The VLDB Journal (2007) 16: 483–505
DOI 10.1007/s00778-005-0178-0

REGULAR PAPER

Caetano Traina Jr · Roberto F. Santos Filho · Agma J. M. Traina · Marcos R. Vieira · Christos Faloutsos

The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient

Received: 2 November 2004 / Accepted: 1 July 2005 / Published online: 29 June 2006

Abstract Similarity search operations require executing expensive algorithms, and although broadly useful in many new applications, they rely on specific structures not yet supported by commercial DBMS. In this paper we discuss the new Omni-technique, which allows to build a variety of dynamic Metric Access Methods based on a number of selected objects from the dataset, used as global reference objects. We call them as the Omni-family of metric access methods. This technique enables building similarity search operations on top of existing structures, significantly improving their performance, regarding the number of disk access and distance calculations. Additionally, our methods scale up well, exhibiting sub-linear behavior with growing database size.

Keywords Similarity search · Metric access methods · Index structures · Multimedia databases

1 Introduction

The evolution of the Computer Science field brought us new challenges and the demand to deal with diverse human knowledge, considerably expanding the gamut of data types that must be supported by the computational applications, as is the case of the Database Management Systems (DBMS) for multimedia data. Protein sequences, DNA strings, images, audio, video and time series are examples of new and complex data types that must be stored and retrieved. Complex data need to be searched and compared through their content, demanding more elaborate processing in the retrieval operations. Answering queries based on data that are "alike" but not "equal" is known as *similarity search*. Performing similarity searches on complex data is a cumbersome process, having its own challenges and problems.

A requirement to query sets of complex data types is a function to evaluate the degree of similarity (or the opposite, the dissimilarity or distance) between two objects of the dataset. The challenge is to simulate the process performed by the human specialist when comparing such data. The comparison usually requires expensive algorithms, taking a lot of computing time to find the answer. Another problem is how to store large sets of complex data. Such data tend to be structurally complex and each element composed of thousands of bytes, requiring efficient strategies for secondary storage management. Therefore, querying large datasets requires using efficient indexing structures to reduce the number of comparison operations.

Many approaches to accelerate similarity queries in large datasets have been proposed, the majority of them creating new access methods. In this paper we present a new approach to improve existing access methods, the Omni-technique, leading to the development of a family of new indexing structures. It aims at improving the performance of similarity queries by reducing both the required number of comparison operations and the amount of data accessed during the query execution. We show that our technique is faster than the existing ones, while it guarantees the accuracy of the query answer with no false dismissals and no false alarms, as false alarms are cleaned in a post-processing phase.

1.1 Paper scope, contributions and organization

As the evaluation of similarity between two complex objects is usually expensive and time consuming, the reduction of the number of comparisons in an important issue. Therefore, we regard the cost of distance calculations as the most expensive step during the process of answering similarity queries. Also, while the information technology community urges for techniques to efficiently manage complex data types, inserting a new access method in commercial

C. Traina Jr (✉) · R. F. S. Filho · A. J. M. Traina · M. R. Vieira
Department of Computer Science, Drop and Statistics,
University of São Paulo at São Carlos,
Avenida Trabalhador Saocarlense, 400, São Carlos, SP, Brazil
E-mail: {caetano, figueira, agma, mrvieira}@icmc.usp.br

C. Faloutsos
Department of Computer Science, Carnegie Mellon University,
5000 Forbes Ave., Pittsburgh, PA, USA
E-mail: christos@cs.cmu.edu

database management systems (DBMS) is an expensive task. In commercial DBMS, problems such as concurrent access, robustness and intensive insertions and deletions of objects in a highly dynamic environment, among others, must also be carefully treated.

To target this problem, we propose a family of methods lying on the metric space model that can be implemented on top of existing access methods, such as B-Trees and R-Trees, or even on top of sequential scan. The proposed technique uses a number of selected objects from the dataset as global reference points. These objects, called *Omni-foci*, are chosen at the beginning of the indexing process and are used to improve the underlying access method, decreasing the amount of distance calculations needed to answer similarity queries. We call this new approach the "Omni-technique", because it can be applied on the majority of existing access methods.[1] Coupling the Omni-technique with another existing method generates a new one, leading to a whole new class of access methods that we call the "Omni-family".

This paper presents the following contributions:

1. It provides a technique to minimize the number of distance calculations required to answer similarity queries, using the Omni-foci from the dataset. Experimental evidences show that this technique gives good performance even if the query retrieves significantly more than 10% 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 the (secondary) memory demanded and the number of distance calculations required;
3. It depicts an inexpensive algorithm to choose adequate Omni-foci;
4. It shows how to combine the Omni-technique with other existing access methods. Three members of the Omni-family are presented: the Omni-Sequential, OmniR-Tree and OmniB-Forest, including improved algorithms for the two most required kinds of similarity queries: the range and the k-nearest neighbor queries. Algorithms for post-optimization of the data structure (after many insertions) are also discussed;
5. Finally, it presents experimental results comparing the performance of the proposed Omni-family members and the sequential scan, the Slim-Tree, M-Tree, VP-Tree and the R-Tree.

The remainder of this paper is organized as follows. Section 2 provides the background information about metric distance functions, similarity queries and intrinsic dimensionality of datasets. Section 3 surveys the state-of-the-art works about Metric Access Methods (MAMs). Section 4 introduces the Omni-technique, including file organization and query algorithms. Section 5 shows how to combine the Omni-technique with three access methods: sequential scan, B^+-Trees and R-Trees. Section 6 presents experiments showing the behavior of these Omni-family members and a comparison with existing access methods. Finally, Sect. 8 gives the conclusions of this paper.

[1] A preliminary version of this paper was presented at IEEE-ICDE' 2001 [35].

Table 1 Summary of symbols and definitions

Symbol	Definition
$AS[]$	Answer set of a similarity query
$C(s_i)$	Omni-coordinates of object s_i
$d()$	Distance function
$df_g(s_i)$	Distance from an object s_i to focus f_g
D_2	Correlation fractal dimension of the dataset S
E	Embedded dimension of the dataset S
f_g	The gth focus of base $\mathcal{F}$
$\mathcal{F}$	The Omni-foci base
h	Number of foci in $\mathcal{F}$
$IOid$	Object Id used internally by an Omni-member
k	The number of neighbors in a NN query
nb_f	Farthest neighbor in a k-nearest neighbor list
N	Number of objects in the dataset S
r_q	Radius of a range query
s_q	a Query object (or query center)
s_i, s_j	Objects of S
S	Set of objects in domain $\mathbb{S}$
$\mathbb{S}$	Domain of objects

2 Background

This section presents the basic definitions and fundamental concepts supporting this work. Table 1 summarizes the symbols used through this paper.

A core problem related to retrieving complex data comparing their content is the definition of a similarity (or dissimilarity) function that ought to match the notion of similarity of the human specialist in the data domain. A distance function should return small values for object pairs that a human would consider close (or similar), and larger values as more dissimilar the objects are. At least two different approaches related to the data domain have been modeled and studied in the literature, the vector space model and the metric space model:

1. *Vector Space Model* The complex objects are described by their main features, which are extracted by well-tailored algorithms designed by specialists from the data domain. Such information is placed in the so-called *feature vectors*, which are e-dimensional arrays, where e is the number of features extracted from the objects. Each feature vector can be seen as a point pertaining to an e-dimensional space. The dissimilarity between two objects can be measured, for example, by any Minkowski distance function [49], such as Euclidean (L_2), Manhattan (L_1), or Infinity (L_∞). Color histograms are common examples of feature vectors extracted from images, where the dimensionality of the space is the number of colors used in the image quantization process.
2. *Metric Space Model* For some domains of objects, extracting features does not lead to a fixed number of characteristics. This happens, for example, with fingerprints, because everyone has a particular number of features,

such as deltas, endings and so on [38]. Therefore, it is not possible to map one object into a point in an e-dimensional vector space. This restriction often occurs when the notion of similarity is complex and domain dependent [48]. However, it may still be possible to evaluate the dissimilarity between objects in these domains by defining a metric distance function (Sect. 2 will discuss the basic definitions of metric spaces). In this model, similarity search operations rely only on the degree of dissimilarity given by a pair-wise comparison between objects. The L_{Edit} distance [30] employed to compare words and DNA chains and the Metric-Histogram distance [40] employed to compare compressed histograms are examples of metric distance functions.

It is worthwhile to highlight that the aforementioned Minkowski distance functions employed to compare multidimensional data are special cases of the metric space model. Thus, we shall focus on the metric space model because it is less restrictive and, provided with adequate distance functions, includes the vector space.

The set of access methods can also be classified into two main classes: methods to index data types under the vector space model, known as Spatial Access Methods (SAMs), such as the R-Tree [23] (and its variations R$^+$-Tree [37] and R*-Tree [5]), TV-Tree [31] and SR-Tree [27] (see [21] for a survey on SAMs); and methods to index data types under the metric space model, known as Metric Access Methods (MAMs), such as the MVP [9], M-Tree [16] and Slim-Tree [43, 41] among others. As similarity relations only holds in domains that posses a distance function, MAMs are the best-suited structures to handle similarity queries.

2.1 Metric spaces and similarity queries

Similarity queries requires a dissimilarity function, also called a distance function, to generate the metric space. Formally, a metric space is defined as follows.

Definition 1 (*Metric space*) A metric space is a pair $\mathbb{M} = \langle \mathbb{S}, d() \rangle$, where $\mathbb{S}$ denotes the universe of valid elements and $d()$ is the function $d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^+$ that expresses a "distance" between elements of $\mathbb{S}$. Hence the smaller the distance the more similar or closer are the elements [15]. Distance functions must satisfy the following properties:

1. *Symmetry*: $d(s_x, s_y) = d(s_y, s_x)$;
2. *Non negativity*: $0 < d(s_x, s_y) < \infty$ if $s_x \neq s_y$ and $d(s_x, s_x) = 0$; and
3. *Triangular inequality*: $d(s_x, s_y) \leq d(s_x, s_z) + d(s_z, s_y)$, where $s_x, s_y, s_z \in \mathbb{S}$.

A dataset S is said to be in a metric space if $S \in \mathbb{S}$. Similarity queries are the most common ones for multimedia and complex datasets. We consider two classes of similarity queries: range queries and nearest-neighbor queries, as follows.

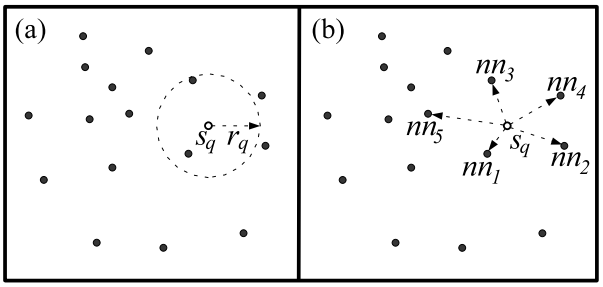

Fig. 1 Examples of similarity queries on $\mathbb{R}^2$. (**a**) range query and (**b**) 5-nearest neighbor query

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

Given a metric space composed of the dataset of English words and the L_{Edit} distance function, an example of a range query is Rq('America', 3), that is: "Find the words that are within distance 3 from the word 'America' ". Figure 1a graphically shows an example of a range query in a dataset in the $\mathbb{R}^2$ domain, where the objects inside the dashed circle qualify.

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

A common variation is the k-nearest neighbor query, where the objective is to find the k closest objects to s_q in the dataset ($0 < k \leq |S|$, where $|S|$ represents the cardinality of S). An example of a k-nearest neighbor query (k-NNq) with $k = 5$ could be: "Select the 5 nearest words to the word 'America'. " Figure 1b shows an example of a 5-nearest neighbor query in a dataset in the $\mathbb{R}^2$ domain.

The triangular inequality property of metric spaces is very useful to decrease the number of distance calculations required to answer queries in such data domains. In general, a metric access method divides the dataset into regions or nodes and chooses strategic objects to be representatives of each region. The representatives, the objects in their region and the distances from the objects to the representatives are stored in the nodes, which are organized hierarchically as a tree. When a query is issued, the query object and the representatives are first compared, and then the triangular inequality property is used to prune distance calculations between the query object and objects of the node. This reduces the number of distance calculations and the total query processing time. One of the main contributions of this paper is showing that the triangular inequality property can be used more extensively to prune distance calculations with fixed foci (global pruning), obtaining better results than using only the single node representative (local pruning).

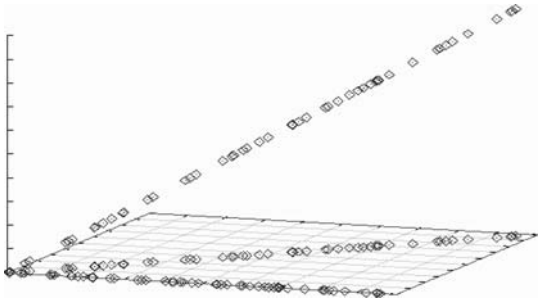

Fig. 2 Dataset of points distributed within a line embedded in one, two and three dimensions

2.2 Intrinsic dimensionality

The behavior of queries posed on a dataset can be estimated employing the dataset dimensionality. Several works have been presented on this subject, dealing with the so-called 'dimensionality curse' [1]. Some of them assume that the embedding dimensionality of the dataset defines its behavior [47], and for datasets with a high embedding dimensionality it would be better to answer queries performing a simple sequential scan [8, 47]. However, the embedding dimensionality does not always indicate the real behavior of queries on the dataset, because the objects 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 [17, 28]. In [34], the authors presented the concept of intrinsic (or fractal) dimensionality, showing that it effectively gives better precision in selectivity estimation for nearest-neighbor queries than the embedding dimension of the dataset. There is an equivalent study for range queries in [39, 42].

To better understand the concept of intrinsic dimensionality, consider for example a dataset composed of points randomly placed along a line. A line in any embedded space has intrinsic dimensionality one, i.e., it does not matter if the line is embedded in a 2- or 3- or e-dimensional space (see Fig. 2). Thus, selectivity estimations using the embedding dimension of a line in a 3-dimensional space tend to overestimate the answer.

