definition 2 (*Range query*): Given a query object $s_q \in S$, and a maximum search distance r_q, the answer is the subset of S such that $Rquery(s_q, r_q)=\{ s_i \mid s_i \in S: d(s_i, s_q) \leq r_q\}$.

An example of a range query on a word dataset with the L_{Edit} distance function is: *"Find the words that are within distance 3 from the word 'Germany' "*.

Definition 3 (*Nearest Neighbor query*): Given a query object $s_q \in S$, the nearest neighbor is the unitary subset of S such that $NNquery(s_q)=\{s_n \in S \mid \forall\, s_i \in S: d(s_n, s_q) \leq d(s_i, s_q)\}$.

A common variation is the k-nearest neighbor query. It finds the $k \leq N$ closest objects to s_q in the dataset. An example of a k-nearest neighbor query with k=5 is: *"Select the 5 words nearest to 'Germany' "*.

A property useful to decrease the number of distance calculations is the triangular inequality. Generically, a Metric Access Method, MAM, divides the dataset into regions or nodes and chooses strategic objects to be representatives for every region. The representatives, objects in their region and their distances to the representatives are stored in the nodes. The nodes are organized hierarchically forming a tree. When a query is issued, the query object is first compared with the representatives of the nodes, and then triangular inequality is used to prune distance calculations between the query object and objects of the node. In this paper we show that the triangular inequality property can be used to prune distance calculations with fixed foci (global pruning), obtaining better results than using representatives of the nodes (local pruning).

2.2. Intrinsic dimensionality

The behavior of a dataset on queries can be estimated by its dimensionality. Several works have been presented on this subject, dealing with the so called 'dimensionality curse'. Some of them assume that the embedding dimensionality of the dataset defines this behavior. If that were true, it would be better to answer queries on datasets with high embedding dimensionality performing a simple sequential scan [22] [5]. However, the embedding dimensionality does not indicate the correct behavior of the dataset every time, because a dataset can inhabit just a "small" portion of the embedding space. Moreover, metric datasets do not have an embedded dimensionality at all. A better way to quantify the behavior of a dataset is to consider its intrinsic dimensionality.

It was shown in [14] that the concept of intrinsic (or fractal) dimensionality gives better precision in selectivity estimation for nearest neighbor queries than the embedding dimension. An equivalent development is made for range queries in [16].

To understand the role of intrinsic dimensionality in queries, consider for example a dataset which represents points along a line. A line has intrinsic dimensionality one; it does not matter if the line is embedded in a 2-, 3- or any n-dimensional space. Thus, estimating the selectivity of queries on a line in a 3-dimensional space using the embedding dimension gives a figure much larger than the actual objects in the dataset.

We use the correlation fractal dimension D_2 as an approximation of the intrinsic dimension of a dataset [3] [16]. Algorithms to estimate the correlation fractal dimension of vector datasets in linear time is depicted in [12] and [19]. For metric datasets, a quadratic algorithm needs to be used [15]. Table 1 summarizes the symbols used in this paper.

Table 1 - Summary of symbols and definitions.

Symbols	Definitions
S	domain of objects
S	set of objects in domain S
N	number of objects in the dataset S
D_2	correlation fractal dimension of the dataset S
s_q	a query object (or query center)
s_i, s_j	objects of S
r_q	radius of a range query
k	the number of neighbors in a NN query
$\mathcal{F}$	the Omni-foci base
f_k	a focus of S in $\mathcal{F}$
l	number of foci in $\mathcal{F}$
$d()$	distance function
$df_k(s_i)$	distance from an object s_i to focus f_k

3. Related work

The design of efficient access methods has long been an objective of many researchers. Multidimensional access methods have been proposed in the literature, and an excellent survey is given in [13]. However, most of these methods only work for vector data. Regarding image datasets, some works have used selected objects as reference points to prune distance calculations [4] and to organize index structures [17].

The seminal work of Burkhard and Keller [9] provides different interesting techniques for partitioning a metric data set where the recursive process is materialized as a tree. The first technique partitions a dataset by choosing a representative from

the set and grouping the elements with respect to their distance from it. The second technique partitions the original set into a fixed number of subsets and chooses a representative from each of the subsets. The representative and the maximum distance from the representative to a point of the corresponding subset are also maintained to support nearest neighbor queries. The metric tree of Uhlmann [20] and the vantage-point tree (vp-tree) of Yanilos [23] are somehow similar to the first technique of [9] as they partition the elements into two groups according to a representative, called a vantage point. In [23] the vp-tree has been generalized to a multi-way tree. In order to reduce the number of distance calculations, Baeza-Yates et al [2] suggested using the same vantage point in all nodes that belong to the same level. Then, a binary tree degenerates into a simple list of vantage points. Another method [20] is the generalized hyper-plane tree (gh-tree), which partitions the data set into two by picking two points as representatives and assigning the remaining to the closest representative. Bozkaya and Ozsoyoglu [7] [6] proposed an extension of the vp-tree called multi-vantage-point tree (mvp-tree) which chooses in a clever way m vantage points for a node which has a fanout of m^2. The Geometric Near Access Tree (GNAT) of Brin [8] can be viewed as a refinement of the second technique presented in [9]. It stores the distances between pairs of representatives in addition to the representative and the maximum distance. These distances can be used to prune the search space using triangle inequality. An excellent survey of metric trees can be found in [10]

