A Cost Model for Similarity Queries in Metric Spaces *

Paolo Ciaccia
DEIS - CSITE-CNR
Bologna, Italy
pciaccia@deis.unibo.it

Marco Patella
DEIS - CSITE-CNR
Bologna, Italy
mpatella@deis.unibo.it

Pavel Zezula **
IEI-CNR
Pisa, Italy
zezula@iei.pi.cnr.it

Abstract

We consider the problem of estimating CPU (distance computations) and I/O costs for processing range and k-nearest neighbors queries over metric spaces. Unlike the specific case of vector spaces, where information on *data* distribution has been exploited to derive cost models for predicting the performance of multi-dimensional access methods, in a generic metric space there is no such a possibility, which makes the problem quite different and requires a novel approach. We insist that the *distance* distribution of objects can be profitably used to solve the problem, and consequently develop a concrete cost model for the M-tree access method [10]. Our results rely on the assumption that the indexed dataset comes from a metric space which is "homogeneous" enough (in a probabilistic sense) to allow reliable cost estimations even if the distance distribution with respect to a specific query object is unknown. We experimentally validate the model over both real and synthetic datasets, and show how the model can be used to tune the M-tree in order to minimize a combination of CPU and I/O costs. Finally, we sketch how the same approach can be applied to derive a cost model for the *vp*-tree index structure [8].

1 Introduction

In this work we consider the problem of estimating execution costs for processing similarity (range and nearest neighbors) queries over a set of points (objects) drawn from a *metric space* $\mathcal{M} = (\mathcal{U}, d)$, where $\mathcal{U}$ is a value domain and d is a metric – a non-negative and symmetric function which satisfies the triangular inequality ($d(O_i, O_j) \leq d(O_i, O_k) + d(O_k, O_j)$ $\forall O_i, O_j, O_k \in \mathcal{U}$) – which measures the distance (dissimilarity) of points of $\mathcal{U}$.

*This work has been funded by the EC ESPRIT project no. 9141 HERMES, and Italian C.N.R. MIDA. Pavel Zezula has also been supported by Grants GACR No. 102/96/0986 and KONTAKT No. PM96 S028.

**On leave from the CVIS, Technical University, Údolni 19, Brno, Czech Republic.

Metric spaces include multi-dimensional vector spaces, where objects are compared using, say, Euclidean (L_2) or Manhattan (L_1) distance, as well as non-vector spaces, which are growingly encountered in advanced database applications, such as sequence comparison in molecular biology [7], shape matching [15], fingerprint recognition [17], and many other problems which typically occur in multimedia environments. Common to all these cases is the fact that comparing two objects (i.e. computing their distance) is a non-trivial operation, which can take a time of the order of milliseconds and even more.

In order to efficiently support similarity queries over metric spaces, several index structures have been developed so far, including the *vp*-tree [8], the *GNAT* [6], and the *mvp*-tree [4]. Although different in their organizing principles, these indexes share a common *static* nature, and only attempt to reduce the number of distance computations (CPU costs), paying no attention to I/O costs. In contrast, the M-tree [10] nicely combines principles of metric trees with basic features of database access methods, thus resulting in a paged, dynamic, and balanced access structure which aims to minimize both I/O and CPU costs. Although application of metric trees to real world problems has been proved to be effective [6, 5], even on "traditional" vector spaces, no theoretical analysis able to predict and justify their performance results is available nowadays. In other terms, *no cost model for metric trees exists yet*, which makes their applicability to a specific dataset still a matter of chance. For instance, given a large set of keywords extracted from a text, to be compared using the *edit* (Levenshtein) distance – the minimal number of changes (insertions, deletions, substitutions) needed to transform a string into another one – and an input (query) keyword Q, which is the expected (CPU and I/O) cost to retrieve the, say, 20 nearest neighbors of Q, that is, the 20 strings which are "closest" to Q? Being able to answer questions like this is relevant for database design, query processing, and optimization, since it would provide us with the capability of understanding and tuning a metric access method, and will make it possible to apply optimizers' technology to metric query processing too.

In this paper we introduce the first cost model for metric trees. We concentrate on the M-tree, whose basic principles are reviewed in Section 1.1, since it is the only disk-based metric index. The first problem we deal with concerns the absence of a coordinate-based space, which consequently inhibits us to exploit, among other things, information on *data* distribution, which has indeed been the key to develop ef-