In this paper we use the correlation fractal dimension D_2 as an approximation of the intrinsic dimension of the dataset [6, 42]. For vector datasets, an algorithm to estimate the correlation fractal dimension in linear time is depicted in [18, 44]. For metric nonvectorial datasets a quadratic algorithm is used [13, 36].

3 Related work

Many access methods well suited to the vector-space model have been proposed in the literature, and an excellent survey is given in [21]. As we are interested in the more general case provided by the metric-space model, in this section we will concentrate in the Metric Access Methods (MAM) [15, 24].

Indexing the distances between objects was first discussed in the work of Burkhard and Keller [12], where different techniques for partitioning a metric dataset were proposed and 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 the representatives, clustering those with similar distances. The second technique partitions the dataset into a fixed number of subsets such that, for each subset, there is a representative and a maximum radius, and every remaining object is allocated to one of the subsets. Finally, the third technique partitions the dataset into *cliques* of graphs where two nodes (database objects), such as s_x and s_y, are connected only when $d(s_x, s_y) \leq r$, and every element in the dataset is assigned to at least one *clique*. After the division of the dataset, one element of each *clique* is picked as the representative.

Examples exploiting the first technique of [12] are the Metric Tree of Uhlmann [45] and the Vantage-Point-Tree (VP-Tree) of Yanilos [50]. The VP-Tree has also been generalized to a multi-way tree. Later, in [51], in an effort to avoid backtracking during search, Yianilos proposed to exclude from the tree those elements that are near to the threshold and insert them into a bucket. Once completion of the tree, the bucket is organized in the same way, and after some iterations, the full process results in a forest of trees. This method is called Excluded Middle VP-Forest. Gennaro et al. [22] extended this method by providing a hashing function to organize the forest into disk pages, creating the Similarity Hashing technique. In order to reduce the number of distance calculations, Baeza-Yates et al. [4] suggested using the same *vantage point* in all nodes that belong to the same level, creating the Fixed-Queries-Tree, where a binary tree degenerates into a simple list of *vantage points*.

Another example of the first technique is the Generalized-Hyperplane-Tree (GH-Tree) [45], where the dataset is partitioned into two by picking two points as representatives and the remaining are assigned to the closest one. Bozkaya and Ozsoyoglu [9, 10] 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 [11] is a refinement of the second technique presented in [12]. In addition to the representative and the maximum distance, it stores the distances between pairs of representatives, which are used to prune the search space using the triangle inequality property. Despite defined in the context of the vector space model, the *iDistance* method presented in [52] can be adjusted to deal with metric data as well. In this method, a *partition id* is associated to each subset and its representative and a key is created for each object of a subset by a function that combines the *partition id* and the distance between the object and the partition's representative. A B^+-Tree is used to index the keys and the respective objects. Unfortunately,

the heuristic to define the representatives works only with vectors and the nearest-neighbor algorithm is restricted to find only one neighbor. AB$^+$-Tree is also employed in [29] to index the digital codes of hyper-dimensional objects. In [26] the authors propose a method to deal with the problem of the dimensionality curse finding clusters of objects projected by a technique based on adaptive multi-level Mahalanobis distance. The inconvenience of this technique is the lossy characteristic of the indexing obtained.

Another approach is to use a fixed number of k pivots (or representatives) to map the original metric space to a k-dimensional vector space with the L_∞ metric, as presented in [7, 19, 32, 33] (see [15, 24] for a more complete survey). The main problem with this approach is that, despite the mapping guaranteeing that no false dismissals will occur, there is no guarantee that false alarms will not happen. Moreover, except for [33], the nearest-neighbor algorithms presented are restricted to find only one neighbor.

The partition processes employed on the previously discussed methods require the entire dataset available at the construction time, because the definition of the clusters during the partitioning process requires calculating the median of the distances between the objects. Moreover, object deletion usually involves costly reorganization of the structures.

A variation of the fixed pivots approach is the Spaghettis-method [14], where the distances to each pivot are sorted in vectors linked as an inverted list. During range queries, a candidate set is created by intersecting the results given by the search over each vector, comparing the distance from the query object and the respective pivots, and pruning the candidates with the triangular inequality. The candidate set is then reprocessed with the original distance function, discarding the false alarms. However, the nearest-neighbor algorithm is restricted to find only one neighbor and the data structure is not optimized for secondary storage management

The M-Tree of Ciaccia et al. [16] was the first dynamic MAM proposed in the literature. The M-Tree is height-balanced and the data elements are stored in the leaves. Another dynamic MAM is the Slim–Tree [41, 43], which quantifies and reduces the amount of overlap between nodes of the tree. With its process of 'slimming down' the tree, the Slim-Tree leads to a small number of disk accesses in order to answer similarity queries.

The structures of the M-Tree and the Slim-Tree are similar and can be briefly described as follows: the dataset is divided into regions or nodes (disk pages) and an object of the node is chosen as the node's representative. The nodes are organized hierarchically as a tree, and each one is composed of the representative, the objects and their distances to the representative. When a query is issued, the query object is first compared to the representatives of the nodes. Then the triangular inequality property is used to prune distance calculations between the query object and objects of the node. This reduces the number of distance calculations and, consequentially, reduces the total time to answer the query.

The Dynamic VP-Tree of Fu et al. [20] extends the algorithm for nearest-neighbor searching of the VP-Tree to allow retrieving more than just one neighbor. Additionally, algorithms for individual insertion and deletion are proposed, but the algorithms are too complex and may result in a global reorganization of the entire structure. An efficient tree, regarding performance during queries, is only achieved through the bulk-loading operation.

The MB$^+$-Tree [25] can be seen as a variation of the first technique of [12] that tries to implement dynamic updating operations. The objects are managed by a B$^+$-Tree structure, where the leaf nodes are buckets of objects that lie in a partition of the dataset. Whenever a node split occurs, a slicing value is defined, which depends on the representative of the partition, and the objects are distributed among the nodes. Then, the *partition id* and the slicing value are combined to generate a binary key that is used to index the objects. The dynamism is possible due to an additional (main memory and unbalanced) structure used to manage the blocks, storing the representative of each partition and the respective slicing value. As the partitioning strategy may lead to poor node occupancy and the nearest-neighbor algorithm is too expensive, deletions are not possible or require expensive reorganization of the structure. Also, the scalability is restricted by the number of bits composing the key and by the auxiliary structure, and, as with the Dynamic VP-Tree, an efficient tree is only possible through a bulk-loading operation.

4 Proposed idea: the Omni-technique

Typically, metric access methods using only one representative per node can be dynamic, as the Slim-Tree and the M-Tree, whereas trees using more than one representative per node are static, as the GH-Tree and the MVP-Tree. The Omni-technique embodies a different approach, where global (*Omni*) representatives are used to prune distance calculations, obtaining better results than using local representatives in the nodes, whereas maintaining the access method dynamic. A set of few global objects is selected from the dataset and used as representatives for the whole set, instead of using different objects per tree node or per tree level. This approach leads to a global data pruning, because all objects are pruned by the same representatives. The Omni-technique calls each global representative an Omni *focus* and the set of foci as the Omni foci-base.

The main idea is to elect a set of h objects to be foci and represent each object in the dataset by its pre-computed distances to these foci. The objects are stored in a Random Access File (RAF), while the sets of distances are accessed by an access method or by sequential scanning. Note that each object in the dataset will be seen through their set of distances to the foci. The main components of the Omni-technique are presented as follows.

Definition 4 (Omni-foci base $\mathcal{F}$) Given a metric space $\mathbb{M} = \langle \mathbb{S}, d() \rangle$, the Omni-foci base is a set $\mathcal{F} = \{f_1, f_2, \ldots,$

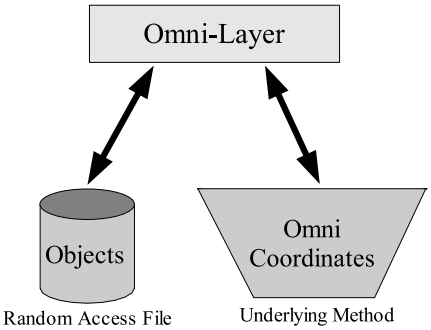

Fig. 3 Example of use of the Omni-technique. The Omni-coordinates are managed by an access method (e.g., sequential scan)

$f_h | f_g, f_j \in \mathbb{S}, f_g \neq f_j, \forall g \neq j\}$, where each f_g is a focus (or focal point) of $\mathbb{S}$, and h is the number of foci in the Omni-foci base.

Definition 5 (Omni-coordinates) Given the object $s_i \in S$, where $S \subseteq \mathbb{S}$, and the Omni-foci base $\mathcal{F}$, the Omni-coordinates $C(s_i)$ of an object s_i is the set of distances from s_i to each focus in $\mathcal{F}$, that is $C(s_i) = \{\langle f_g, d(f_g, s_i)\rangle, g = 1, 2, \ldots, h\}$, where h is the number of focus in the Omni-foci base. For simplicity we will denote $d(f_g, s_i)$ as $df_g(s_i)$ from now on.

The general architecture for the Omni-technique implementation is presented in Fig. 3. The Omni-layer is the interface of the method. It holds the foci-base and is responsible to calculate the Omni-coordinates of every object. When a new object s_i is inserted, its Omni-coordinates $C(s_i)$ are calculated and the object is stored in the RAF. The RAF generates a sequence number that marks where the object is stored. This number, called $IOid(s_i)$, is used as an internal object identifier to access the object during queries. The $IOid(s_i)$ and the respective Omni-coordinates are stored together in the indexing method.

The gain proportioned by the Omni approach derives from how the objects are represented in the underlying method and by the concise data description given by the Omni-coordinates. Usually, the structure of the object entries used in the internal nodes of SAMs and MAMs is quite similar to the structure of the corresponding entries in the leaf nodes. As a consequence, the cardinality of both internal and leaf nodes are nearly the same and the overhead generated by the internal nodes is proportional to the size of the objects. Thus, sizable objects such as large feature vectors from images imply lower node cardinality, and consequently more disk access. Assuming that few foci are needed, this overhead is reduced because the underlying method is built in accordance to the number of foci in use. However, to reach an effective gain, the number of foci and their selection from the dataset must be carefully defined. The following sections consider these issues and show that the gain is quite relevant. The measurements presented in Sect. 6 corroborate this affirmation. It is also worth noting that the implementation cost of the Omni-technique is low, due to the simplicity of the structure and operations required.

A range query $Rq(s_q, r_q)$, with the center object s_q and the query radius r_q, is usually performed in two steps: *filtering* and *refinement*. This also happens when using the Omni approach. In the *filtering* step, the Omni-coordinates $C(s_q)$ are calculated. Then, a range query with center $C(s_q)$ and radius r_q is issued over the index structure that holds the Omni-coordinates, resulting in a list of candidates. The triangular inequality property guarantees that false dismissals will not occur (see proof in Sect. 4.2). However, the absence of false alarms is not guaranteed, requiring a *refinement* step, which corresponds to use the $IOid$ of the candidates to retrieve the original objects from the RAF to compare them to the query object itself. Finally, those objects within distance r_q to s_q will compose the response set. To explain how the list of candidates is obtained, the following definition is needed:

Definition 6 (Minimum-bounding-Omni-region $mbOr$) For a given Omni-foci base $\mathcal{F} = \{f_1, f_2, \ldots, f_h\}, \mathcal{F} \subset \mathbb{S}$, the Minimum-bounding-Omni-region $mbOr_{\mathcal{F}}(s_q, r_q)$ of a query with center $s_q \in \mathbb{S}$ and radius r_q, a subset of objects of S is defined such that:

$$mbOr_{\mathcal{F}}(s_q, r_q) = \bigcap_{g=1}^{h} I_g \tag{1}$$

where I_g is the subset of objects $s_i \in S$ such that its Omni coordinates corresponding to the focus g is in the closed interval $[I_g^{\inf}, I_g^{\sup}]$, where $I_g^{\inf} = df_g(s_q) - r_q$ if $df_g(s_q) \geq r_q$ or $I_g^{\inf} = 0$ otherwise, and $I_g^{\sup} = df_g(s_q) + r_q$. That is:

$$I_g = \left\{ s_i | I_g^{\inf} \leq df_g(s_i) \leq I_g^{\sup}, \forall s_i \in S \right\} \tag{2}$$

As it can be seen in Fig. 4, each focus generates a hyper-ring in the metric space that encompasses the query. An $mbOr$ corresponds to the intersection of such hyper-rings generated by all foci, which is the gray area shown in Fig. 4c. Notice that an $mbOr$ always includes all objects of the response set (there are no false dismissals), although it can also include other objects (false alarms) requiring the *refinement* step. The real distance between each candidate and the query object is calculated only in the refinement step, when the majority of the objects in the dataset were already discarded.

Nearest-neighbor queries are a little trickier than range queries. A general solution is to estimate the radius that encompasses the desired number of neighbors using the selectivity estimation formulas provided by the use of fractal theory [6, 34, 18]. This number can be corrected by successive iterations if the requested number of objects is not retrieved. However, this approach can be improved if it is possible to change the code of the access method that stores the Omni-coordinates. In Sect. 5 we present new similarity search algorithms for the three Omni members: Omni-Sequential, OmniB-Forest and OmniR-Tree.