Some methods for metric datasets presented above are static, in the sense that the data structure is built once, and new insertions are not supported. The M-tree of Ciaccia, Patella and Zezulla [11] was the first MAM to overcome this deficiency. The M-tree is a height-balanced tree where the data elements are stored in the leaves. The Slim-tree [18] is another dynamic MAM, which reduces the amount of overlap between tree nodes. With a process to 'slim-down' a tree, it leads to a smaller number of disk accesses to answer similarity queries.

4. The Omni-concept and definitions

This section presents the proposed technique, which uses a set of global foci to prune distance calculations. This concept can be used either alone, empowering sequential scanning, or with existing MAMs, thus generating the Omni-family of indexing methods. The elements of the Omni-concept are presented as follows.

Definition 4 - Omni-foci base ($\mathcal{F}$): Given a metric space $M=<S, d()>$, an Omni-foci base is a set $\mathcal{F}=\{f_1, f_2, ..., f_l \mid f_k \in S, f_k \neq f_j, l \leq N\}$ where each f_k is a focus (or focal point) of S, and l is the number of focus in the Omni-foci base.

Definition 5 - Omni-coordinates: Given the object $s_i \in S$ and the Omni-foci base $\mathcal{F}$, the Omni-coordinates C_i of an object s_i is the set of distances from s_i to each focus in $\mathcal{F}$, that is $C_i=\{ < f_k, d(f_k, s_i) >, \forall f_k \in \mathcal{F}\}$. To distinguish the distance $d(f_k, s_i)$ as a coordinate, we use the notation $df_k(s_i)= d(f_k, s_i)$. The cardinality of C_i is the same number l of the foci in the Omni-foci base $\mathcal{F}$.

When a new object is inserted in the dataset, its Omni-coordinates are evaluated and stored. During a search, the Omni-coordinates are used to prune distance calculations through the triangle inequality property.

The cost of using foci comes from two sources: the time to calculate the Omni-coordinates for each object in S; and the memory space in a data structure to store the Omni-coordinates. We consider that both the Omni-coordinates and the datasets are stored in disk. Disk space is cheap, so memory space is not an issue. However, increased disk usage requires more disk accesses, slowing down the query answering process.

Assuming that few foci are needed, both costs can be kept low. We claim the extra distance calculations needed between the objects and the foci will pay off during the query process. The practical evidence shown in Section 6 corroborates this.

To analyze memory usage, we need to consider the demand from the Omni-coordinates relative to the memory required to store the objects themselves. Complex and large objects, such as images and audio, need huge memory, so the space needed to keep a few extra numbers that store distances is relatively insignificant. However, the increase in memory required to store the Omni-coordinates with smaller objects is at a first glance more significant. We argue the corresponding increase in disk accesses is not relevant, because pruning distance calculations compensates reducing disk accesses. In this way, global foci reduces drastically the number of distance computations, without any other significant cost. It is also worth noting that the implementation cost of the Omni-concepts is low, due to the simplicity of the structure and operations required.

4.1. The Omni-foci base

The issues of choosing the foci base $\mathcal{F}$ and its cardinality l are tightly related. There is a tradeoff between the number of foci and the space and time spent to process them. Therefore, it is important to maximize the gain with the minimum number of foci. To discover such a number we need the following concept.

Definition 6 - Minimum Bounding Omni Region (*mbOr*): given the Omni-foci base $\mathcal{F}=\{f_1, f_2, ..., f_l\}$ and a collection of objects $A=\{x_1, x_2, ..., x_n\} \subset S$, the *mbOr* of A is defined as the intersection of the metric intervals $R_A= \mid_1^l I_i$, where $I_i= [\min(d(x_j, f_i)), \max(d(x_j, f_i))]$, $1 \leq i \leq l$, $1 \leq j \leq n$.

As we can see from Figure 1, each focus defines a metric sub-space "ring". An *mbOr* is the sub-set of S contained in the hyper-volume that the Omni-coordinates identifies as including the answer of a query, that is, the region where objects cannot be pruned by the foci. Notice that an *mbOr* always includes all objects of the response set (there are no false dismissals) although it can include objects that are not in the response set, (there are false alarms) so a final refinement step must be applied. Only in this refinement step the real distance of the candidate object to the query object is calculated.