fective cost models [16, 12, 20, 19, 3] for multi-dimensional access methods, such as the R-tree [13, 1] (see Section 1.2). To obviate this situation, we take the direction of building our model around the *distance* distribution, F, of pairwise distances between objects, which is the basic property of a metric space for which we can get and maintain statistics. A basic problem in using F (or some statistics of F) concerns what we call the "homogeneity of viewpoints", formally defined in Section 2. Intuitively, F is a "summary" of distance values, which mixes up together the distances each object has with respect to others. In case of (highly) non-homogeneous spaces, the (relative) distance distribution of an object would be markedly different from that of other objects, which could seriously limit predictive capabilities of a cost model based on F. It is therefore a pleasant finding to observe that real, as well as "realistic" synthetic, datasets are highly homogeneous, a fact we exploit in our cost model. Section 3 introduces the model, called N-MCM (Node-based Metric Cost Model), which can accurately predict both I/O and CPU expected costs for range and nearest neighbors queries. The problem with N-MCM is the amount of information about the tree to be kept, which is proportional to the number of index nodes. To obviate this, a simplified model, L-MCM (L for Level), is then introduced, which only uses average statistical information on the tree structure, collected on a per-level basis. Experimental results in Section 4 show that L-MCM still serves its purpose quite well. We also show, in Section 4.1, how our cost model(s) can be effectively used to tune the M-tree, namely how an optimal node size, depending on the specific dataset, can be chosen to minimize a combination of CPU and I/O costs. In Section 5 we discuss how a cost model to predict *vp*-tree query performance, based on the same assumptions of the M-tree models, could be derived. Several problems raised by our approach are finally discussed in Section 6.

1.1 The M-tree

The M-tree [10] can index objects from a generic metric space $\mathcal{M} = (\mathcal{U}, d)$, but, besides this peculiarity, it shares many similarities with R-trees and other spatial access methods, since it adheres to the GiST framework [14], which specifies a common software kernel for developing database indexes.

Given a set of objects $\mathcal{O} = \{O_1, \ldots, O_n\}$, $\mathcal{O} \subseteq \mathcal{U}$, the M-tree stores them into fixed-size leaf nodes, which correspond to regions (*balls*) of the metric space. Each entry in a leaf node has the format $[O_i, \text{oid}(O_i)]$, where $O_i \in \mathcal{O}$ and $\text{oid}(O_i)$ is a pointer to the corresponding description of O_i. Internal (non-leaf) nodes store entries with format $[O_r, r(N_r), \text{ptr}(N_r)]$, where O_r is a so-called *routing point*, $r(N_r) > 0$ is a *covering radius*, and $\text{ptr}(N_r)$ is a pointer to the child node N_r. The basic property of the covering radius is that *each object O_i in the sub-tree rooted at N_r satisfies the constraint $d(O_r, O_i) \leq r(N_r)$*, that is, all the objects O_i reachable from node N_r are in the ball of radius $r(N_r)$ centered in O_r. Clearly, the actual "shape" of such balls depends on the specific metric space $(\mathcal{U}, d)$. For instance, balls are "diamonds" in $(\Re^2, L_1)$, circles in $(\Re^2, L_2)$, and squares in $(\Re^2, L_\infty)$. From an overall point of view, it can be seen that the M-tree organizes the database objects into a set of, possibly overlapping, balls, to which the same principle is recursively applied up to the root of the tree.

Similarity queries supported by the M-tree are of two basic types: *range* queries, denoted $\texttt{range}(Q, r_Q)$, where, given a query (reference) object $Q \in \mathcal{U}$, all the objects whose distance from Q does not exceed r_Q are selected; *k-nearest neighbors* queries, $\texttt{NN}(Q, k)$, where the k ($k \geq 1$) closest objects to Q have to be retrieved, with ties arbitrarily broken. The `NN` search algorithm implemented in the M-tree is *optimal*, in the sense of [3], since it only accesses those nodes whose region intersects the $\texttt{NN}(Q, k)$ ball, that is, the ball centered in Q with radius given by the distance between Q and its k-th nearest neighbor.

1.2 Related Work

No specific work has addressed yet the problem of estimating costs for queries over generic metric spaces. Nonetheless, it is useful to review what has been done in recent years for the case of vector (Cartesian) spaces, which are a subset of metric spaces, with a major focus on the performance of R-tree and its variants.

In [16] the authors present a model to estimate I/O costs for range queries as a function of the geometric characteristics of R-tree's nodes, on the assumption that queries are rectangles uniformly distributed in the space. Extension to *fractal datasets* is presented in [12], assuming that nodes are square-shaped, and showing that relative errors are rarely above 12%. The estimation of the fractal dimension of a dataset is based on a box-counting algorithm which subdivides the indexed space into hyper-cubic grid cells. Extension of the model to predict selectivities of range queries and spatial joins under the *biased* query model (queries are more likely in high-density areas of the space) is presented in [2]. A model for predicting I/O costs of range queries, without knowledge of R-tree characteristics, is introduced in [20]. The model exploits statistical information on data distribution, represented in the form of a *density* histogram, and exhibits relative errors around 5%-15% for 2-dimensional range queries.

Performance of nearest neighbors queries, for the case $k = 1$, has been considered in [19] and [3]. In [19] the analysis focuses on 2-dimensional Euclidean spaces and derives lower and upper bounds for I/O costs by exploiting the notion of fractal dimension and assuming that query points belong to the dataset. Finally, [3] introduces a cost model for nearest neighbor search in high-dimensional Euclidean spaces. The model estimates the expected NN distance and the number of accessed index pages under the assumption of a uniform distribution of points.