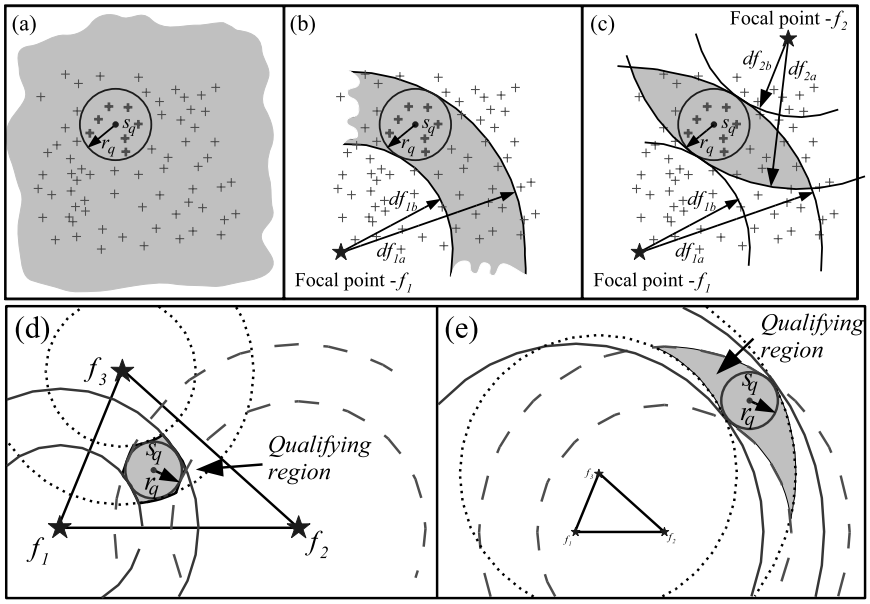

Fig. 4 A range query with center s_q and radius r_q on a 2D dataset. The shadow illustrates the $mbOr$ region, containing the qualifying objects given by the *filtering* step. (**a**) No focus, so all the dataset is the candidate response set. (**b**) One focus f_1. (**c**) Two foci f_1 and f_2. (**d**) The three foci f_1, f_2 and f_3 placed near the border of the dataset: the $mbOr$ region closely circumscribes the query region. (**e**) The three foci f_1, f_2 and f_3 placed near to each other and not enclosing the dataset, leading to a larger $mbOr$ that reduces pruning opportunities

4.1 The Omni-foci base

The issues of choosing the foci base $\mathcal{F}$ and its cardinality h are tightly related. There is a tradeoff between the number of foci, the improvement in pruning distance calculations during queries and the space and time spent to process them. Therefore, it is important to maximize the gain achieved with the minimum number of foci.

We claim that a focus is adequate for a query if it reduces the $mbOr$ noticeably. Regarding spatial datasets, it is intuitive to conjecture that the intrinsic dimension defines a lower bound for the proper number of foci. Figure 5 illustrates this idea through a set of objects distributed along a line – a one-dimensional dataset – embedded in a two-dimensional space using the L_2 distance function. If a range query must be answered and no focus is available, then all objects must be compared. If only one focus is placed farther from the border of the dataset, as illustrated in Fig. 5a, it can prune many distance comparisons. In this figure, the gray area corresponds to the region where the objects must be compared. However, if the focus is placed near or at the border of the dataset, the occurrence of false positives when considering only this focus is smaller, as shown in Fig. 5b. Notice that selecting more foci than at most twice the intrinsic dimension of the dataset, regardless of the embedding dimension leads to little or no reduction in the $mbOr$ (the region of the dataset where are the candidates to answer the query). Figure 5c shows the $mbOr$ defined by two foci, that is twice the intrinsic dimension of this line dataset. As it can be seen, the resulting $mbOr$ in this example does not allow pruning more objects than using just one focus.

The positioning of the foci also plays an important role on the size of the $mbOr$. Figure 4d shows the $mbOr$ resulting on the qualifying region of having the foci on the border of the dataset. It can be seen that the qualifying region tightly encloses the query region. Thus, the number of objects that must be compared (during the refinement step) to get the query answer is small. On the other side, when positioning the foci near to each other and not at the border of the dataset, the ability of them on pruning the objects that would not be part of the query answer decreases. This can

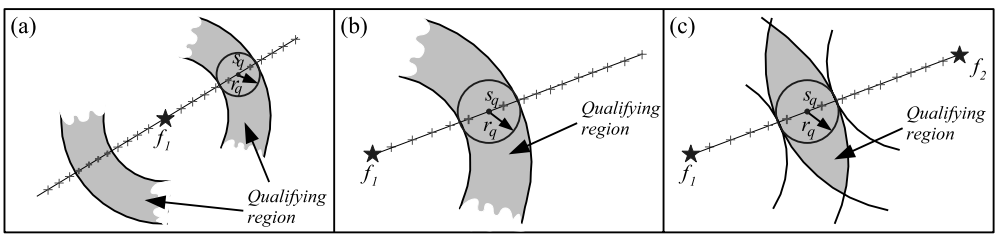

Fig. 5 Active space (*in gray*) defined by: (**a**) one focus f_1 far from border, (**b**) one focus f_1 near the border, (**c**) two foci f_1 and f_2

be seen at Fig. 4e, as the $mbOr$ rendered by the three foci f_1, f_2 and f_3 does not circumscribe well the query region, demanding more objects comparison during the refinement step of the algorithm.

Two foci lead to the maximum reduction of the $mbOr$ when they are far apart, forming an "orthogonal" vectorial base with the origin coinciding with the query center. However, as the foci are pre-defined, we cannot assume orthogonality centered on a generic query. Moreover, in metric datasets it is not always possible to create "artificial" objects to act as well located and far apart foci (for example, how to create a fingerprint that assumes a particular "position" at the metric space?). Therefore, the maximum reduction cannot always be achieved. Thus, one extra focus can be used to distribute the overhead of having foci not ideally placed for each query. Using this additional focus, the new average maximum reduction occurs if they are far apart and almost equally distant from each other. These new requirements are simple to achieve in a metric space. Hence, a suitable lower bound for the cardinality h of $\mathcal{F}$ would be the next integer that contains the intrinsic dimension $\lceil D_2 \rceil + 1$, as we will show is corroborated by the experiments performed (see Sect. 6.1).

Figures 4 and 5 rely on spatial datasets holding the Euclidean metric to illustrate the concepts in an intuitive manner. However, the same deductive process can be applied over metric datasets, because only distances between objects are employed. Thus, equivalent concepts hold for any metric dataset, and, we use this intuition to propose the following practical guideline to choose the Omni-foci base:

Proposition 1 (*Omni-foci base) An adequate Omni-foci base should be composed of* $\lceil D_2 \rceil + 1$ *equally spaced foci, which are located the most far apart possible from each other, surrounding the other objects of the dataset.*

To obtain the Omni-foci base it is only necessary to use the distances between the dataset objects. But this is just what exists in metric spaces. Two foci are enough for datasets with intrinsic dimension one or less, if they are the most far apart pair of objects in the dataset. The foci base of datasets with intrinsic dimension D_2, such that $1 < \lceil D_2 \rceil \leq 2$ should have three foci defining an equilateral triangle; datasets with D_2, such that $2 < \lceil D_2 \rceil \leq 3$ will lead to four foci defining a tetrahedron, and so on. In many situations there will be no objects allowing to build regular polygons. Therefore, the polygon employed to set the foci must be those with sides the most similar available, with vertices near to the border of the dataset.

4.2 How to choose the foci: the HF algorithm

The objects of the foci base chosen by the proposed algorithm are restricted to those in the dataset because, in general, it is impossible to synthesize a specific object in metric domains, as can be done, for example, in spatial domains. This may lead to foci not ideally located, that is, not surrounding the dataset or not positioned on vertices of a perfect hyper-tetrahedron. Moreover, a naive, exhaustive algorithm to find the best possible foci base must try every possible combination of h foci among the N objects, leading to an algorithm with complexity $O(N!/(N-h)!)$, given by the permutations formula from Statistics, where N is the number of objects in the dataset, and h is the number of foci. An index structure can possibly reduce this complexity, but we expect such an algorithm shall be costly.

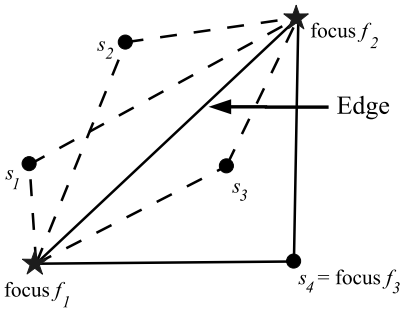

Fig. 6 Given that f_1 and f_2 are already choosen as foci, the algorithm selects the third focus between s_1, s_2, s_3 and s_4. The best is s_4

Instead, we propose the "Hull of Foci" (HF) algorithm to find a good foci base, which performs $O(N)$ distance calculations to select foci near to the hull of the dataset, that comes close to a hyper-tetrahedron. The algorithm starts searching for a pair of objects far apart (see Fig. 6). It randomly chooses an object s_x, find the farthest object f_1 from s_x, and set f_1 as the first focus. With this first round of distance calculations, the object f_1 is found near the border of the dataset. The next step is to find the farthest object f_2 from f_1, setting it as the second focus, and storing the distance $d(f_1, f_2)$ as $edge$. The next foci will be chosen from the objects that have the most similar distances to the previously chosen foci. To do this, using the foci already chosen, for each object s_i not yet picked as a focus, the following error is calculated:

$$\text{error}(s_i) = \sum_{g \in \mathcal{F}} |\text{edge} - df_g(s_i)| \tag{3}$$

Now, select as the next focus the object s_i that produced the minimum $error(s_i)$. This last step is repeated until the required number of foci is selected. The HF algorithm is presented in Algorithm 1. It requires only $h \times N$ distance calculations and yet, the $(h-1) \times N$ calculations required in steps 3 and 6 of Algorithm 1 correspond to distances from each object to a focus, which must be calculated anyway as they are part of the Omni-coordinates of each object. Therefore, the extra distance calculations performed by the algorithm are just those occurring in step 2, which accounts for only N extra distance calculations. Hence, the HF algorithm select the foci-base and also prepares the Omni-coordinates (in steps 3 and 6) of the dataset. Note that in this case, another step is required to calculate the values $df_h(s_i)$ for the last focus f_h chosen, completing the Omni-coordinates.

The HF algorithm does not need to search the full dataset to select the Omni-foci base. Our experiments have shown that suitable foci can be chosen using a well-distributed

sampling of the dataset. Thus, new objects can be inserted or deleted without changing the Omni-foci base. In fact, the presented HF algorithm chooses the foci from the dataset, but theoretically a focus does not need to be part of the dataset.

Algorithm 1 The HF algorithm to find the Omni-foci base in the hull of a dataset S.

Input: the dataset S and the number of foci h.
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 farthest object f_2 from f_1. Insert f_2 in $\mathcal{F}$.
3. Set $edge = d(f_1, f_2)$.
4. While there are foci to be found, do:
5. For each $s_i \in S$, $s_i \notin \mathcal{F}$: calculate $error(s_i) = \sum_g^{g \in \mathcal{F}} |edge - d(f_g, s_i)|$
6. Select $s_i \in S$ such that $s_i \notin \mathcal{F}$ and $error_i$ is minimal.
7. Insert s_i in $\mathcal{F}$.

End

4.3 Proof of no false dismissals

As mentioned before, two steps should be performed to process a query taking advantage of the Omni-technique: filtering and refinement. In the filtering step, the triangular inequality is used to prune the Omni-coordinates that cannot be part of the answer, creating a candidate set. Those that could not be pruned, the false alarms, are then discarded during the refinement step.

The distance calculation in the second step guarantees that there are no false alarms in the response set, discarding any additional proof, because the object itself is compared to the query object. On the other hand, nothing was said about false dismissals. In this section we will prove that the use of the Omni-coordinates in conjunction with the $mbOr$ during the filtering step does not cause false dismissals.

The following proof refers to a range query. Nearest-neighbor queries can be seen as a sequence of nested range queries with a decreasing radius. Thus, the proof for nearest-neighbor queries follows similar considerations.

Given a $RQ(s_q, r_q)$ and an Omni-foci base $\mathcal{F} = \{f_1, f_2, \ldots, f_h\}$, 2 from Definitions 2 and 4, we have:

$$|df_g(s_q) - df_g(s_i)| \leq r_q, 1 \leq g \leq h \tag{4}$$

where, s_i is an object stored in a dataset managed by the Omni-technique. Therefore, we can state the following two Lemmas.

Lemma 1 *If s_i belongs to the response set, it must comply to* Eq. (4).

Lemma 2 *If s_i does not comply to* Eq. (4)*, it would not belong to the response set and neither to the candidate set.*

Proof of Lemma 1 Hypothesis: s_i belongs to the response set (i.e., it is inside $Rq(s_q, r_q)$)

$$d(s_q, s_i) \leq r_q \tag{5}$$

by symmetry and triangle inequality properties and Definition 5 (recalling that $df_g(s) = d(f_g, s)$), we have

$$|df_g(s_q) - df_g(s_i)| \leq d(s_q, s_i) \tag{6}$$

Combining (5) and (6) gives:

$$|df_g(s_q) - df_g(s_i)| \leq d(s_q, s_i) \leq r_q \tag{7}$$

thus:

$$|df_g(s_q) - df_g(s_i)| \leq r_q \tag{8}$$

□

Proof of Lemma 2 Hypothesis: s_i does not attend to Eq. 4, so

$$r_q < |df_g(s_q) - df_g(s_i)| \tag{9}$$

combined with (6) and (8) gives

$$r_q < |df_g(s_q) - df_g(s_i)| \tag{10}$$

and

$$|df_g(s_q) - df_g(s_i)| \leq d(s_q, s_i) \tag{11}$$

thus:

$$r_q < d(s_q, s_i) \tag{12}$$

which implies that s_i does not pertain to the query result set. □

Therefore, Lemma 1 and Lemma 2 prove that the Omni-technique enables answering queries with no false dismissals.

5 The members of the Omni-family

This section presents how to use the Omni-technique with sequential scan, B-Trees and R-Trees as underlying access methods. The general approach of the technique regarding the data structure construction was described in Sect. 4. In this section we emphasize the specialized version of the similarity search algorithms for each Omni member.

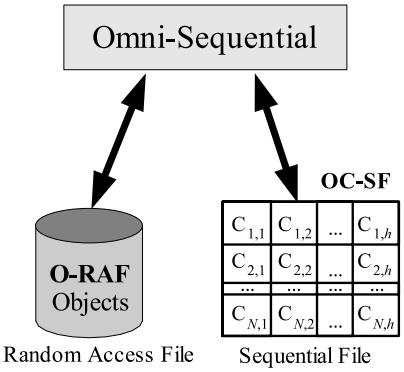

Fig. 7 Schematic view of an Omni-Sequential, with the RAF storing the objects and the sequential file storing the respective Omni-coordinates

5.1 The Omni-technique and the sequential scan: the Omni-sequential

In the Omni-Sequential, the Omni-coordinates are managed by a sequential file, where each entry corresponds to the array of Omni-coordinates of an object s_i and its internal object identification ($IOid$). From now on we will call the random access file as O-RAF, and the sequential file with the Omni-coordinates as OC-SF. Figure 7 presents a schematic view of the Omni-Sequential files.