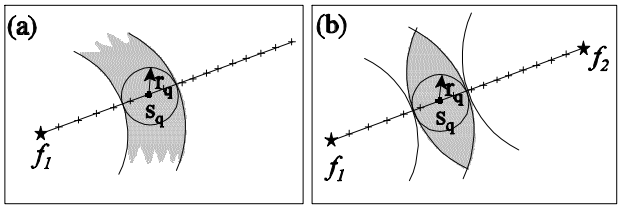

Figure 1 - Active space defined by: (a) one focus; (b) two foci.

A focus is adequate if it reduces the *mbOr* noticeably. Considering spatial datasets, it is inherent that the intrinsic dimension and the distance function define a limit for the proper number of foci. Figure 2(a) illustrates this idea showing a line (a one-dimensional data set) embedded in a two-dimensional space considering the L_2 distance function, where a range query is pruned by one focus (Figure 2(b)) and two foci (Figure 2(c)). Considering the L_p-metric family of distance functions, twice the intrinsic dimension would be enough to maximize the benefit. Selecting more foci than twice the intrinsic dimension of the dataset, regardless of the embedding dimension, leads to no or little reduction in the *mbOr*.

Two foci lead to the maximum reduction of the *mbOr* if they are "orthogonal", far apart, and with the origin coinciding with the query center. However, as the foci are pre-defined, we cannot assume orthogonality. Also, in metric datasets it is not always possible to create "artificial" objects to act as far apart foci, so the maximum reduction cannot always be achieved. In this way one more focus can be used to "distribute" the overhead of having foci not ideally placed for each query. Using this extra focus, the new average maximum reduction occurs if they are far apart and equally distant from each other. These requirements are easier to achieve in a metric space. Therefore, a good number for the cardinality l of $\mathcal{F}$ would be between the next integer that contains the intrinsic dimension $\lceil D_2 \rceil+1$ and $2*\lceil D_2 \rceil+1$.

Notice that although we used spatial datasets to achieve these results, excluding the distance function considerations, the same deductive process can be repeated using metric datasets, so they hold for any metric dataset. Therefore, we propose the following practical guideline to choose the Omni-foci base:

> Choose $\lceil D_2 \rceil+1$ equally spaced foci, which are the most far apart possible from each other, surrounding the other objects of the dataset.

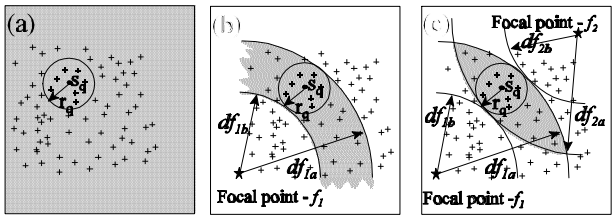

Figure 2- A range query with center s_q and radius r_q on a dataset. The shadow illustrates the objects included in the *mbOr*. (a) No focus, so all the dataset is the candidate response set. (b) One focus f_1. (c) Two foci f_1 and f_2.

The sole information about the objects needed to obtain the foci is the distance between them, which is known in metric spaces. Two foci are chosen for datasets with intrinsic dimension one or less, corresponding to the most far apart pair of objects in the dataset. Datasets with $1<\lceil D_2 \rceil \leq 2$ will lead to three foci defining a equilateral triangle; datasets with $2<\lceil D_2 \rceil \leq 3$ will lead to four foci defining a tetrahedron, and so on.

4.2. How to choose the foci: the HF-algorithm

Here we describe the algorithmic implementation of the guidelines proposed. The algorithm must choose only objects inside the dataset to be foci because sometimes it is impossible to synthesize an object of the dataset, e.g. a new fingerprint. This leads to foci not ideally located, neither surrounding the dataset nor positioning the vertices on the tetrahedron. Second, the algorithm to find the "best" foci has complexity $O(N!/(N-l)!)$, where N is the number of objects in the dataset, and l is the number of foci.

We show a practical algorithm requiring $O(N)$ distance calculations, which try to find the foci near the hull of the dataset, the HF-algorithm. This algorithm starts searching a pair of objects far apart. To do this, it randomly chooses an object s_1, find the farthest object f_1 from this object, and set it as the first focus. Following, it finds the farthest object f_2 from f_1, setting it as the second focus, and storing the distance between them as *edge*.

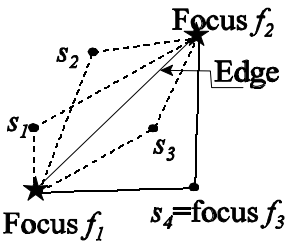

Figure 3 - Given foci f_1 and f_2, select the third one between s_1, s_2, s_3 and s_4. The best is s_4.