Above models either strongly rely on the Cartesian nature of the space and on knowledge of the data distribution or exploit the concept of fractal dimension. Since this is indeed a *metric* concept [18, page 357], it would in principle be possible to apply it to generic metric spaces too, and we leave it as an interesting topic for future research.

2 The Distance Distribution

In this work we pursue an approach which does not rely on data distribution, rather it takes a more general view, able to deal whith generic metric spaces, which can be characterized by the following positions:

1. The only information derivable from the analysis of a metric dataset is (an estimate of) the *distance* distribution of objects. No information on *data* distribution is used.

2. A *biased* query model is considered, with query objects which do not necessarily belong to the indexed dataset.

We claim that the distance distribution is the correct counterpart of data distribution used for vector spaces, since it is the "natural" way to characterize a metric dataset. As to the biased query model, which apparently contradicts the position that no knowledge of the data distribution is assumed, we clarify this point by precisely defining our working scenario.

We consider *bounded random metric* (BRM) spaces, $\mathcal{M} = (\mathcal{U}, d, d^+, S)$, where $\mathcal{U}$ and d are as usual, d^+ is a finite upper bound on distance values, and S is a measure of probability over (a suitable Borel field defined on) $\mathcal{U}$. To help intuition, we slightly abuse terminology and call S the *data* distribution over $\mathcal{U}$. Although S has no specific role in our cost model (we do not need to know S and never use it), its existence is postulated both for formal reasons and to account for the nature of the observed datasets (see Section 2.1).

For instance, $([0,1]^D, L_2, \sqrt{D}, U([0,1]^D))$ is the BRM space characterized by a uniform distribution of points over the D-dimensional unit hypercube, and where distance between points is measured by the Euclidean (L_2) metric. As another example, $(\Sigma^m, L_{edit}, m, S)$ is the BRM space whose domain is the set of all the strings of length up to m over the Σ alphabet, whose distance is measured by the *edit* (Levenshtein) metric, and with data distribution S, here left unspecified.

The (overall) distribution of distances over $\mathcal{U}$ is defined as:

$$F(x) = \Pr\{d(\mathbf{O_1}, \mathbf{O_2}) \leq x\} \tag{1}$$

where $\mathbf{O_1}$ and $\mathbf{O_2}$ are two (independent) S-distributed random points of $\mathcal{U}$. For each $O_i \in \mathcal{U}$, the *relative* distance distribution, RDD, of O_i is obtained by simply setting $\mathbf{O_1} = O_i$ in above expression, i.e:

$$F_{O_i}(x) = \Pr\{d(O_i, \mathbf{O_2}) \leq x\} \tag{2}$$

For what follows it is useful to regard F_{O_i} as the O_i's "viewpoint" of the $\mathcal{U}$ domain, as determined by d and S. The fact that different objects can have different viewpoints is the general rule. Even for a uniform distribution over a bounded Cartesian domain, the center's viewpoint is not the same as the viewpoint an object close to the boundary has. In order to measure how much two viewpoints are (dis-)similar, we introduce the concept of *discrepancy*.

Definition 1 *The discrepancy of the two* RDDs F_{O_i} *and* F_{O_j} *is defined as:*

$$\delta(F_{O_i}, F_{O_j}) = \frac{1}{d^+} \int_0^{d^+} \|F_{O_i}(x) - F_{O_j}(x)\| \, dx \tag{3}$$

The discrepancy of any pair of RDDs is a real number in the unit interval $[0,1]$, and equals 0 iff F_{O_i} and F_{O_j} are the same (which does not imply, however, that $O_i \equiv O_j$). Note that δ is a metric on the functional space $\mathcal{F} = \{F_{O_i} : O_i \in \mathcal{U}\}$, as it can be easily verified.

Consider now the case where both $\mathbf{O_1}$ and $\mathbf{O_2}$ are random points. Under this view, $\mathbf{\Delta} \stackrel{\text{def}}{=} \delta(F_{\mathbf{O_1}}, F_{\mathbf{O_2}})$ is a random variable, and $G_{\mathbf{\Delta}}(y) = \Pr\{\mathbf{\Delta} \leq y\}$ is the probability that the discrepancy of two (random) RDDs is not larger than y. The higher, for a given y, $G_{\mathbf{\Delta}}(y)$ is, the more likely is that two RDDs F_{O_i} and F_{O_j} "behave" the same, up to an y level of discrepancy. Therefore, if $G_{\mathbf{\Delta}}(y) \approx 1$ for a "low" y value, we can say that the BRM space $\mathcal{M}$ from which G is derived is somewhat "homogeneous" as to the viewpoints that objects in $\mathcal{U}$, weighted according to the S distribution, have. This observation is captured by introducing an index of homogeneity for BRM spaces.