Executing, through the Omni-Sequential, a range query with query radius r_q centered at the query object s_q, starts with a filtering operation, where the OC-SF is sequentially scanned, comparing the Omni-coordinates of each stored object s_i to the Omni-coordinates of s_q. The first step of the algorithm is to calculate the distance $df_g(s_q)$ from the query object s_q to each focus f_g, creating the Omni-coordinates $C(s_q)$ for s_q. Then, for each Omni-coordinate in the sequential file, for each focus g, where $1 \leq g \leq h$, verify if $|df_g(s_i) - df_g(s_q)| > r_q$. If this comparison holds for at least one focus, then discard the object s_i and skip the corresponding distance calculation between s_i and s_q. Notice that $df_g(s_i)$ was already calculated and is stored in the OC-SF as the $C_{i,g}$ coordinate.

If the distance calculation cannot be skipped, the correspondent object s_i must be retrieved from the O-RAF, what is done through the internal $IOid(s_i)$ associated to the Omni-coordinate, and compared to the query object to avoid false alarms. If $d(s_i, s_q) \leq r_q$ then include s_i in the answer set $AS[]$. Otherwise, discard s_i.

The k-nearest neighbor query is equivalent to a range query with decreasing radius, and the candidate set corresponds to an ordered list of size k, where the query radius is initially set to infinity. The ordering criterion of the list is the distance between the neighbors and the query object. The algorithm is described in Algorithm 2, where s_q is the query object, $C(s_q)$ are the Omni-coordinates of s_q, k is the desired number of neighbors, r_q is the decreasing radius and $AS[k]$ is the farthest element already put in the answer set.

An object is retrieved from the O-RAF and compared to the query object only when no focus can prune it, in both kinds of similarity queries. This enables a large reduction in the number of distance calculations, as shown in Sect. 6.

Algorithm 2 The k-nearest-neighbor query for the Omni-Sequential.

Input: the dataset S in **O-RAF**, the dataset of Omni-Coordinates **OC-SF**, the query object s_q and the number of neighbors k.

Output: the list AS of nearest neighbors NN.

```
Begin
 1. Calculate the Omni-Coordinates
    C(s_q) = {df_g(s_q), 1 ≤ g ≤ h} of the query
    center s_q.
 2. Insert the first k objects from O-RAF in the
    candidate list AS, keeping the list ordered by the
    distances from each object to s_q, so that the object
    stored in AS[k] is the farthest from s_q, and set
    r_q = d(AS[k], s_q).
 3. For each Omni-coordinate df_g(s_q) ∈ C(s_i)
    in OC-SF, do:
 4. Begin
 5.   For each g, where 1 ≤ g ≤ h, do:
 6.     If |c_{i,g} − df_g(s_q)| > r_q then discard the object s_i.
 7.   If s_i was not discarded by any focus then:
 8.   Begin
 9.     Retrieve s_i from O-RAF.
 10.    If d(s_i, s_q) < r_q then:
 11.      Insert s_i into the candidate list AS[], keeping
          it ordered, and
 12.      Set r_q = d(AS[k], s_q).
 13.  End.
 14. End.
End
```

5.2 Indexing the Omni-coordinates in B-Trees: the OmniB-Forest

Objects in a metric space do not embody the notion of order. Therefore, access methods based on the property of total ordering relation on the data domain, such as B-Trees, cannot directly index them. However, the distances $df_g()$ from each focus g to the set of objects can be ordered and indexed by B-Trees. Therefore, it is possible to replace the Omni-coordinates file by a set of h B-Trees, one per focus, where each entry in the gth B-Tree is composed of the distance $df_g(s_i)$ and $IOid(s_i)$. We call this member of the Omni family as the OmniB-Forest. Figure 8 shows the OmniB-Forest schematic view, with the O-RAF storing the objects, and the B-Forest storing the Omni-coordinates, now called the OC-BF. The importance of this member of the Omni-family is that it provides an effective support to metric datasets using the resources already present in commercial DBMS [2].

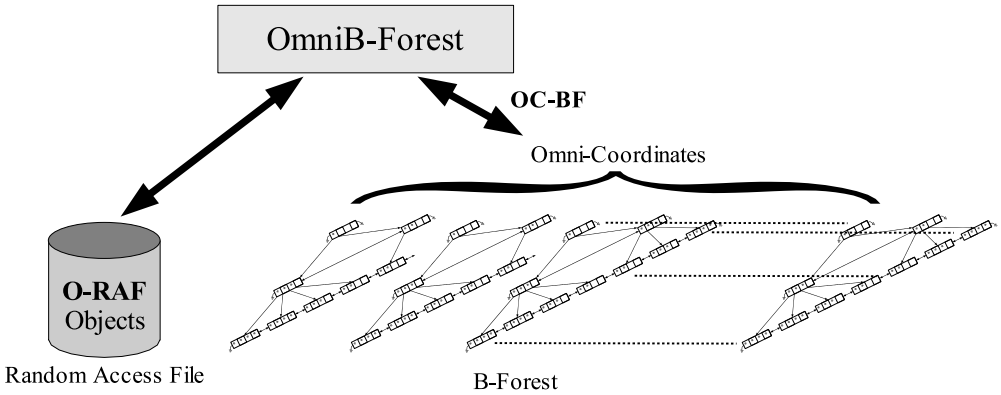

Fig. 8 Schematic view of the OmniB-Forest, with the RAF storing the objects, and a forest of B$^+$-Trees indexing the Omni-Coordinates

The candidate set for the filtering step of a range query is obtained by the intersection of the candidates given by each individual focus. To fasten the intersection operation, we use an array V of counters, one counter for each object $IOid(s_i)$ in the dataset. Each element counts how many foci consider the respective Omni-coordinate as a candidate. The refinement step consists of traversing the array V looking for every position topping to the number of foci h. The process is described in Algorithm 3, where s_q is the query center object, $C(s_q)$ keeps the Omni-coordinates of s_q, and r_q is the query radius.

Algorithm 3 The range query for the OmniB-Forest.

Input: the dataset S in **O-RAF**, the set of B-Trees built over the Omni-Coordinates **OC-BF**, the query object s_q and the query radius r_q.
Output: the response set.

Begin
1. Calculate the Omni-Coordinates $C(s_q) = df_g(s_q)$, $1 \leq g \leq h$ of the query center s_q.
2. Be V an array of counters, one counter for each object s_i in **O-RAF**. Set all elements of V to zero.
3. For each g, where $1 \leq g \leq h$, do: // *for each B-Tree*
4. Begin // *Filtering step*
5. Define the interval of valid keys $I_g = [df_g(s_q) - r_q, df_g(s_q) + r_q]$.
6. Traverse the B-Tree corresponding to the focus g, searching for the coordinates (keys) that lies in the interval I_g, and:
7. For each object s_i whose $df_g(s_i)$ coordinate is inside the I_g interval, get the respective $IOid(s_i)$, and increment the value of $V[IOid(s_i)]$.
8. End.
9. Sequentially scan the array V and for each s_i where $V[i] = h$, do: // *Refinement step*
11. Retrieve s_i from the O-RAF, and if $d(s_i, s_q) \leq r_q$ then insert s_i into the response set.

End

As the intrinsic dimension of the majority of the datasets tends to be small (usually less than 10), so is the number h of foci in Omni members. Therefore, in our implementation we represented the array of counters V as an array of unsigned bytes, maintained in main memory. If scalability for very huge datasets is needed, the array can be replaced by a binary tree or a hash table indexing a sparse set of $IOids$, or even on multiple passes over the B-Tree, similarly to the merge-join algorithm.

Searching each gth B-Tree corresponds to retrieving a set of keys sorted by the distance of each object s_i to the focus f_g. Therefore, we used a B$^+$-Tree to have all objects in the leaf nodes and all nodes in the same level linked with bi-directional pointers to improve the k-nearest-neighbor algorithm, as described later. The search starts with a depth first search, to find the first key greater than or equal to $df_g(s_q) - r_q$, and a subsequent forward sequential scan until the upper limit of the interval I_g, i.e., until $df_k(s_q) + r_q$. As the nodes are double linked, we also improved the tree occupancy by inspecting both the left and the right siblings of a node before a split, like in a B*-Tree. If the left (right) sibling has a free slot, it will receive the first (last) key of the overfilled node.

Another issue that needs attention is the key collision: it may happen that distinct objects are at the same distance from one focus. The frequency of collisions is highly dependent of the domain of the metric in use. As the use of a collision list inside the B$^+$-Trees would increase the complexity of the algorithms for all kind of metrics, we changed the algorithms of insertion and searching. The traversal of a deep first search always gets the first occurrence of the key and the insertion point is always prior to the first occurrence of a key with the same value. Figure 9 shows a B$^+$-Tree with all nodes double linked and key collisions.

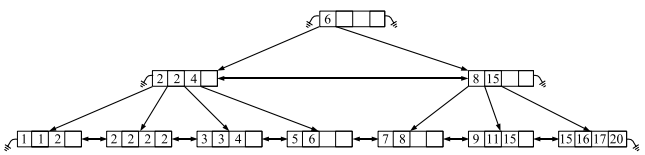

Fig. 9 Schematic view of the B$^+$-Tree used in the OmniB-Forest where all nodes are at the same level are double linked and key collisions are allowed

Algorithm 4 The k-nearest-neighbor for the OmniB-Forest.

Input: the dataset S in **O-RAF**, the set of B-Trees built over the Omni-Coordinates **OC-BF** (where t_g is the gth tree), the query object s_q and the number of neighbors k.
Output: the set of nearest neighbors NN in AS.

Begin
1. Calculate the Omni-Coordinates $C(s_q) = \{df_g(s_q), 1 \leq g \leq h\}$ of the query center s_q.
2. Let V be an array of counters, one counter for each object s_i in **O-RAF**. Set every element $V[i]$ to zero.
3. Insert the first k objects from **O-RAF** in the answer set AS, keeping the list sorted by the distances from each object to s_q, so that the object stored in $AS[k]$ is the farthest from s_q, and set $r_q = d(AS[k], s_q)$.
4. For each g, where $1 \leq g \leq h$ do:
5. Begin
6. Define the interval of valid keys $I_g = [df_g(s_q) - r_q, df_g(s_q) + r_q]$.
7. Execute a deep first search for the first key value $\geq df_g(s_q)$ at tree t_g.
8. Set $back\text{-}cursor_g$ and $forth\text{-}cursor_g$ to the position returned by the search.
9. End.
10. Set $g = 1$;
11. While (it is possible to do a step backward or forward for at least one of the trees) do:
/* that is, one of the following conditions hold for at least one tree: $back\text{-}cursor_g$ is not at the beginning of the index or $forth\text{-}cursor_g$ is not at the ending of the index, or $back\text{-}cursor_g > I_g^{inf}$, and $forth\text{-}cursor_g < I_g^{sup}$) */
12. Begin // *Filtering step*
13. Define the interval of valid key $I_g = [df_g(s_q) - r_q, df_g(s_q) + r_q]$.
14. Execute *step-backward()* on B-Tree g.
15. If $back\text{-}cursor_g \geq I_g^{inf}$, then get $i = IOid(back\text{-}cursor_g)$ and increment $V[i]$.
16. If $V[i] = h$ do:
17. Begin
18. Retrieve s_i from the **O-RAF**.
19. If $d(s_i, s_q) < r_q$ then:
20. Begin // *Refinement step*
21. Insert s_i into the candidate list $AS[]$, keeping it ordered, and
22. Set $r_q = d(AS[k], s_q)$.
23. End.
24. End.
25. Repeat the steps 14-24, changing: *step-backward()* for *step-forward()* in step 14, and $back\text{-}cursor_g \geq I_g^{inf}$ by $forth\text{-}cursor_g \leq I_g^{sup}$ in step 15.
26. Set g cyclically to the next tree.
27. End.
End

The k-nearest-neighbor query requires an algorithm distinct from the usual approach of performing a range query with decreasing radius. Initially setting the current radius r_q to infinity is not useful for the OmniB-Forest, because it would lead to get the starting of the interval to the very first key of each tree. The solution we developed is the opposite, starting with a current radius r_q equal to zero, increasing it until k objects are found. Therefore, each B-Tree is searched to find the position where the Omni-coordinates of the query object s_q should be, in a point query, afterward stepping backward and forward through each tree, looping around every tree after each pair of steps.

To do so, two functions were created to support this process: *step-backward* and *step-forward*, running over a B-Tree. Both functions return the key of the gth B-Tree (the pre-computed $df_g(s_i)$ distance) and the IOid(s_i) of the new position. Two cursors defined for each B-Tree, *back-cursor* and *forth-cursor*, allow respectively to fetch one key back and one key forth in the tree. The same vector V is used to keep track of the candidates. This process is described in Algorithm 4, where s_q is the query object, $C(s_q)$ are the Omni-coordinates of s_q, k is the desired number of neighbors, r_q is the decreasing radius, $AS[k]$ is the farthest element of the already found answer set, and I_g^{inf} and I_g^{sup} are, respectively, the minimum and maximum values of the current interval I_g defined by the current search radius r_q. Step 3 sets an initial maximal span of nodes in the B-Trees that the algorithm is allowed to navigate.

In fact, the B-Tree inner structure is used only in step 7 to set the $back\text{-}cursor_g$ and $forth\text{-}cursor_g$ for each foci g. After that, two leaf nodes of each B-Tree is maintained in main memory to execute the *step-backward()* and *step-forward()* functions. The experimental results, presented in Sect. 6, show impressively low values for distance calculations with k-nearest-neighbor queries when using the OmniB-Forest.

5.3 Storing the foci in R-Trees: the OmniR-Tree

The R-Tree is a spatial access method developed to index vectorial data where each element is represented as a minimum bounding rectangle (MBR). The MBR of an object is composed of the minimum intervals at each dimension that bounds it. Individual objects in a metric space do not define a "region", but we can imagine that each object lies inside the $mbOr$ of a range query with radius r_q=0, that is, a point query, centered at that object. Therefore, the $mbOr(s_i, 0)$ of an object s_i is an MBR in the space of coordinates defined by the Omni-foci base, where the initial and the final values of each interval are equal. Therefore, the Omni-coordinates file is replaced by an R-Tree to improve the filtering step. In this case, each entry is composed of the $mbOr(s_i, 0)$ and the $IOid$ of s_i. We call this new metric access method as the OmniR-Tree.