The next focus will be the object that has the most similar distances to the previously chosen foci (see Figure 3). To do this, using the foci already chosen, for each object s_i not yet picked as a focus, calculate the following error:

$$error_i = \sum_{k}^{k \text{ is focus}} \left| edge - d(f_k, s_i) \right|$$

Now, select as the next focus the object s_i that produced the minimum $error_i$. This last step is repeated until the required

HF-Algorithm: finding the foci of a dataset S
Input: the dataset S and the number of foci l
Output: foci set $\mathcal{F}$
Begin
1. Randomly choose an object $s_i \in S$.
2. Find the farthest object f_1 from s_i. Insert f_1 in $\mathcal{F}$.
3. Find the farthest object f_2 from f_1. Insert f_2 in $\mathcal{F}$.
4. Set $edge=d(f_1, f_2)$, used to calculate $error_i$.
5. While there are foci to be found, do:
6. For each $s_i \in S$, $s_i \notin \mathcal{F}$: calculate $error_i$
7. Select $s_i \in S$ such that $s_i \notin \mathcal{F}$ and $error_i$ is minimal.
8. Insert s_i in $\mathcal{F}$.
End.

Figure 4 -The HF-algorithm to find the foci set.

number of foci is selected.

The HF-algorithm requires $l*N$ distance calculations, and is shown in Figure 4. Notice that those distance calculations required in steps 3 and 6 of Figure 4 correspond to distances from each object to a focus, so those distances would be calculated anyway, as they are part of the Omni-coordinates of each object. Therefore, the extra distance calculations performed by this algorithm are only those occurring in step 2, which accounts for just N extra distance calculations. Hence, the HF-algorithm also prepares the Omni-coordinates of a dataset (in steps 3 and 6).

The HF-algorithm does not require the full dataset to choose Omni-coordinates. Experiments have shown that good foci can be obtained within a well-distributed sampling. Objects can be inserted or deleted without changing the Omni-foci base. Also, the Omni-foci base can be changed without affecting the underlaying index method, at the cost of recalculating the Omni-coordinates.

5. Members of the Omni-family

The Omni-concept is effective for pruning distance calculations in both nearest neighbor and range queries. Here we present how to use the Omni-concepts together with sequential scan, B-trees and R-trees.

5.1. The Omni-sequential

To perform a sequential scan the whole dataset is read. The algorithms to answer similarity queries are simple:

Range queries: Read throughout the dataset comparing the query object s_q with every object s_i in the dataset. Whenever the resulting distance is less or equal the query radius r_q, s_i is placed in the answer set.

K-Nearest neighbor queries: Assign the first k objects of the dataset to a working response set, sorted by their distances to the query object s_q. The largest distance is used as the current query radius r_c. Then, each remaining objects s_i is read and compared with s_q. Whenever the distance is less than the current query radius r_c, the farthest object in the working response set is replaced by this s_i, and r_c is updated accordingly.

The processing of both queries ends when the whole dataset has been read. In both cases the number of distance calculations is the number N of objects in the dataset.

Applying the Omni-concept to sequential scans requires a second file to store the Omni-coordinates, composed by N registers, one for each Omni-coordinate.

We named the sequential scan modified by the Omni-concept, Omni-sequential. In this new access method, a range query with query radius r_q is executed as in a conventional sequential scan, with the following two modifications. First, calculate the distance $df_k(s_q)$ from the query object s_q to each focus f_k, creating the Omni-coordinates $df_k(s_q)$ for s_q. Second, precede the distance calculation of the query object s_q with each object s_i of the dataset by the following computation:

for each $f_k \in \mathcal{F}$ do: if $| df_k(s_i) - df_k(s_q) | > r_q$ then skip the distance calculation.

The modifications in the k-nearest neighbors algorithm are similar. For both kinds of queries, an object from the dataset is retrieved from the file and compared to the query object only when no focus can prune it. This enables a large reduction of the number of distance calculations, as shown in Section 6.

5.2. The OmniB-tree

Objects in a metric space do not embody the notion of order, so index structures such as B$^+$-trees cannot index them. However, the distances $df_k(s_i)$ from one focus k to each object s_i can be sorted. By this way the Omni-coordinates file can be replaced by a set of B$^+$-trees, one for each focus. The algorithms to answer similarity queries using B$^+$-trees follows.

Applying the Omni-concepts with B$^+$-trees originates a new index structure we call the OmniB$^+$-tree. This member of the Omni-family is, as the others, able to index metric datasets. This leads to an effective way to support metric datasets using the existing access methods in commercial database management systems, with a lower development cost [1].