Definition 2 *The index of "Homogeneity of Viewpoints" of a* BRM *space* $\mathcal{M}$ *is defined as:*

$$HV(\mathcal{M}) = \int_0^1 G_{\mathbf{\Delta}}(y)dy = 1 - E[\mathbf{\Delta}] \tag{4}$$

Example 1 Consider the BRM space $\mathcal{M} = (\{0,1\}^D \cup \{(0.5, \ldots, 0.5)\}, L_\infty, 1, U)$, where the domain is the D-dimensional binary hypercube extended with the "midpoint" $C = (0.5, \ldots, 0.5)$, points are uniformly distributed, and $L_\infty(O_i, O_j) = \max_k\{\|O_i[k] - O_j[k]\|\}$. For all $O_i, O_j \in \{0,1\}^D$ it is $\delta(F_{O_i}, F_{O_j}) = 0$, whereas $\delta(F_{O_i}, F_C) = 1/2 - 1/(2^D+1)$. It follows that $G_{\mathbf{\Delta}}(y) = (2^{2D}+1)/(2^D+1)^2$ for $0 \leq y < 1/2 - 1/(2^D+1)$, and $G_{\mathbf{\Delta}}(y) = 1$ for $1/2 - 1/(2^D+1) \leq y \leq 1$. Therefore

$$HV(\mathcal{M}) = 1 - \frac{2^{2D} - 2^D}{\left(2^D+1\right)^3} \stackrel{D\to\infty}{\longrightarrow} 1$$

For instance, when $D = 10$, it is $HV(\mathcal{M}) \approx 1 - 0.97 \times 10^{-3} \approx 0.999$. Since all points but C have the same view of $\mathcal{M}$, even for moderately large values of D the presence of C has a negligible effect on $HV(\mathcal{M})$. □

The relevance of HV lies in the fact that, with a single value, we can characterize the whole BRM space with respect to how objects "see" such a space. The higher $HV(\mathcal{M})$ is, the more two random points are likely to have an almost common view of $\mathcal{M}$. Therefore, $HV(\mathcal{M}) \approx 1$ denotes high homogeneity in the $\mathcal{M}$ space as to objects' distance distributions (but not necessarily with respect to data distribution!). As we will show in the next section, HV can be high even for real datasets.

2.1 Dealing with Database Instances

A database *instance* $\mathcal{O} = \{O_1, \ldots, O_n\}$ of $\mathcal{U}$ is, according to our view, an n-sized sample of objects, selected according to the (unknown) S data distribution over $\mathcal{U}$. From this sample, the basic information we can derive about the BRM space $\mathcal{M}$ is an estimate of F, denoted $\widehat{F^n}$, represented by the $n \times n$ matrix of pairwise distances between objects in $\mathcal{O}$.

For estimating the cost of a query (either range or nearest neighbors), the best thing would be to know F_Q, i.e. the RDD of the query object itself. However, in the general case this is a hopeless alternative, since Q is not restricted to belong to $\mathcal{O}$. What if we use $\widehat{F^n}$ in place of F_Q? As long as the two distributions behave almost the same, we do not expect relevant estimate errors from *any* cost model we could devise, provided the model correctly uses $\widehat{F^n}$.

In order to verify the above possibility, we computed the HV index for several synthetic and real datasets, summarized in Table 1. The `clustered` datasets consist of D-dimensional vectors normally-distributed (with σ=0.1) in 10 clusters over the unit hypercube. Text datasets in Table 1 are sets of keywords extracted from 5 masterpieces of Italian literature. For all these datasets we observed HV values always above 0.98, which justifies the following assumption we make for deriving our cost model:

Name	Description	Size	Dim. (D)	Metric
`clustered`	clustered distr. points on $[0,1]^D$	$10^4 - 10^5$	$5 - 50$	L_∞
`uniform`	uniform distr. points on $[0,1]^D$	$10^4 - 10^5$	$5 - 50$	L_∞
`D`	Decamerone	17,936		edit
`DC`	Divina Commedia	12,701		edit
`GL`	Gerusalemme Liberata	11,973		edit
`OF`	Orlando Furioso	18,719		edit
`PS`	Promessi Sposi	19,846		edit

Table 1: Datasets

Assumption 1 *The homogeneity of viewpoints index, HV, is "high" (close to 1), and the relative distance distribution of a query object Q is well approximated by the sampled $\widehat{F^n}$ distance distribution.*

The second part follows from the facts that: (a) if objects' RDDs almost behave the same, so will do their average, that is $\widehat{F^n}$, and (b) we assume a biased query model.

3 The Cost Models for M-tree

Symbol	Description
n	number of indexed objects
$F(x)$	distance distribution
$f(x)$	distance density function
Q	query object
r_Q	query radius
k	number of nearest neighbors
M	number of nodes in the M-tree
O_r	routing object
$r(N_r)$	covering radius of node N_r
$e(N_r)$	number of entries in node N_r
L	height of the M-tree
M_l	number of nodes at level l of the M-tree
$\overline{r_l}$	average covering radius of nodes at level l

Table 2: Summary of symbols and respective definitions

In this Section we present two different models: the Node-based Metric Cost Model (N-MCM) makes use of statistics for each node of the tree, while the simplified Level-based model (L-MCM) exploits only statistics collected on a per-level basis. Relevant symbols and their descriptions are given in Table 2.