Figure 10 shows the schematic view of the OmniR-Tree where an MBR of the R-Tree bounds the minimum and maximum values for each foci, leading to regions in the original (metric) space. The figure shows the object file O-RAF (see

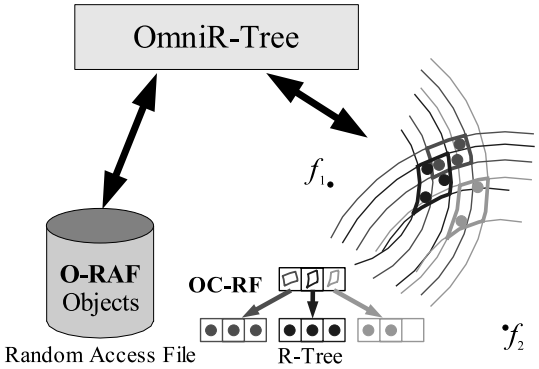

Fig. 10 Schematic view of the OmniR-Tree, with the RAF storing the objects, and the R-Tree indexing the Omni-Coordinates and the $mbOrs$

Sect. 5.1) and the R-Tree with the Omni-coordinates (OC-RT). The proof presented at Sect. 4.2 guarantees the OmniR-Tree usability.

The benefits of using an OmniR-Tree instead of R-Trees are two-fold: first, it permits the use of R-Trees even for nonvector metric datasets; second, for vector datasets, as the number of foci is close to the intrinsic dimension of a dataset, which is usually much smaller than the embedding dimension, it permits to reduce the dimensionality of the MBR of the R-Tree, to increase the number of objects per node and to reduce the tree height, thus improving the speed of retrieval operations.

The algorithms to perform insertion, node partitioning and range queries in OmniR-Trees are the same used in R-Trees. However, special care must be taken with respect to the algorithm used to perform nearest-neighbor queries. In fact, there are two approaches. The first one is to estimate a final radius. Nearest-neighbor queries do not know the final query radius, but an estimated equivalent radius can be adopted using the selectivity estimation formulas provided by the use of fractal theory [3, 6, 18, 34]. This number can be corrected by successive iterations if the requested number of objects is not retrieved. This approach can be used with any implementation of the R-Tree, so it is not specific to the use of R-Trees with the Omni-technique.

The second approach corresponds to adopting the same algorithm used to find nearest neighbors in metric trees. In this case, the Omni-coordinates of a hypothetical center of the minimum bound rectangle of each node of the R-Tree are used to define the traverse order of the subtrees originating in each node. We used this second approach to evaluate the OmniR-Tree, whose results are presented in the next section.

The OC-RT structure is an R-Tree that groups in the same node the coordinates of objects that are close to each other. Thus, it is expected that, when processing a leaf node, if it is not possible to prune an $mbOr$, most of the others in the same node will have low probability to be pruned too. As a consequence, many of the objects indexed in a leaf node of the OmniR-Tree must be restored from the O-RAF and compared to the query object. The O-RAF corresponds to a random access file, where many objects can be stored in a single page. To reduce disk accesses in the O-RAF, it is worthwhile to group the objects in the O-RAF following the way that their Omni-coordinates are grouped in the leaf nodes of the R-Tree in the OC-RT. Although this synchronization does not need to be performed after every update in the R-Tree, it is valuable after major updates, as a post-insertion optimization task.

We implemented a small function to perform this synchronization of the R-Tree's leaf nodes and the O-RAF's pages. The R-Tree is fully traversed following the order of its pointers, and the O-RAF is sorted to store the objects as they appear during the tree traversal. At the same time, the new $IOid$ returned by the O-RAF is saved with the respective Omni-coordinate in the R-Tree. No additional distance computations are executed during the synchronization, so this function is cheap regarding CPU time. The experiments presented in Sect. 6 show that this optimization task considerably improves the performance of the OmniRT-ree.

In the next section, we present experimental results comparing the three presented members of the Omni-family, with other representative related access methods.

6 Experiments

To evaluate the effectiveness of the Omni-technique, we worked on a variety of datasets, both from the real world and syntheticones. Table 2 contains the description of the representative datasets used in the experiments including: the name of the dataset, the intrinsic (D_2) and embedded (E) dimensions, the number of objects in the dataset, the number of objects used as foci, the metric used and a brief description of the dataset. All the algorithms were implemented in C++ language and run on an AMD Athlon XP 2600 processor, with 256 MB of RAM memory and a hard disk of 40 GB spinning at 7200 RPM, running the MS Windows 2000 operating system. The measurements are the average number of distance calculations, the average number of disk accesses and the total time (in seconds), to perform 500 queries. We instrumented our implementation to count the number of distance calculations and the number of disk accesses. For the Omni-family members the reported number of distance calculations includes distances to foci and to data objects. We have executed range queries (R_q) with varying radius and k-nearest-neighbor queries (k-NN) for a several number of neighbors. Each measured point in a plot corresponds to 500 queries with the same radius or number of neighbors, using 500 different query objects (for coherence we have used the same query objects for the different points in the plot), where the query objects are samples extracted from the respective datasets.

Every member of the Omni-family (Omni-Sequential, OmniR-Tree and OmniB-Forest) was configured to use a fixed disk page size of 4 KB. The R-Tree, Slim-Tree, M-Tree and sequential scan (SeqScan), except when indexing the *Histograms* dataset, were also configured to a disk page size of 4 KB. It was not possible to index the *Histograms* dataset using the R-Tree with page size equal to 4 KB, because there

Table 2 Datasets used for the experiments

Dataset name	# Objects	E	D_2	# Foci	Metric	Description
English Words	25,143	–	4.75	6	L_{Edit}	Words from the English language dictionary.
Eigenfaces	11,900	16	4.67	6	L_2	Feature vectors extracted from human face images with the Eigenfaces method. From Informedia project [46] at Carnegie Mellon University - USA.
Histograms	4,247	256	2.5	4	L_2	Grey level histograms of medical X-Ray images obtained from the Clinical Hospital of Ribeirao Preto.
VideoVectors	79,094	50	7.73	9	L_2	Feature vectors extracted from the Video Library of the Informedia project.
Synthetic30D	1,000,000	30	3.24	5	L_2	Synthetic vectors built as follows: 5 dimensions corresponding to coordinates of points uniformly distributed among the diagonal of a 5 d-hipercube; 4 dimensions corresponding to coordinates of points uniformly distributed among the diagonal of another 4 d-hipercube; one dimension was randomly created; every remaining 20 dimensions are linear combinations of the previous ones.

The *EnglishWords* and *Histograms* can be obtained contacting the authors. For the *Eigenfaces* and *VideoVectors* contact the Informedia Project at informedia@cs.cmu.edu.

was an insufficient number of entries in each node. This situation happens when the dimension of the dataset is high. The R-Tree was, therefore, set to work with a page size of 10 KB for the *Histograms* dataset. The Slim-Tree, the M-Tree and the Sequential Scan were also configured to work with page size of 10 KB. Every member of the Omni-technique uses pages of size 4 KB, to demonstrate that the Omni-technique is very suitable to index complex dataset with varied sizes. The results from the Slim-Tree were measured after executing the *Slim-Down* optimization algorithm.

6.1 Analysis on the number of foci to be used

Here we analyze the behavior of the Omni-technique with varying number of foci while answering similarity queries. We present the results obtained with Omni-Sequential for the datasets *Eigenfaces* and *Histograms*, as the other datasets presented similar behavior. Figure 11 shows the number of distance calculations and time required to answer range (for radii 0.04 and 0.08) and k-NN queries (for $k = 5$ and $k = 10$) as a function of the number of foci.

As we can see in the first row of Fig. 11 (*Eigenfaces*, which has $D_2 = 4.75$), increasing the number of foci up to seven, gradually reduces the number of distance calculations and the total time (second row). But after this, the improvements in the $mbOrs$ are overcome by the increase of the distance calculations demanded by each extra focus. The effect is the increase in time shown after the 7th focus (graphs (e)–(h) of Fig. 11), for both range and k-NN queries.

The behavior shown by the experiments on the *Eigenfaces* dataset is exemplary, reflecting what happened with the majority of the datasets. However, it is interesting to observe what occurred with a high-embedding/low-intrinsic dimensionality dataset, such as the *Histograms* dataset. The total time and the number of distance calculations for this dataset is shown at the bottom row of Fig. 11. As we can see, the best gain happens around the 4th focus, and after that the graph almost flattens. From those experiments, the statement that the number of foci should follow the value of the intrinsic dimensionality of the dataset (see Sect. 4.1) is corroborated: the best gain in all datasets regarding total time and number of distance calculations comes when using the number of foci $h = \lceil D_2 \rceil + 1$.

6.2 Synchronizing the object file in OmniR-Tree

The synchronization process proposed in Sect. 5.3 reorganizes the objects inside O-RAF, ordering them following the organization of their Omni-coordinates in the OC-RT (R-Tree). Figure 12 shows the results for the average number of disk accesses and total time while answering 500 similarity queries with varying radius and number of neighbors. The plots for distance calculations are the same as for the next experiments, so these graphs are shown in the next Section in Figs. 14 (graphs (a) and (d)) and 15 (graphs (a) and (d)).

Figure 12 shows that the improvements regarding both disk accesses and total time are expressive. For the *EnglishWords* dataset (first row of Fig. 12) the synchronized OmniR-Tree executes up to three times fewer disk accesses and spends up to 25% of the time demanded by the unsynchronized OmniR-Tree. The second row of Fig. 12 (*Eigenfaces*) presents even larger improvements (so the measurements are presented in log scales), the synchronized OmniR-Tree performs up to 15 times fewer disk accesses and is up to eight times faster than the unsynchronized OmniR-Tree. It is worth to highlight that the process of synchronization does not require any additional distance calculation, involving only disk reads and writes, making the process reasonably fast. It took less than 1 s for most of the cases, including those shown in Fig. 12.

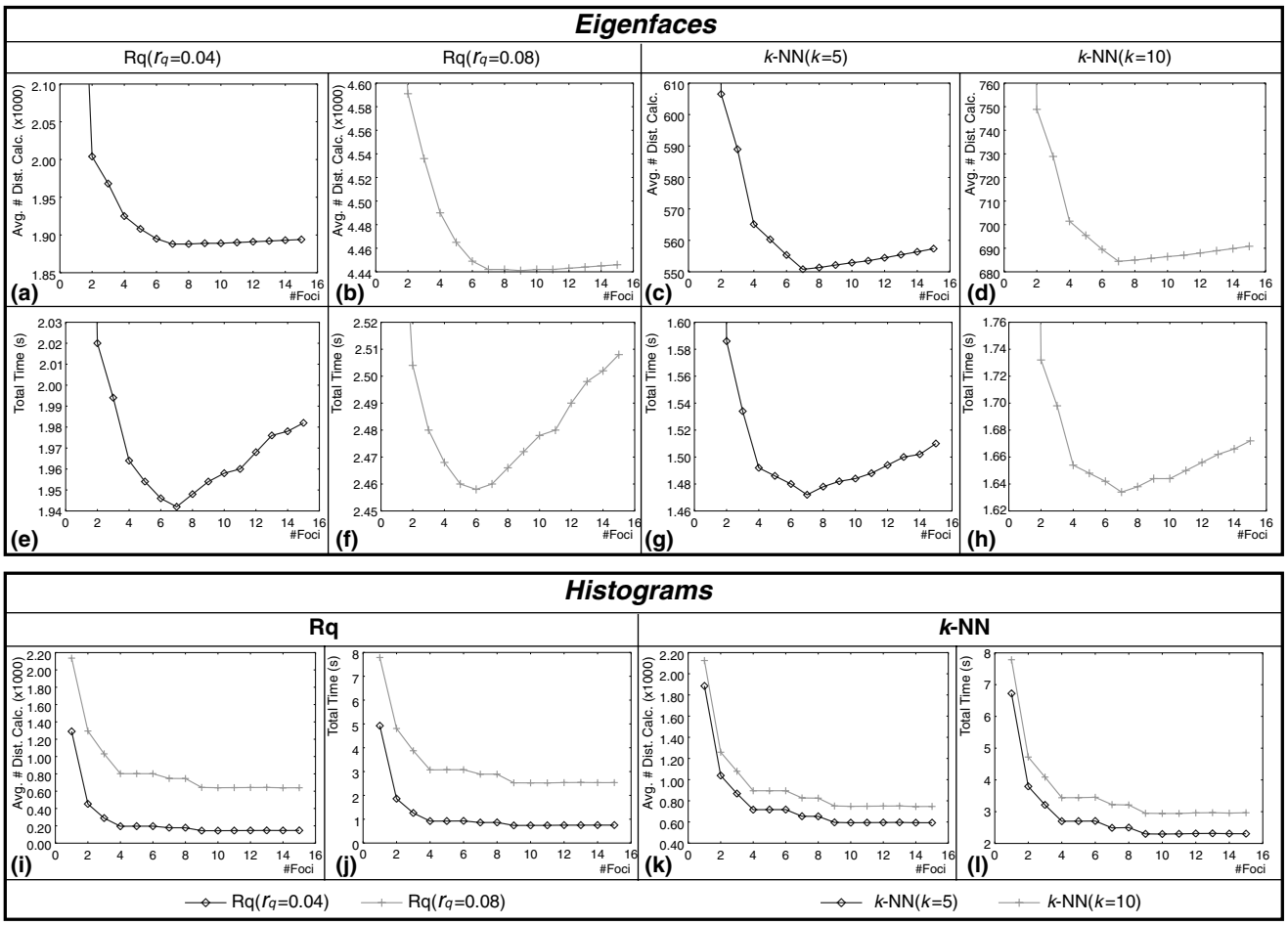

Fig. 11 Results for Range query, with $r_q = 0.04$ (graphs **a** and **e**), $r_q = 0.08$ (graphs **b** and **f**), and both (graphs **i** and **j**). For k-nearest neighbor query, with $k = 5$ (graphs **c** and **g**), $k = 10$ (graphs **d** and **h**), and both (graphs **k** and **l**), comparing: the average number of distance calculations (graphs **a**–**d**, **i**, **k**) and the total time (graphs **e**–**h**, **j**, **l**) when querying the *Eigenfaces* (rows 1 and 2) and the *Histograms* (row 3) datasets with the Omni-Sequential, increasing the number of foci

6.3 Comparing the Omni-family with other access methods

This section compares the three members of the Omni-family of access methods presented (the Omni-Sequential, the OmniR-Tree and the OmniB-Forest) with the sequential scan, the traditional R-Tree, and to existing metric access methods (the VP-Tree, the M-Tree and the Slim-Tree). The motivation here is to compare the Omni-family with representative access methods from both models (vector space and metric) to answer similarity queries. Figure 13 shows the results of indexing the *Histograms* dataset with the Omni-family members, the VP-Tree, the M-Tree and the Slim-Tree for range (graph (a)) and k-NN (graph (b)) queries. As the VP-Tree works only in memory, the number of disk accesses and total time cannot be measured in a common base, so here we show only results for the number of distance calculations. Moreover, the results of the Slim-Tree for every experiment were better than those of the M-Tree. Therefore, for the remaining experiments we compare the Omni-family members only with the Slim-Tree, as a representative MAM.