OmniB$^+$-trees store the Omni-coordinates in l B$^+$-trees, one for each focus f_k. Whenever a query centered in the object s_q with query radius r_q arrives, the subsets $I_k \subset S$ (as in definition 6) are retrieved from the corresponding B$^+$-tree and used to generate the *mbOr*. Each I_k is retrieved using the range of objects in the B$^+$-tree that is between $r_{min}=df_k(s_q) - r_q$ and $r_{max}=df_k(s_q) + r_q$. The answer set is obtained calculating the distance from the query object s_q to each object in the intersection. Nearest neighbor queries do not have a fixed radius, but an estimated radius can be adopted using the selectivity estimation formulas provided by the fractal theory [3] [14] [12], and successively corrected if the requested number of objects is not retrieved.

5.3. The OmniR-tree

Here we show how to store the Omni-coordinates using one R-tree, through the metric index structure we call OmniR-tree. The benefits of using an OmniR-tree over R-trees are two-fold: first, it permits to use R-trees even for non-vector datasets; second, for vector-datasets it permits reducing the dimensionality of the *MBR*s of the R-tree to the number of foci adopted. The number of foci is related to the intrinsic dimension and not to the usually larger embedding dimension used by a plain R-tree. Therefore, the dimensionality of the dataset seen as Omni-coordinates will be lower, improving the performance of the R-tree.

The algorithms to do insertion, node partitioning, range queries, etc. in OmniR-trees are the same used in plain R-trees. However, the algorithm to perform k-nearest neighbors queries needs to be replaced. There are two approaches. The first one is to estimate a final radius, as described for the OmniB-tree. This approach can be used with any implementation of R-tree, so it is not tailored to the use of R-trees with the Omni-concepts. The second approach uses the nearest neighbors algorithm used

in metric trees. In this case, a deep-search is first performed to find *k*-candidates. The algorithm continues reducing the radius whenever the furthest neighbor is replaced, until every entry that overlaps the radius in the query has been tested. We used this approach to evaluate the datasets presented in the next section. These approaches guarantees no false dismissals, an essential property that the usual nearest neighbors algorithm of the R-tree cannot ensure when used with Omni-concepts.

OmniR-trees also use two data structures to store the database: an R-tree to store the Omni-coordinates, and a paged direct access file to store the objects in the dataset (storing multiple objects per page). Whenever a leaf node in the R-tree is retrieved, and Omni-coordinates stored in this node qualify objects, the actual distance must be calculated. So, each qualifying object must be retrieved from the sequential file. To reduce disk accesses in the sequential file, it is worth storing the objects following the order they occur in the leaf nodes of the R-tree. This re-ordering does not need to be done after every update in the R-tree, but it is valuable after major updates.

6. Experiments

To evaluate the effectiveness of the Omni-concept, we worked on a variety of datasets, both real and synthetic. Due to space restrictions, we show here only the results for the following datasets.

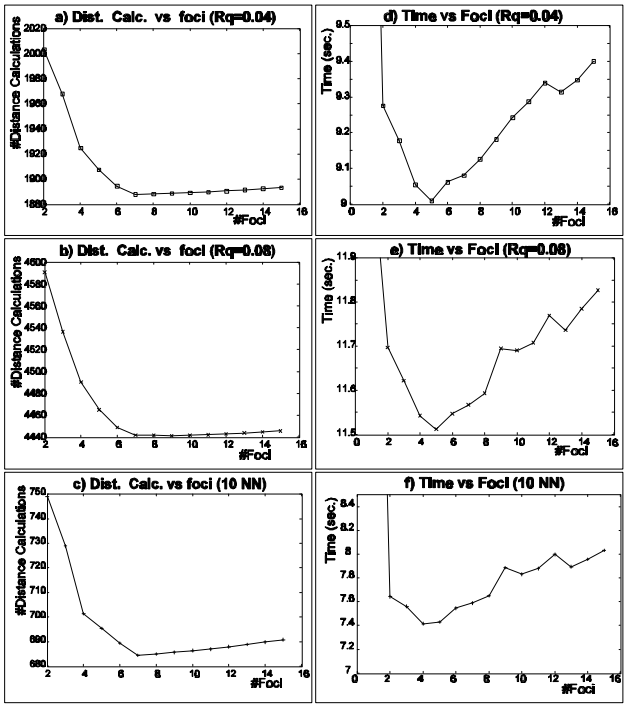

Figure 5 - Results for the number of distance calculations (left column) and for time (right column) querying the Eigenfaces dataset with increasing number of foci. (a,d) Range query for radius r_q=0.04 (7% of the dataset qualify). (b, e) Range query for radius r_q=0.08 (31% of the dataset qualify. (c, f) 10 Nearest neighbors.

EnglishWords - A set of 25,143 objects from the English language, using the L_{Edit} distance function.

Eigenfaces - A set of 11,900 face vectors from the Informedia project [21] at Carnegie Mellon University. Each face was processed with the eigenfaces method, resulting in a 16-dimensional vector. It employs the Euclidean distance function.

Sinthetic30D - A synthetic dataset with 250,000 objects and thirty attributes, with an intrinsic dimension of 9.45. We use it with the Euclidean distance function.