3.1 The Node-based Metric Cost Model

Consider a range query $\texttt{range}(Q, r_Q)$. A node N_r of the M-tree has to be accessed iff the ball of radius r_Q centered in the query object Q and the region associated with N_r intersect. This is the case iff $d(Q, O_r) \leq r(N_r) + r_Q$, as it can be derived by triangular inequality, which requires that the distance between the two "centers" is not greater than the sum of the two radii. The probability that N_r has to be accessed can therefore be expressed as:[1]

$$\begin{aligned} &\Pr\{\text{node } N_r \text{ is accessed}\} = \\ &= \Pr\{d(Q,\mathbf{O}) \leq r(N_r) + r_Q\} = \\ &= F_Q(r(N_r) + r_Q) \approx F(r(N_r) + r_Q) \end{aligned} \tag{5}$$

where the uncertainty is due to the position in the space of the routing object of N_r, here considered to be a random point, and the approximation exploits Assumption 1. To determine the expected I/O cost for a range query is sufficient to sum above probabilities over all the M nodes of the tree:

$$nodes(\texttt{range}(Q, r_Q)) = \sum_{i=1}^{M} F(r(N_{r_i}) + r_Q) \tag{6}$$

The number of distance computations (CPU cost) is estimated by considering the probability that a page is accessed multiplied by the number of its entries, $e(N_{r_i})$, thus obtaining:[2]

$$dists(\texttt{range}(Q, r_Q)) = \sum_{i=1}^{M} e(N_{r_i})\, F(r(N_{r_i}) + r_Q) \tag{7}$$

Finally, the expected number of retrieved objects is estimated as:

$$objs(\texttt{range}(Q, r_Q)) = n \cdot F(r_Q) \tag{8}$$

Let us now consider the case of a nearest neighbors query, $\texttt{NN}(Q, k)$, on the assumption that $k < n$. As a first step we determine the expected distance between the query object and its k-th nearest neighbor, which depends on the distance distribution F. Let $\mathbf{nn}_{Q,k}$ be the RV standing for the distance of the k-th nearest neighbor of Q. The probability that $\mathbf{nn}_{Q,k}$ is at most r equals the probability that at least k objects are inside the ball of radius r centered in Q, that is:

$$\begin{aligned} P_{Q,k}(r) &\stackrel{\text{def}}{=} \Pr\{\mathbf{nn}_{Q,k} \leq r\} = \\ &= \sum_{i=k}^{n} \binom{n}{i} \Pr\{d(Q,\mathbf{O}) \leq r\}^i \Pr\{d(Q,\mathbf{O}) > r\}^{n-i} = \\ &= 1 - \sum_{i=0}^{k-1} \binom{n}{i} F(r)^i (1 - F(r))^{n-i} \end{aligned} \tag{9}$$

The density function $p_{Q,k}(r)$ is obtained by taking the derivative of $P_{Q,k}(r)$:

$$\begin{aligned} p_{Q,k}(r) &= \frac{d}{dr} P_{Q,k}(r) = \\ &= \sum_{i=0}^{k-1} \binom{n}{i} F(r)^{i-1} f(r) (1 - F(r))^{n-i-1} (nF(r) - i) \end{aligned} \tag{10}$$

[1] The radius $r(N_r)$ is not defined for the root node. To obviate this we assume that the root has radius $r(N_{root}) = d^+$.

[2] The optimization strategies described in [10] for reducing the number of distance computations are not considered here, and their inclusion in the cost model is left as a subject for future research.

and the expected k-th nearest neighbor distance is computed by integrating $p_{Q,k}(r)$ over all r values:

$$\begin{aligned} E[\mathbf{nn}_{Q,k}] &= \int_0^{d^+} r \cdot p_{Q,k}(r)\ dr = \\ &= |r\, P_{Q,k}(r)|_0^{d^+} - \int_0^{d^+} P_{Q,k}(r)\ dr = \\ &= d^+ - \int_0^{d^+} P_{Q,k}(r)\ dr \end{aligned} \quad (11)$$

For $k = 1$, above results simplify as follows:

$$P_{Q,1}(r) = 1 - (1 - F(r))^n \quad (12)$$

$$p_{Q,1}(r) = \frac{d}{dr} P_{Q,1}(r) = n\, f(r)\, (1 - F(r))^{n-1} \quad (13)$$

$$\begin{aligned} E[\mathbf{nn}_{Q,1}] &= \int_0^{d^+} r \cdot p_{Q,1}(r)\ dr = \\ &= \int_0^{d^+} (1 - F(r))^n\ dr \end{aligned} \quad (14)$$

and reduce to those derived in [3] for vector spaces and Euclidean distance. We remark, however, that above formulas are suitable for generic metric spaces, since they do not require the computation of any "Cartesian volume", as done in [3].