Figures 14 and 15 show the average number of distance calculations, the average number of disk accesses and the total time (in seconds) to index the following datasets: *EnglishWords* a metric dataset; *Histograms* a vector dataset with high embedded dimension, but low intrinsic dimensionality; and *Eigenfaces* a vector dataset with intermediary embedded and intrinsic dimensions; *VideoVectors*, also a vector dataset, but with high embedded and relatively high intrinsic dimensions. The graphs for *EnglishWords* (first row of Figs. 14 and 15) do not include the R-Tree, because this

Table 3 Selectivity of range queries for the *EnglishWords* dataset

Radius	*EnglishWords*	
	Avg. no. obj.	SD
1	3.29	4.82
2	34.56	67.79
3	287.39	477.40
4	1393.61	1742.56
5	4161.67	3758.35
6	8678.99	5635.49

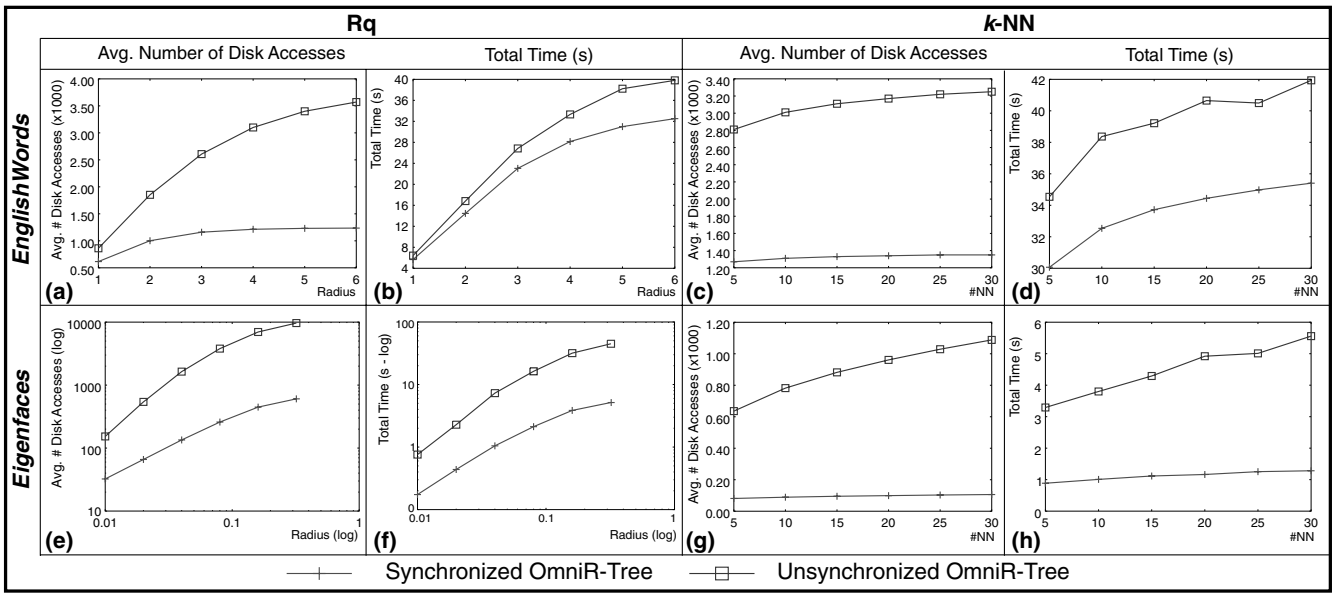

Fig. 12 Results for Range query varying the radius, and k-nearest neighbor query varying k, comparing: the average number of disk accesses for Rq in graphs **a** and **e**, and for k-NN in graphs **c** and **g**; and total time for Rq in graphs **b** and **f**, and for k-NN in graphs **d** and **h**; of the OmniR-Tree access method before (Unsynchronized OmniR-Tree) and after (Synchronized OmniR-Tree) the synchronization of the object file and the R-Tree (see section 5.3), over the *EnglishWords* (row 1) and *Eigenfaces* (row 2) datasets

method can only index vectorial data. The results for range queries with the vector datasets are plotted in log–log scale to allow visualizing a wide range of radii. The radii values were chosen aiming to show the behavior of the methods within a selectivity ranging from a couple of objects to more than 80% of the dataset. Table 3 contains the average number of objects (avg. no. obj.) and the standard deviation (SD) returned by the queries for each range radii (1 through 6) for the *EnglishWords* dataset and Table 4 for the *Eigenfaces*, *Histograms* and *VideoVectors* datasets. For nearest-neighbor queries, the k values were chosen to reflect the usual requests performed on real datasets. Next we analyze the three categories of measurements.

6.3.1 Average number of distance calculations

The first column of Figs. 14 and 15 presents the average number of distance calculations when answering range queries for all datasets, aiming to compare the performance of the methods. We do not show results for the R-Tree, because it relies on comparisons of the coordinates of the bounding rectangles. As expected, the number of distance calculations for the members of the Omni-family are always the same, because the same foci base were used by all of them, what gives the same $mbOr$ for the same queries. The graphs (a), (d), (g) and (j) of Fig. 14 show that the Omni-family MAMs execute up to 17 times fewer distance calculations, when compared to the basic MAM Slim-Tree (the best performing MAM so far). Another impressive result happened with the *VideoVectors* dataset for the range query. For example, with radius 0.02, corresponding to an average selectivity of 5.58 objects (see Table 4), the Omni-family performed 195 times fewer distance calculations. We believe that this result is related to the high intrinsic dimension of this dataset.

The Omni-family members also presented the best results for distance calculations in k-NN queries. From the

Table 4 Selectivity of range queries for the *Eigenfaces*, *Histograms* and *VideoVectors* datasets

	Eigenfaces		*Histograms*		*VideoVectors*	
Radius	Avg. no. obj.	SD	Avg. no. obj.	SD	Avg. no. obj.	SD
0.1	1.92	2.65	1.32	0.95	–	–
0.2	43.74	51.22	3.14	3.89	5.58	18.97
0.4	842.83	549.56	12.24	15.09	8.95	23.28
0.8	3777.91	1616.07	103.79	92.78	16.13	32.67
0.16	7963.22	2167.99	768.27	472.86	52.70	112.92
0.32	11277.64	1210.26	3068.59	881.00	6660.76	7834.24
0.64	–	–	–	–	68360.88	14730.86

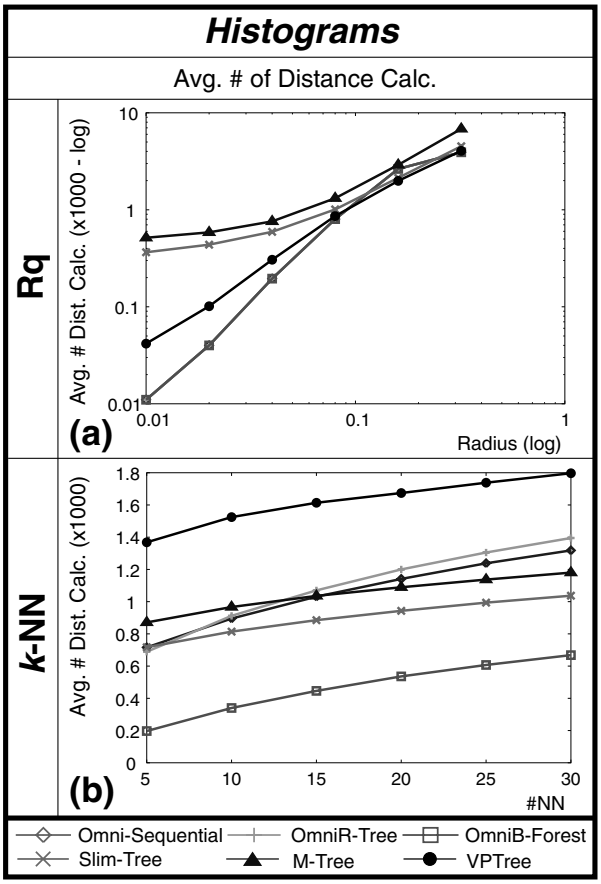

Fig. 13 Results for Range queries, varying the radius, and for k-Nearest neighbor queries, varying k, comparing the average number of distance calculations of Omni-Sequential, OmniR-Tree, OmniB-Forest, Slim-Tree, M-Tree and VP-Tree access methods, over the datasets *Histograms*

first column of Fig. 15 (graphs (a), (d), (g) and (j)) we can see that these methods performs up to five times fewer distance calculations when compared to the Slim-Tree. The results presented by the Omni-Sequential, OmniR-Tree and the OmniB-Forest are quite different from each other. Despite the equality of the foci-base, in this case, as the algorithm for k-NN queries is specialized for each one of the Omni methods, they tend to inspect the objects in a different order. Thus, the convergence of the radius is different for each member, leading to different numbers of distance calculations. Overall, we can see that for distance calculations the winner is the OmniB-Forest, which performs up to three times fewer distance computations than the other two members, and up to five times fewer than the Slim-Tree.

6.3.2 Average number of disk accesses

Regarding the number of disk accesses for range queries, the experiments show that as the size of the object grows, the Omni-family members become increasingly better than the other methods. The size of the objects is an important factor in these measurements. The dataset with the smallest objects is the *EnglishWords* (Fig. 14b), for which the best result is given by the sequential scan, followed by the Slim-Tree. The next dataset regarding the size of its objects is the *Eigenfaces*, with 16-dimensional objects (Fig. 14e), where the Slim-Tree, a method specialized on reducing the number of disk accesses, ties with the OmniR-Tree for small radii, but wins as the radius grows. The measurements from the Slim-Tree were taken after the execution of the Slim-Down algorithm. It is important to notice that among the previous published MAM, the Slim-Tree is the one that in general requires the lowest number of disk accesses. Considering the 50-dimensional *VideoVectors* dataset (Fig. 14k), both the intrinsic dimension and the size of the objects contribute to make the OmniR-Tree the most adequate method, which performs up to 10 times fewer disk accesses than the other methods. Finally, the results obtained from the 256-dimensional *Histograms* dataset (Fig. 14h), the dataset with the largest objects, shows that the three members of the Omni-family are the winners, as the three members require almost the same number of disk accesses. This behavior can be understood through the following observations:

1. the Omni-family methods index the objects through their Omni-coordinates;
2. the size of the Omni-coordinates is defined by the intrinsic dimension, instead of the embedded dimension of the dataset, and it tends to be very low, usually bellow 10;
3. only the file that stores the objects (O-RAF) is directly affected by their size, while the file that indexes the Omni-coordinates is affected by the intrinsic dimension of the dataset;
4. the number of disk accesses is related to the node cardinality of the method, which depends on the size of the object;
5. traditional methods, as Slim-Tree and R-Tree, store the full object in both leaf and internal nodes, so their performance regarding disk accesses are directly dependant on the size of the objects. Another evidence seen from Fig. 14 is that, as the size of the object grows, the number of disk accesses of all traditional methods tends to increase at a faster pace than the members of the Omni-family do.