6.1. The number of foci to be used

Here we show that the number of foci is related to the intrinsic dimensionality of the dataset. Due to space limitations, we show the results for only one dataset as the other datasets presents similar behavior. Figure 5 shows the number of distance calculations and time required to answer range (for radii 0.04 and 0.08) and *k*-nearest neighbor queries (for *k*=10), as a function of the number of foci employed. The graphs show the Omni-sequential indexing the Eigenfaces dataset, which exhibits an intrinsic dimension of 4.67. In the first column of Figure 5 we can see that increasing the number of foci up to seven always reduces the number of distance calculations, but after this, this number starts to increase again. The second column of Figure 5 shows that time reduces progressively when using up to five foci, then starts to climb again. The cost of one Euclidean distance calculation in the Eigenfaces dataset, is small. With costlier distance functions, the response time would more closely follow that of the number of distance computations. However, these plots corroborates the intrinsic dimensionality of the dataset as a good reference for the number of foci.

6.2. Comparing Omni-family with other structures

We compared two index methods of the Omni-family, the Omni-sequential and the OmniR-tree, with sequential scan, R-tree and Slim-tree. The datasets used were the metric dataset EnglishWords and the spatial dataset Eigenfaces. Figure 6 shows the results for time (in seconds), the number of distance calculations and the number of disk accesses. The results are the average number of distance calculations and of disk accesses for 500 range queries and 500 nearest neighbors queries.

The results for range queries in the Eigenfaces dataset (Figures 6(a, b, c)) are presented in log scale to visualize a wider range of radii. Time shown are total to answer 500 queries (Figures 6(d and j)). The Omni-family members were tested using six foci for both datasets, as the intrinsic dimensionality of the EnglishWords is 4.75 and of the Eigenfaces is 4.67. It is worth noting that the number of distance calculations for the Omni-sequential and for the OmniR-tree are the same. This is due the same foci were used for both structures.

Time measurements (Figures 6(a, d, g and j)) and distance calculations (Figures 6(b, e, h and k)) show that the Omni variation of the index method always wins over the flat ones, achieving an speed up of more than ten times for small radii (where the

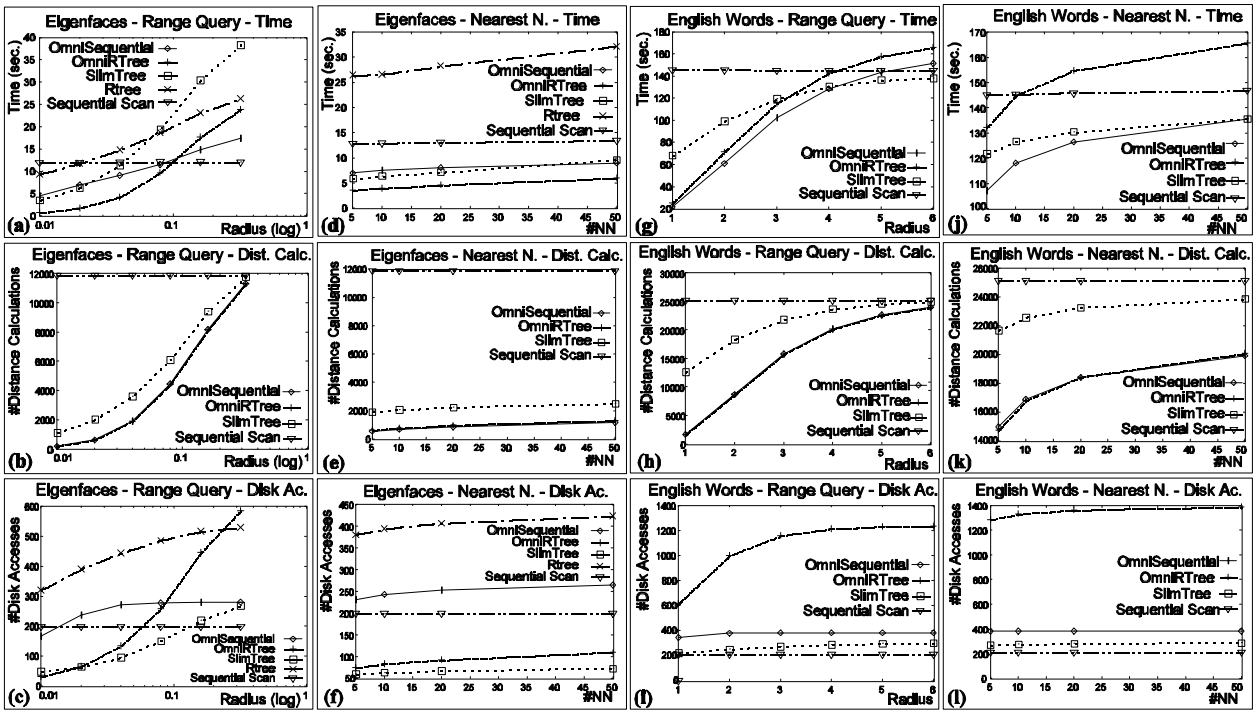

Figure 6 - Comparing time, number of distance calculations and number of disk accesses of Omni-sequential, OmniR-tree, sequential scan, R-tree and Slim-tree access methods, for range queries and nearest neighbors queries over the Eigenfaces and English Words datasets.