The expected number of page reads can now be obtained by integrating Eq. 6 over all radius values, each value weighted by its probability to occur, as given by Eq. 10. For the case $k = 1$ we obtain

$$nodes(\mathtt{NN}(Q,1)) = \int_0^{d^+} nodes(\mathtt{range}(Q,r))\, p_{Q,1}(r)\ dr =$$

$$= \int_0^{d^+} \sum_{i=1}^{M} F(r(N_{r_i} + r)\ n\ f(r)\, (1 - F(r))^{n-1}\ dr$$

The same principle is applied to determine the expected number of computed distances, for which the number of entries in each node has to be considered:

$$dists(\mathtt{NN}(Q,1)) = \int_0^{d^+} dists(\mathtt{range}(Q,r))\, p_{Q,1}(r)\ dr =$$

$$= \int_0^{d^+} \sum_{i=1}^{M} e(N_{r_i})\ F(r(N_{r_i} + r))\ n\ f(r)\, (1 - F(r))^{n-1}\ dr$$

3.2 The Level-based Metric Cost Model

The basic problem with N-MCM is that mantaining statistics for every node of the tree requires $O(M) = O(n)$ space and the computation of expected values has the same complexity, thus can be very time consuming when the index is (very) large. To obviate this, we consider a simplified model, called L-MCM, which uses only average information collected for each level of the M-tree. This will intuitively lead to a lower accuracy with respect to N-MCM, but, as shown in Section 4, estimates are still accurate enough.

For each level l of the tree ($l = 1, \ldots, L$, with the root at level 1 and leaves at level L), L-MCM just uses two information: M_l (the number of nodes at level l), and $\overline{r_l}$ (the average value of the covering radius considering all the nodes at level l). Given these statistics, and referring to Eq. 6, the number of pages accessed by a range query can be estimated as:

$$nodes(\mathtt{range}(Q, r_Q)) \approx \sum_{l=1}^{L} M_l\ F(\overline{r_l} + r_Q) \quad (15)$$

Similarly, we can estimate CPU costs as:

$$dists(\mathtt{range}(Q, r_Q)) \approx \sum_{l=1}^{L} M_{l+1}\ F(\overline{r_l} + r_Q) \quad (16)$$

where $M_{L+1} \stackrel{\text{def}}{=} n$ is the number of indexed objects. Compared to Eq. 7, we have exploited the simple observation that the number of nodes at a given level equals the number of entries at the next upper level of the tree.

Correspondingly, I/O and CPU costs for a $\mathtt{NN}(Q, 1)$ query are estimated as follows:

$$nodes(\mathtt{NN}(Q,1)) \approx \quad (17)$$

$$\approx \int_0^{d^+} \sum_{l=1}^{L} M_l\ F(\overline{r_l} + r)\ n\ f(r)\ (1 - F(r))^{n-1}\ dr$$

$$dists(\mathtt{NN}(Q,1)) \approx \quad (18)$$

$$\approx \int_0^{d^+} \sum_{l=1}^{L} M_{l+1}\ F(\overline{r_l} + r)\ n\ f(r)\, (1 - F(r))^{n-1}\ dr$$

4 Experimental Evaluation

In order to evaluate the accuracy of our cost models, we ran several experiments on both synthetic and real datasets, as described in Table 1. Estimates were compared with actual results obtained by the M-tree, which was built using the `BulkLoading` algorithm described in [9] with a node size of 4 Kbytes and a minimum node utilization of 30%.

The first set of experiments concerns the `clustered` datasets, and investigates accuracy of estimates as a function of the dimensionality D of the space. The distance distribution is approximated by an *equi-width* histogram with 100 bins, respectively storing the values of $\widehat{F^n}(0.01)$, $\widehat{F^n}(0.02)$, and so on.

Figures 1 (a) and 1 (b) show estimated and real (averaged over 1000 queries) CPU and I/O costs, respectively, for range queries with radius $\sqrt[D]{0.01}/2$. It can be seen that N-MCM is very accurate, with a maximum relative error of 4%, while the performance of L-MCM is worse yet still good (with error below 10%). Figure 1 (c) shows that also query selectivity is well estimated, with errors never exceeding 3%.

Analysis of nearest neighbors queries is presented in Figure 2 for the case $k = 1$. Actual costs (averaged over 1000 queries) are contrasted with those estimated by three different models:

1. The L-MCM (Equations 18 and 19);
2. The costs of a range query, $\mathtt{range}(Q, E[\mathbf{nn}_{Q,1}])$, with radius equal to the expected NN distance (Eq. 14);
3. The costs of a range query with a radius such that the expected number of retrieved objects is at least 1, that is, $\mathtt{range}(Q, r(1))$, $r(1) = \min\{r : n \cdot F(r) \geq 1\}$ (Eq. 8).