The previous analysis regarding number of disk accesses and the number of elements per node of the Omni-family holds only for range queries, where it is possible to restrict the candidate set prior to performing the refinement step. Therefore, the O-RAF traversal during the refinement step can be optimized by ordering the $IOid$ of the objects, decreasing the number of disk accesses whenever two candidates lies in the same node. But this optimization cannot be taken into account during k-NN queries, because whenever a candidate is encountered, the respective object must be retrieved from the O-RAF and compared to the query object

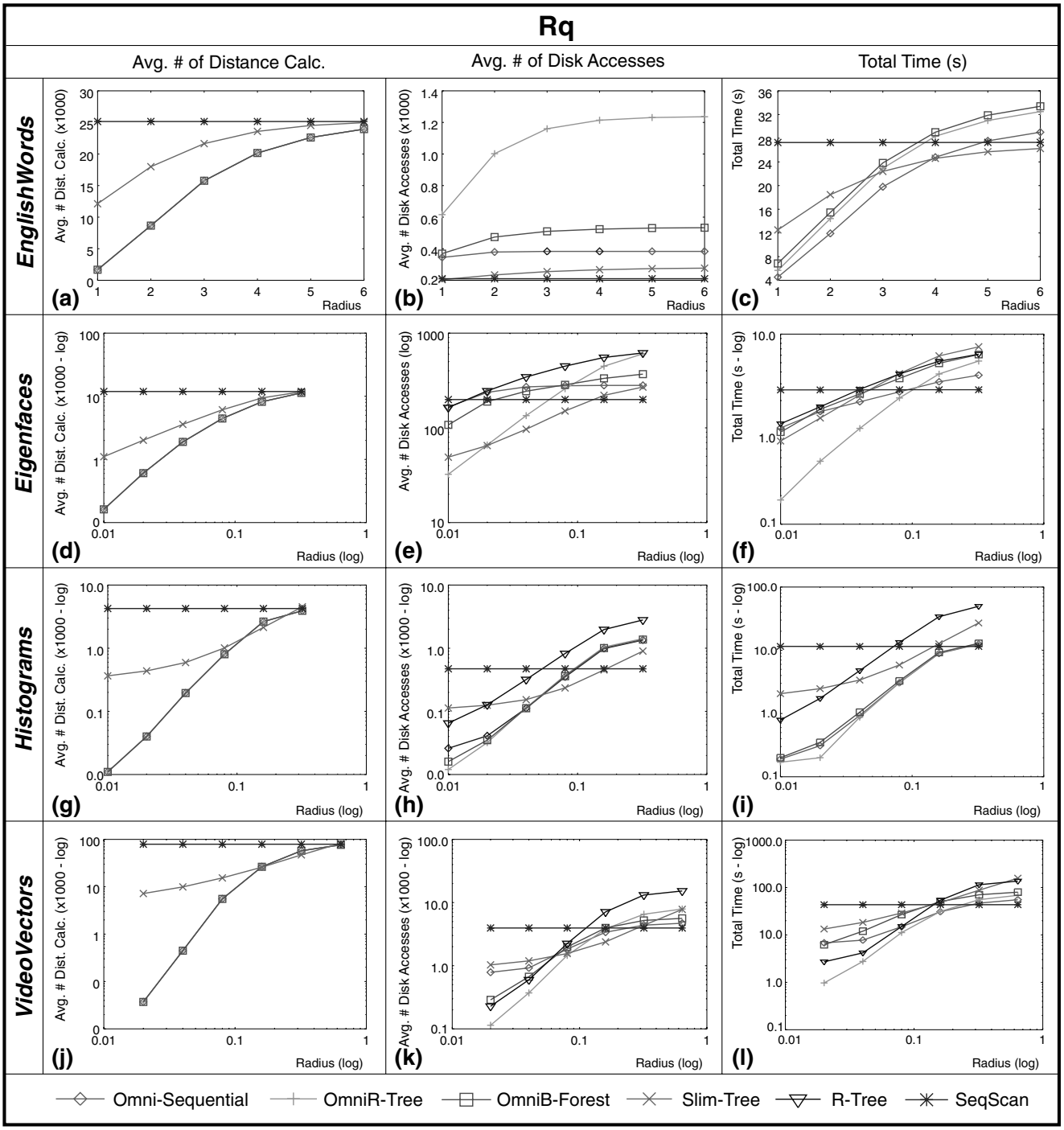

Fig. 14 Results for Range queries, varying the radius, comparing: average number of distance calculations in graphs **a**, **d**, **g** and **j**; average number of disk accesses in graphs **b**, **e**, **h** and **k**; and total time in graphs **c**, **f**, **i** and **l**; of Omni-Sequential, OmniR-Tree, OmniB-Forest, Slim-Tree, R-Tree and sequential scan access methods, over the datasets: *EnglishWords* (row 1), *Eigenfaces* (row 2), *Histograms* (row 3) and *VideoVectors* (row 4)

to reduce the current query radius, avoiding false alarms. Therefore, as can be seen in the second column of Fig. 15 (graphs (b), (e), (h) and (k)), the Slim-Tree presents the best results for the average number of disk accesses. Considering only the Omni-family, the OmniR-Tree presents the best results, possibly because the objects were reorganized during the synchronization process. These figures also reflects that synchronization is more important for datasets with smaller objects, as more objects can be stored together. This effect can be seen comparing the *Eigenfaces* and the *Histograms* datasets, where for the smaller objects from the *Eigenfaces* dataset the synchronized OmniR-Tree performs better, while for the *Histograms* the OmniB-Tree wins.

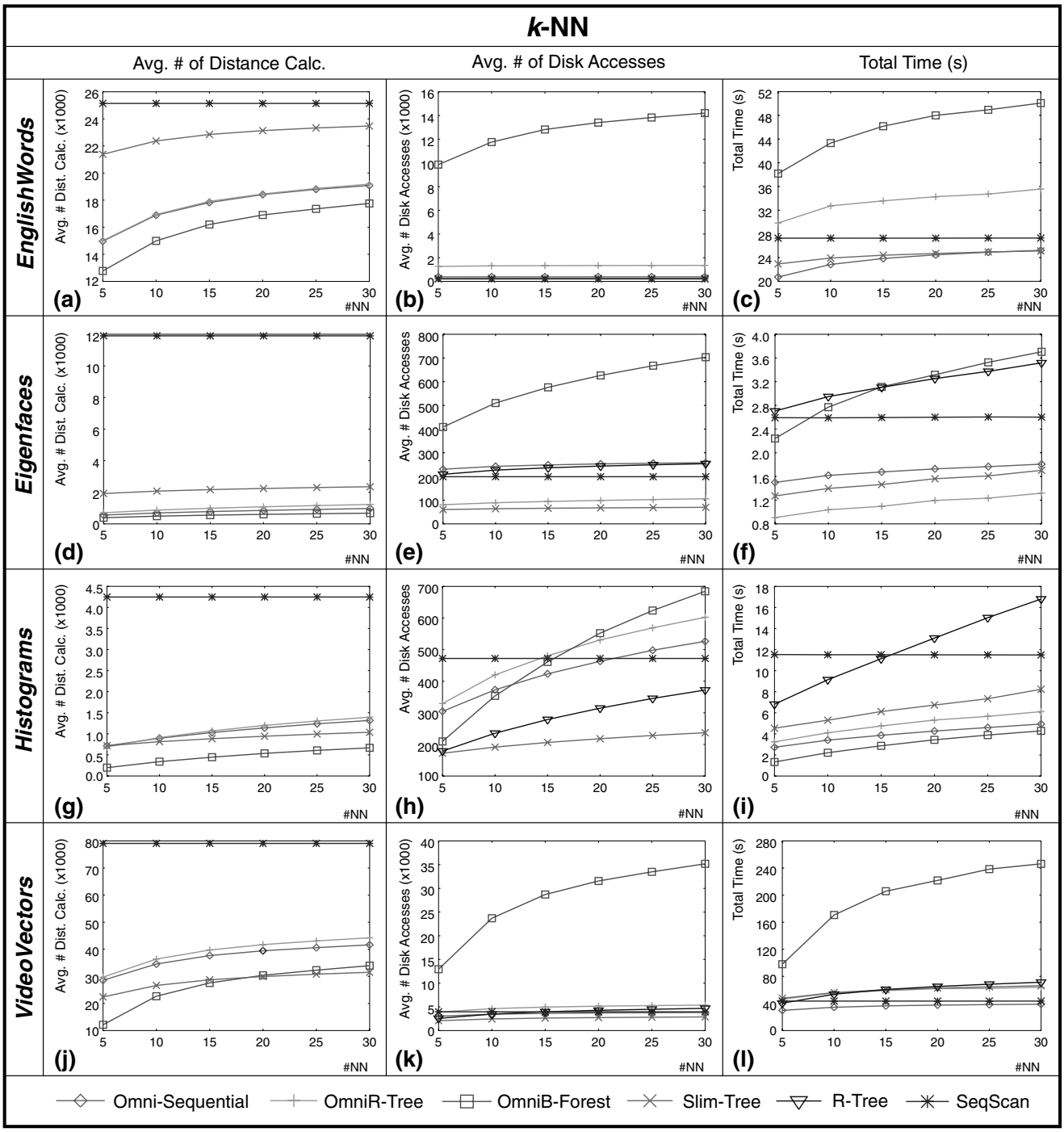

Fig. 15 Results for k-nearest neighbor queries, varying k, comparing: average number of distance calculations in graphs **a**, **d**, **g** and **j**; average number of disk accesses in graphs **b**, **e**, **h** and **k**; and total time in graphs **c**, **f**, **i** and **l**; of Omni-Sequential, OmniR-Tree, OmniB-Forest, Slim-Tree, R-Tree and sequential scan access methods, over the datasets: *EnglishWords* (row 1), *Eigenfaces* (row 2), *Histograms* (row 3) and *VideoVectors* (row 4)

6.3.3 Total time

The relevance on comparing the total time comes from the possibility to analyze, at once, the internal complexity of the algorithms, which are not separately expressed by the number of disk accesses and distance calculations. This is specially true when dealing with the R-Tree, as its operation is not based on distance calculations, but on the comparison of the coordinates of bounding hyper-rectangles. The third column of Figs. 14 and 15 present the total time spent answering range (Fig. 14c, f, i and l) and nearest-neighbor queries (Fig. 15c, f, i and l). These graphs show that the winner is always a member of the Omni-family, where in most of the cases the Omni-Sequential ties or is reason-

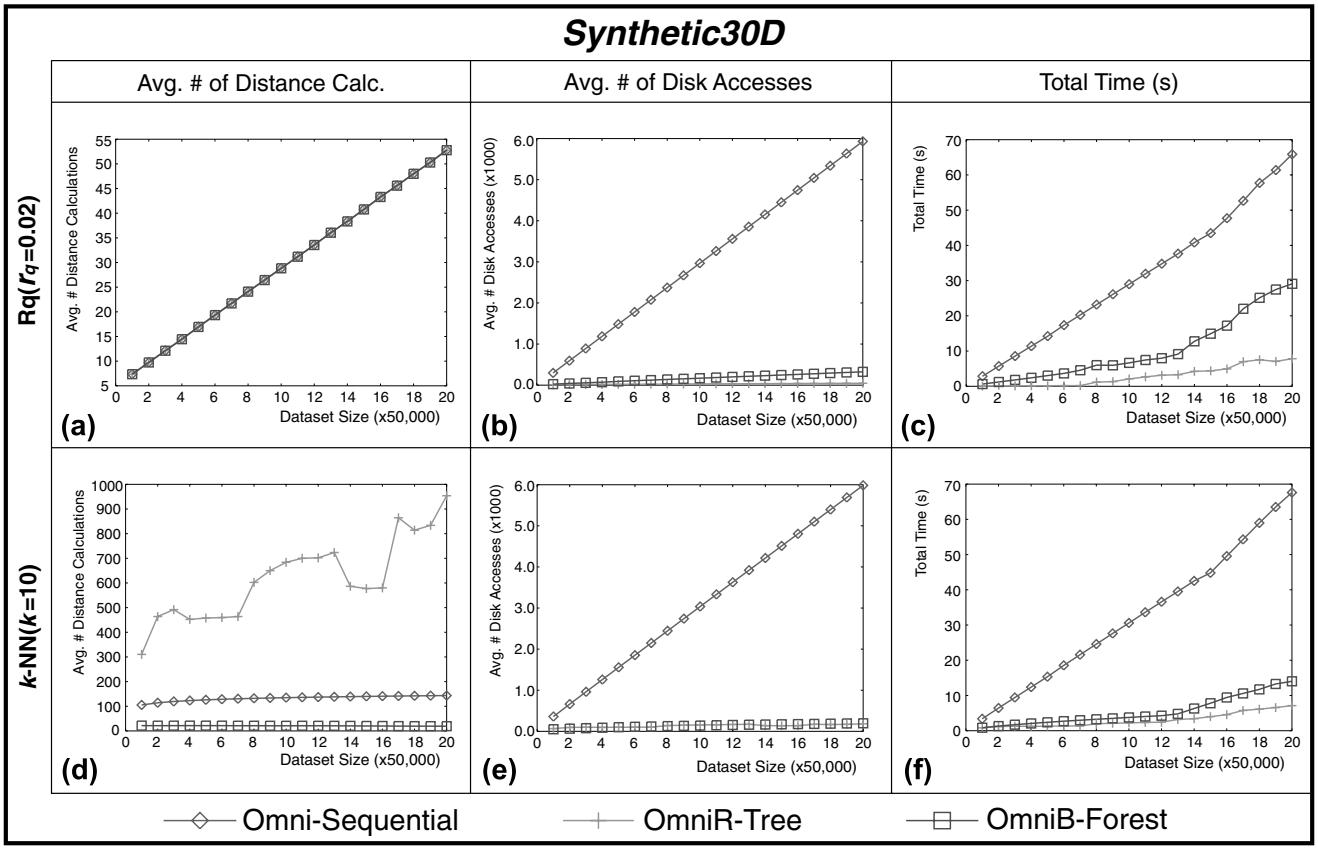

Fig. 16 Scalability of Omni-Sequential, OmniR-Tree and OmniB-Forest for Range query (graphs **a**, **b**, **c**) and for k-Nearest neighbor query (graphs **d**, **e**, **f**) with the *Synthetic30D* dataset. **a**, **d** Average of distance calculations. **b**, **e** Average number of disk accesses. **c**, **f** Total time

ably close to the winner and, for the *EnglishWords* dataset, the Omni-Sequential outperforms every other for both range and k-NN queries. For range queries the three members of the Omni-family presented relatively close results, being up to 15 times faster than the other methods. Regarding the time to answer nearest-neighbor queries, the first place is shared by the Omni-Sequential and the OmniR-Tree, while the OmniB-Tree got the first place only for the *Histograms* dataset, reassuring its appropriateness to datasets with high embedding dimension and low intrinsic dimension. Anyway, also for k-NN queries, an Omni-family member always presents the best performance, being up to four times faster than the others.

6.4 Scalability

The Omni-Sequential, OmniR-Tree and OmniB-Forest scales up well regarding the size of the dataset. Figure 16 shows the behavior of these access methods for the average number of distance calculations (graphs (a), (d)), and the average number of disk accesses (graphs (b), (e)), the total time (graphs (c), (f)), for 500 range queries with radius $r_q = 0.02$, (giving a selectivity of almost 19 objects), and 500 k-nearest neighbors with $k = 10$. It must be noted that the Omni-foci bases were chosen using the first ten percentiles of the database, and maintained untouched throughout this experiments. The dataset used is the *Synthetic30D*, which was prepared to not fit in the available main memory, forcing pagination, and to present relatively high embedded dimensionality and low intrinsic dimensionality, resembling these properties of real datasets.