majority of queries occurs) on range queries. We cannot show distance calculation for R-trees, as their operation relies on comparisons of bounding rectangles. The reduction in the number of distance calculations is up to four times less in nearest neighbors queries and more than ten times less for small radii on range queries. Both comparisons were done with the Slim-tree, an allegedly good metric access method. There is a tradeoff between decreasing distance calculations and increasing disk accesses. However, this is worth paying as the overall time required always reduces.

6.3. Scalability

The Omni-sequential and OmniR-tree are scalable regarding the size of the dataset. Figure 7 shows the behavior of these indexing methods for time (a, d), number of distance calculations (b, e), and disk accesses (c, f). The results are the average for 500 range queries with r_q=0.5 and 500 k-nearest neighbors with k=10. It is worth note that the Omni-foci base was chosen using a random sample of ten percentile of the database. As we can see in Figure 7, the behavior of both index structures is linear for range queries. Interestingly, our methods present sub-linear behavior for nearest neighbors, making these methods an excellent choice to index very large datasets.

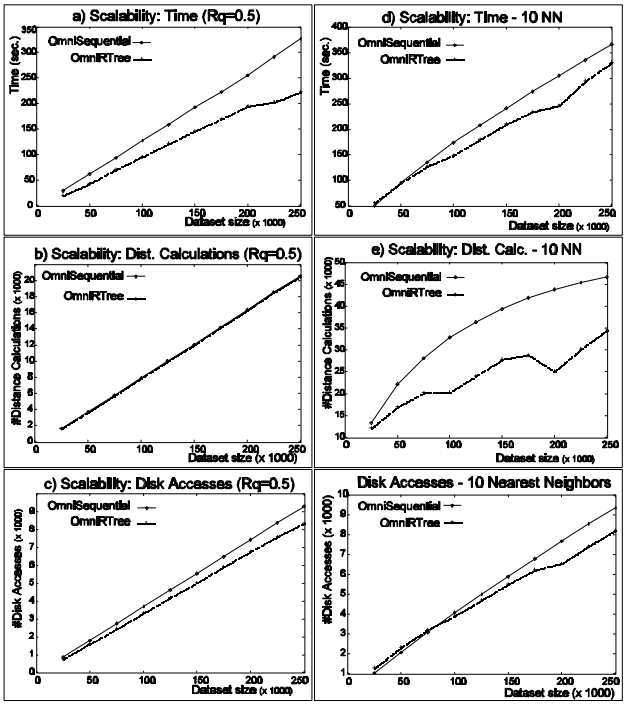

Figure 8 - Scalability of Omni-sequential and OmniR-trees for range (left column) and nearest neighbors (right column) queries, in Syntethic30 dataset. (a, d) Time. (b, e) Number of distance calculations. (c, f) Number of disk accesses.

7. Conclusion

We presented the Omni-concept, which intends to speedup range and nearest neighbors queries by decreasing the number of distance calculations. We show how this new concept can be

used to index metric and vector datasets with any indexing method. For example, we showed how to index words with R-trees! Due to its adaptability, Omni-concept can help to improve the performance of any method already implemented in applications such as database management systems. And due to its simplicity this can be achieved without tears.

Our experiments showed speedups of up to ten times in query answers and equivalent results in reducing distance computations. Therefore, it broadens the scope of the underlining index method already used in the commercial DBMS to also support metric datasets. There are other important contributions as well: the practical guideline to define the number of foci, as a tradeoff between space requirements and distance computations; and the HF-algorithm to choose the foci, which is linear on the size of the dataset.

The Omni-family members tested in this paper are scalable with respect to dataset size, presenting a sub-linear behavior regarding time. We believe that the Omni-concept will greatly contribute to improve the sup3port to metric data provided by database management systems.

Acknowledgments: The authors would like to thank Panos Chrysanthis and the DB group at CMU for their insightful suggestions about the material of this paper.

References

[1] M. Annamalai, R. Chopra, and S. DeFazio, " Indexing Images in Oracle8i," *Proc. ACM Int'l Conference on Data Management (SIGMOD)*, Dallas, TX, May 14-19, 2000, pp. 539-547.

[2] R. A. Baeza-Yates, W. Cunto, U. Manber, and S. Wu, "Proximity Matching Using Fixed-Queries Trees," *Proc. 5th Annual Symp. on Combinatorial Pattern Matching (CPM)*, Asilomar, CA, June 5-8, 1994, pp. 198-212.