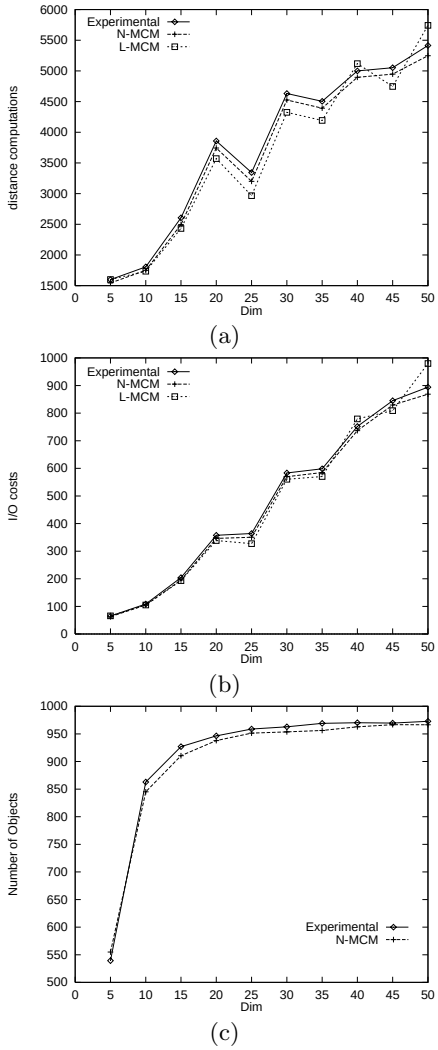


Figure 1: Estimated and real CPU (a) and I/O (b) costs for range queries $\texttt{range}(Q, \sqrt[D]{0.01}/2)$ as a function of the space dimensionality D; (c): Estimated and real cardinality of the result

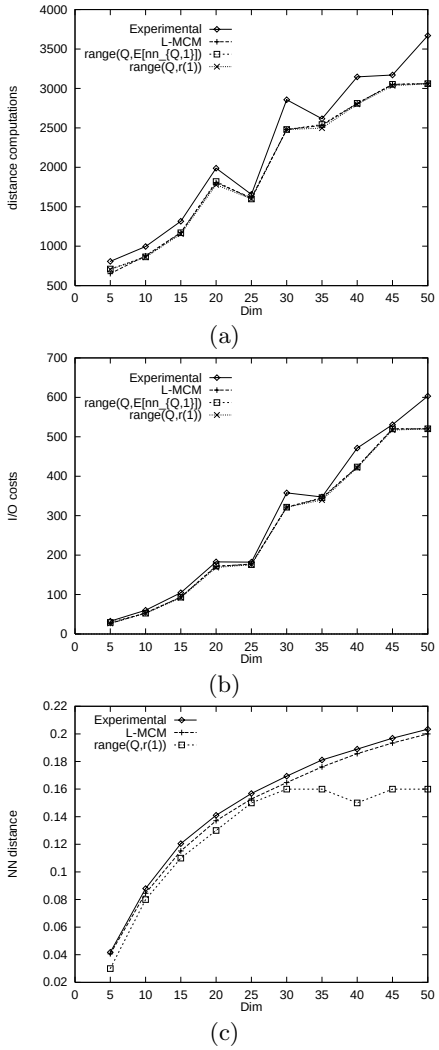


Figure 2: Estimated and real CPU (a) and I/O (b) costs for nearest neighbors queries $\texttt{NN}(Q, 1)$ as a function of the space dimensionality D; (c): Estimated and real distance of the nearest neighbor

Figures 2 (a) and 2 (b) demonstrate that cost estimates are highly reliable, even if errors are higher with respect to the range queries case. Figure 2 (c), showing actual and estimated NN distances, points out how the model based on $r(1)$ can lead to high errors for high D values, which is mainly due to the approximation introduced by the histogram representation.

Experiments on the real datasets of text keywords (see Table 1) were based on 25-bins histograms, since 25 was the maximum observed edit distance. Figures 3 (a) and 3 (b) compare the analitically predicted CPU and I/O costs, respectively, with those experimentally obtained for 1000 range queries with radius 3. Relative errors are usually below 10% and rarely reach 15%.

Finally, Figures 4 (a) and 4 (b) compare estimated and real costs for range queries over the `clustered` dataset with $D = 20$ and a variable query radius.

4.1 Tuning the M-Tree

The cost model(s) can be exploited to tune the M-tree design, by helping to choose a "right" node size which minimizes a combination of I/O and CPU costs. Unlike the case where I/O costs are the dominant factor, and for which the best choice would be to have a node size equal to the maximum size of a block readable from disk, in our scenario distance computations can have a not negligible weight in determining the overall costs. This suggests that increasing the node size can have a negative effect on CPU costs. Figure 5 (a) shows the number of node accesses and of distance computations predicted by the N-MCM for range queries of radius $\sqrt[5]{0.01}/2$ in the 5-dimensional hypercube, where the M-tree indexes 10^6 `clustered` objects, with increasing node sizes in the range $[0.5, 64]$ Kbytes. As predicted, I/O costs are decreasing for increasing node sizes, whereas CPU costs present a marked minimum. Suppose that the cost for a distance computation is c_{CPU} and the cost for a disk access