As it can see from Fig. 16, the behavior of all access methods is linear for range queries, and except for an oscillation in the plot for the number of distance calculations (Fig. 16d) in the OmniR-Tree (due to changes in the tree height), the behavior is linear for k-NN queries too, regarding both number of distance calculations and number of disk accesses, considering the size of dataset. The total time graphics shows that the three access methods are also linear up to the point when pagination begins, and thereafter the linear behavior is maintained with a larger slope.

7 Discussion

This Section aims at summarizing on the applicability of each one of the Omni-family members as well as their relevance for the database area, as follows.

– *Omni-Sequential* Despite the sequential aspect of this method, it almost ties and sometimes even outperforms other access methods, being the Omni member that presented the most stable behavior during the experiments. Its paramount property is its simplicity allied to a good performance. As it is mostly a combination of a random access file and a sequential file, this method is adequate for highly dynamic environments, which require constant updates and concurrent accesses;
– *OmniR-Tree* This method presented impressive results for vectorial datasets. We compared the OmniR-Tree and the R-Tree using the same base code for both methods, and the OmniR-Tree performed up to 10 times faster. An important contribution of this method is the algorithm to synchronize the file that stores the objects with the file that stores the Omni-coordinates in the R-Tree. Therefore, the synchronized OmniR-Tree performed 15 times fewer disk accesses, and was eight times faster than the unsynchronized one. Similar synchronization techniques can be applied over other Omni members;
– *OmniB-Forest* What motivated us to develop this method was the fact that all commercial database system implements the B-Tree. This eases tackling other aspects such as concurrence control, and the possibility of employing parallel processing, as the set of B-Trees of the OmniB-Forest can be spread among multiple processors. Another contribution presented by this paper regarding this method is the algorithm for k nearest-neighbor queries, which in every experiment required the lowest number of distance computations, performing up to three times fewer distance computations than the other methods. We are now working to improve the number of disk accesses based on the synchronization algorithm developed for the OmniR-Tree.

8 Conclusion

This paper describes the new Omni-technique, which was developed to speed up the processing of similarity queries by decreasing the number of distance calculations demanded. It shows how this new technique can be used to index metric and spatial datasets employing existing access methods, through the use of Omni-Coordinates. To exemplify the technique, we showed how to index large vectors with B-Trees (even beating the R-Trees) and words (a metric data) with R-Trees. As the Omni-technique improves the insertion and range query algorithms of the existing methods without changes, incorporating the Omni-technique into existing DBMS is not a difficult task. We also mention how to perform k-NN queries with few changes in the range queries algorithms. However, to achieve better results, we have presented specialized k-NN query algorithms for the three members of the Omni-family presented in this paper.

Our experiments performed on representative datasets showed speedups of up to 15 times to answer queries and equivalent results in reducing distance computations and disk accesses. Moreover, it broadens the scope of the underlining access method already used in the commercial DBMS, enabling them to support indexing of 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 number of distance computations; and the *HF*-algorithm to choose the foci, which is linear on the size of the dataset.

The Omni-family members presented in this paper are scalable with respect to dataset size, and exhibited a sublinear behavior regarding time. We believe that the Omni-technique will greatly contribute to improve supporting multimedia data in DBMS.

Acknowledgements This work has been supported by FAPESP (São Paulo State Research Foundation) and CNPq (Brazilian National Council for Supporting Research).

References

1. Aggarwal, C.C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional spaces. In: Proceedings of the 8th International Conference on Database Theory (ICDT). Lecture Notes in Computer Science, vol. 1973, pp. 420–434. Springer (2001).
2. Annamalai, M., Chopra, R., De Fazio, S.: Indexing images inoracle8i. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 539–547. ACM Press (2000)
3. Arantes, A.S., Vieira, M.R., Traina, A.J.M., Traina, C. Jr.: The fractal dimension making similarity queries more efficient. In: Proceedings of the II ACM SIGKDD Workshop on Fractals, Power Laws and Other Next Generation Data Mining Tools, pp. 12–17. Washington, USA (2003)
4. Baeza-Yates, R.A., Cunto, W., Manber, U., Wu, S.: Proximity matching using fixed-queries trees. In: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM). Lecture Notes in Computer Science, vol. 807, pp. 198–212. Springer (1994)
5. Beckmann, N.: Kriegel, H.-P., Schneider, R., Seeger, B.: TheR*-Tree: an efficient and robust access method for points and rectangles. In: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, pp. 322–331. ACM Press (1990)
6. Belussi, A., Faloutsos, C.: Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In: Proceedings of 21th International Conference on Very Large Data Bases (VLDB), pp. 299–310. Morgan Kaufmann (1995)
7. Berman, A., Shapiro, L.G.: Selecting good keys fortriangle-inequality-based pruning algorithms. In: Proceedings of the International Workshop on Content-Based Access of Image and Video Databases (CAIVD), pp. 12–19. IEEE Computer Society (1998)
8. Beyer, K.S., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: Proceedings of the 7th International Conference on Database Theory (ICDT). Lecture Notes in Computer Science. vol. 1540, pp. 217–235. Springer (1999)
9. Bozkaya, T., Özsoyoglu, Z. Meral.: Distance-based indexing for high-dimensional metric spaces. In: Proceedings of the 1997ACM SIGMOD International Conference on Management of Data, pp. 357–368. ACM Press (1997)
10. Bozkaya, T., Özsoyoglu, Z. Meral.: Indexing large metric spaces for similarity search queries. ACM Trans. Database Syst. (TODS) **24**(3), 361–404 (1999)

11. Brin, S.: Near neighbor search in large metric spaces. In: Proceedings of 21th International Conference on Very Large DataBases (VLDB), pp. 574–584. Morgan Kaufmann (1995)
12. Burkhard, W.A., Keller, R.M.: Some approaches to best-match filesearching. Commun. ACM (CACM) **16**(4),230–236 (1973)
13. Camastra, F., Vinciarelli, A.: Intrinsic dimension estimation of data: an approach based on Grassberger-Procaccia's algorithm. Neural. Process. Lett. **14**(1), 27–34 (2001)
14. Chávez, E., Marroquín, J.L., Baeza-Yates, R.A.: Spaghettis: An array based algorithm for similarity queries inmetric spaces. In: Proceeding of the String Processing and Information Retrieval Symposium & International Workshop on Groupware (SPIRE/CRIWG), pp. 38–46. IEEE Computer Society (1999)
15. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surveys **33**(3), 273–321 (2001)
16. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: An efficient access method for similarity search in metric spaces. In: Proceedings of 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece, pp. 426–435. Morgan Kaufmann Publishers (1997)
17. de Sousa, E.P.M., Traina, C. Jr., Traina, A.J.M., Faloutsos, C.: How to use fractal dimension to find correlations between attributes. In: Proceeding of the First Workshop on Fractals and Self-Similarity in Data Mining: Issues and Approaches (in conjunction with 8th ACMSIGKDD International Conference on Knowledge Discovery & DataMining), Edmonton, Alberta, Canada, pp. 26–30. ACM Press (2002)
18. Faloutsos, C., Seeger, B., Traina, A.J.M., Traina, C. Jr.: Spatialjoin selectivity using power laws. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 177–188, Dallas, USA. ACM Press (2000)
19. Faragó, A., Linder, T., Lugosi, G.: Fast nearest-neighbor search in dissimilarity spaces. IEEE Trans. Pattern Anal. Mach. Intell.(TPAMI) **15**(9), 957–962(1993)
20. Fu, Ada Wai-Chee, Chan, Polly Mei Shuen, Cheung, Yin-Ling, Moon, Yiu Sang.: Dynamic vp-Tree indexing for n-nearest neighbor search given pair-wise distances. VLDB J. **9**(2), 154–173 (2000)
21. Gaede, V., Günther, O.: Multi dimensional access methods. ACM Comput. Surveys **30**(2), 170–231 (1998)
22. Gennaro, C., Savino, P., Zezula, P.: A hashed schema forsimilarity search in metric spaces. In: Proceeding of the 1st DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries, pp. 83–88. Zurich, Switzerland (2000)
23. Guttman, A.: R-Tree : A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, USA, pp. 47–57. ACM Press (1984)
24. Hjaltason, G.R., Samet, H.: Index-driven similarity search inmetric spaces. ACM Trans. Database Syst. (TODS) **28**(4), 517–580 (2003)
25. Ishikawa, M., Chen, H., Furuse, K., Yu, Jeffrey Xu, Ohbo, N.: Mb+tree: A dynamically updatable metric index for similarity searches. In: Proceedings of the First International Conference Web-Age Information Management (WAIM). Lecture Notes in Computer Science, vol. 1846, pp. 356–373. Springer (2000)
26. Jin, Hui, Ooi, Beng Chin, Shen, Heng Tao, Yu, Cui, Zhou, Aoying.: An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In: Proceedings of the 19th International Conference on Data Engineering (ICDE), pp. 87–98. IEEE Computer Society (2003)
27. Katayama, N., Satoh, S.: The SR-Tree: An index structure for high-dimensional nearest neighbor queries. In: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, pp. 369–380. ACM Press (1997)
28. Korn, F., Pagel, Bernd-Uwe, Faloutsos, C.: On the 'dimensionality curse' and the 'self-similarity blessing'. IEEE Trans. Knowledge Data Eng. (TKDE) **13**(1), 96–111 (2001)
29. Koudas, N., Ooi, Beng Chin, Shen, Heng Tao, Tung, A.K.H.: Ldc: enabling search by partial distance in a hyper-dimensional space. In: Proceedings of the 20th International Conference on Data Engineering (ICDE), pp. 6–17. IEEE Computer Society (2004)
30. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Cybernet. Control Theory **10**(8), 707–710 (1966)
31. Lin, K.-I., Jagadish, H.V., Faloutsos, C.: The tv-tree: an index structure for high-dimensional data. VLDB J. **3**(4), 517–542 (1994)
32. Micó, L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. Pattern Recog. Lett. **15**(1), 9–17 (1994)
33. Moreno-Seco, F., Micó, L., Oncina, J.: Extending laesa fastnearest neighbour algorithm to find the k nearest neighbours. In: Proceedings of the International Workshop of Structural, Syntactic, and Statistical Pattern Recognition (SSPR), Lecture Notes in Computer Science, vol. 2396, pp. 718–724. Springer(2002)
34. Pagel, B.-U., Korn, F., Faloutsos, C.: Deflating the dimensionality curse using multiple fractal dimensions. In: Proceedings of the 16th International Conference on Data Engineering (ICDE), pp. 589–598. IEEE Computer Society (2000)
35. Santos Filho, R.F., Traina, A.J.M., Traina, C. Jr., Faloutsos, C.: Similarity search without tears: the OMNI family of all-purpose access methods. In: Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, Germany, pp. 623–630. IEEE Computer Society (2001)
36. Schroeder, M.: Fractals, Chaos, Power Laws. W.H. Freeman &Company, New York, USA (1991)
37. Sellis, T.K.: Nick Roussopoulos, and Christos Faloutsos. TheR^+-Tree: A dynamic index for multi-dimensional objects. In: Proceedings of 13th International Conference on Very Large Databases (VLDB), Brighton, England, pp. 507–518. Morgan Kaufmann Publishers (1987)
38. Senior, A.: A combination fingerprint classifier. IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) **23**(10), 1165–1175 (2001)
39. Traina, C., Agma, J.M. Jr., Faloutsos, C.: Distance exponent: a new concept for selectivity estimation in metric trees. In: Proceedings of the 16th International Conference on Data Engineering (ICDE), San Diego - CA, pp. 195. IEEE Computer Society (2000)
40. Traina, A.J.M., Traina, C. Jr., Bueno, Josiane M., de Azevedo Marques, P.M.: The metric histogram: a new and effiretrieval. In: Proceedings of the Sixth IFIP Working Conference on Visual Database Systems (VDB), Brisbane, Australia, pp. 297–311. Kluwer Academic Publishers (2002)
41. Traina, C. Jr., Traina, A.J.M., Faloutsos, C., Seeger, B.: Fast indexing and visualization of metric datasets using slim-Trees. IEEE Trans. Knowledge Data Eng. (TKDE) **14**(2), 244–260 (2002)
42. Traina, C. Jr., Traina, A.J.M., Faloutsos, C.: Distance exponent:a new concept for selectivity estimation in metric trees. Research Paper CMU-CS-99-110, Carnegie Mellon University - School of Computer Science, Pittsburgh-PA USA, March 1999
43. Traina, C. Jr., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-Trees: High performance metric trees minimizing overlap between nodes. In: Proceedings of the International Conference on Extending Database Technology (EDBT). Lecture Notes in Computer Science, vol. 1777, pp. 51–65, Konstanz, Germany. Springer (2000)
44. Traina, C. Jr., Traina, A.J.M., Wu, L., Faloutsos, C.: Fast feature selection using fractal dimension. In: XV Brazilian Database Symposium (SBBD), João Pessoa, Brazil, pp. 158–171 (2000)
45. Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Inform. Process. Lett. **40**(4), 175–179 (1991)
46. Wactlar, H.D., Christel, M.G., Gong, Y., Hauptmann, A.G.: Lessons learned from building a terabyte digital video library. IEEE Comput. **32**(2), 66–73 (1999)
47. Weber R., Schek, H.-J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of 24rd International Conference on Very Large Data Bases (VLDB), pp. 194–205 (1998)

48. White, D.A., Jain, R.: Similarity indexing with the SS-Tree. In: Proceedings of the 12th International Conference on Data Engineering (ICDE), New Orleans, USA, pp. 516–523. IEEE Computer Society (1996)
49. Wilson, D.R., Martinez, T.R.: Improved heterogeneous distance functions. J. Artif. Intell. Res. **6**, 1–34 (1997)
50. Yianilos, P.N.: Data structures and algorithms for nearestneighbor search in general metric spaces. In: Proceedings of the 4th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA), Austin, USA, pp. 311–321 (1993)
51. Yianilos, P.N.: Excluded middle vantage point forests for nearest neighbor search. Research paper, NEC Research Institute, Princeton, NJ, USA, Princeton, USA (1998)
52. Yu, Cui, Ooi, Beng Chin, Tan, Kian-Lee, Jagadish, H.V.: Indexing the distance: an efficient method to knn processing. In: Proceedings of 27th International Conference on Very Large Data Bases (VLDB), pp. 421–430. Morgan Kaufmann (2001)