[3] A. Belussi and C. Faloutsos, "Estimating the Selectivity of Spatial Queries Using the Correlation Fractal Dimension," *Proc. Intl. Conf. on Very Large Databases (VLDB)*, 1995, pp. 299-310.

[4] A. Berman and L. G. Shapiro, "Selecting Good Keys for Triangle-Inequality-Based Pruning Algorithms," *Proc. Intrl. Workshop on Content-Based Access of Image and Video Databases (CAIVD '98)*, Bombay, India, January 3, 1998, pp. 12-19.

[5] K. Beyer, J. Godstein, R. Ramakrishnan, and U. Shaft, "When is "Nearest Neighbor" Meaningful?," *Proc. 7th International Conference (ICDT '99)*, Jerusalem, Israel, January 10-12, 1999, pp. 217-235.

[6] T. Bozkaya and Z. M. Özsoyoglu, "Distance-Based Indexing for High-Dimensional Metric Spaces," *Proc. ACM Int'l Conference on Data Management (SIGMOD)*, Tucson, AZ, 1997, pp. 357-368.

[7] T. Bozkaya and Z. M. Özsoyoglu, "Indexing Large Metric Spaces for Similarity Search Queries," *ACM Transactions on Database Systems (TODS)*, Vol. 24, No. 3, September 1999, pp. 361-404.

[8] S. Brin, "Near neighbor search in large metric spaces," *Proc. Intl. Conf. on Very Large Databases (VLDB)*, Zurich, Switzerland, 1995, pp. 574-584.

[9] W. A. Burkhard and R. M. Keller, "Some Approaches to Best-Match File Searching," *Communications of the ACM*, Vol. 16, No. 4, 1973, pp. 230-236.

[10] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín, "Searching in Metric Spaces," in *to appear in the ACM Computing Surveys*: ACM Press, 2001.

[11] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," *Proc. Intl. Conf. on Very Large Databases (VLDB)*, Athens, Greece, September 1997, pp. 426-435.

[12] C. Faloutsos, B. Seeger, A. J. M. Traina, and C. Traina, "Spatial Join Selectivity Using Power Laws," *Proc. ACM Int'l Conference on Data Management (SIGMOD)*, Dallas, TX, May 14-19, 2000, pp. 177-188.

[13] V. Gaede and O. Gunther, "Multidimensional Access Methods," *ACM Computing Surveys*, Vol. 30, No. 2, 1998, pp. 170-231.

[14] B.-U. Pagel, F. Korn, and C. Faloutsos, "Deflating the Dimensionality Curse Using Multiple Fractal Dimensions," *Proc. 16th Intl. Conf. on Data Engineering (ICDE2000)*, San Diego, CA, February 28 - March 3, 2000, pp. 589-598.

[15] M. Schroeder, *Fractals, Chaos, Power Laws*, 6 ed. New York: W.H. Freeman and Company, 1991.

[16] C. Traina, A. J. M. Traina, and C. Faloutsos, "Distance Exponent: a New Concept for Selectivity Estimation in Metric Trees," *Proc. Intl. Conf. on Data Engineering (ICDE)*, San Diego - CA, February 28 - March 3, 2000, pp. 195.

[17] C. Traina, A. J. M. Traina, R. R. d. Santos, and E. Y. Senzako, "Support to Content-Based Image Query in Object-Oriented Databases," *Proc. ACM Symposium on Applied Computing*, Atlanta, Georgia, February 27 to March 1, 1998, pp. 241-247.

[18] C. Traina, A. J. M. Traina, B. Seeger, and C. Faloutsos, "Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes," *Proc. Intl. Conf. on Extending Database Technology*, Konstanz, Germany, March 27-31, 2000, pp. 51-65.

[19] C. Traina, A. J. M. Traina, L. Wu, and C. Faloutsos, "Fast feature selection using fractal dimension," *Proc. XV Brazilian Database Symposium*, João Pessoa - PA - Brazil, october 2-4, 2000, pp. to be published.

[20] J. K. Uhlmann, "Satisfying General Proximity/Similarity Queries with Metric Trees," *Information Processing Letter*, Vol. 40, No. 4, November 25, 1991, pp. 175-179.

[21] H. D. Wactlar, T. Kanade, M. A. Smith, and S. M. Stevens, "Intelligent Access to Digital Video: Informedia Project," *IEEE Computer*, Vol. 29, No. 3, 1996, pp. 46-52.

[22] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. 24rd Intl. Conf. on Very Large Data Bases (VLDB)*, New York City, August 24-27, 1998, pp. 194-205.

[23] P. N. Yianilos, "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces," *Proc. Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms - SODA*, Austin, TX, January 25-27, 1993, pp. 311-321.