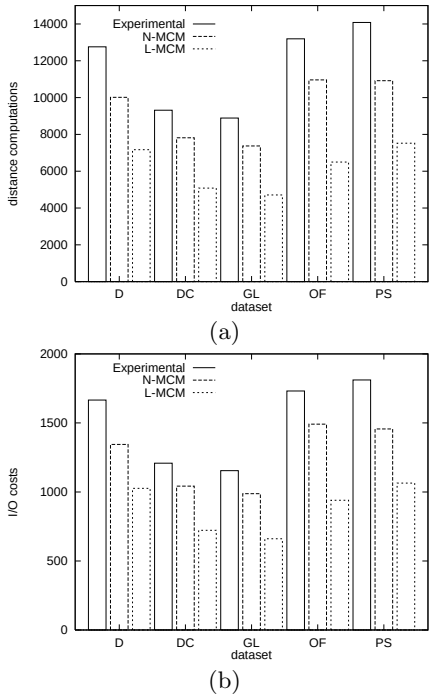


Figure 3: Estimated and real CPU (a) and I/O (b) costs for range queries for each text dataset

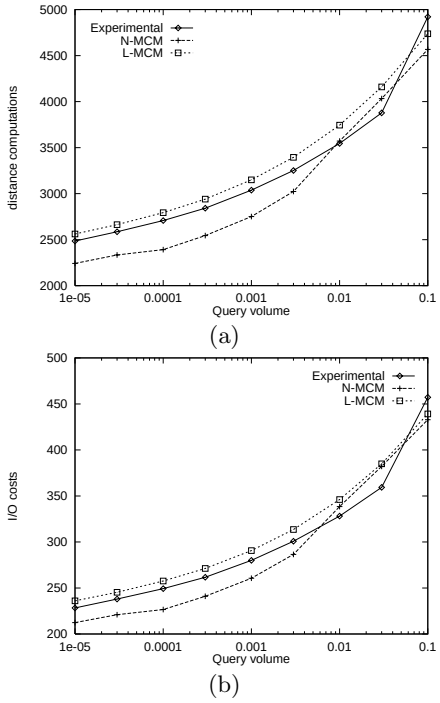


Figure 4: Estimated and real CPU (a) and I/O (b) costs for range queries as a function of the query volume

is $c_{I/O} = t_{pos} + NS \cdot t_{trans}$, where t_{pos} is the disk positioning time and t_{trans} is the transfer time for 1 KB: the optimal choice is then to have a node size NS for which $c_{CPU} \cdot dists(Q; NS) + c_{I/O} \cdot nodes(Q; NS)$ is minimum. In Figure 5 (b) we show estimated and real costs for the case where $c_{I/O} = (10 + NS \cdot 1)$ msecs and $c_{CPU} = 5$ msecs, which leads to an optimal node size of 8 KB.

5 The Case of *vp*-trees

In order to show how our approach could be extended to other metric trees, in this Section we discuss how a cost model for range queries over *vp*-trees [8] could be derived. To this end, we consider the same basic principles used for M-tree, that is:

- The distance distribution is known.
- The *biased* query model is used.
- The homogeneity of viewpoints is high.

The *vp*-tree partitions the data space using spherical cuts around so-called *vantage points*. In a binary *vp*-tree, each internal node has the format $[O_v, \mu, ptr_l, ptr_r]$, where O_v is the vantage point (i.e. an object of the dataset promoted using a specific algorithm), μ is (an estimate of) the median of the distances between O_v and all the objects reachable from the node, and ptr_l and ptr_r are pointers to the left and right child, respectively. The left child of the node indexes the objects whose distance from O_v is less than or equal to μ, while the right child indexes the objects whose distance from O_v is greater than μ. The same principle is recursively applied to the lower levels of the tree, leading to an almost balanced index.

The binary *vp*-tree can be easily generalized into a multiway *vp*-tree, having nodes with a higher fanout, by partitioning the distance values between the vantage point and the data objects into m groups of equal cardinality. Then, the distance values $\mu_1, \ldots, \mu_{m-1}$ used to partition the set are stored in each node, replacing the median value μ. Such values are referred to as *cutoff* values. Figure 6 shows a node of a 5-way *vp*-tree.

Now, consider a range query $\texttt{range}(Q, r_Q)$ on an m-way *vp*-tree.[3] Starting from the root, the system computes the distance between the query object Q and the vantage point O_v, then descends only those branches whose region intersects the query region. Thus, the i-th child of the root, N_{r_i}, has to be accessed, and the distance between the corresponding vantage point and Q computed,[4] iff $\mu_{i-1} - r_Q < d(Q, O_v) \leq \mu_i + r_Q, (i = 1, \ldots, m$, where $\mu_0 = 0$ and $\mu_m = d^+)$. Thus, the probability that N_{r_i} has to be accessed is:

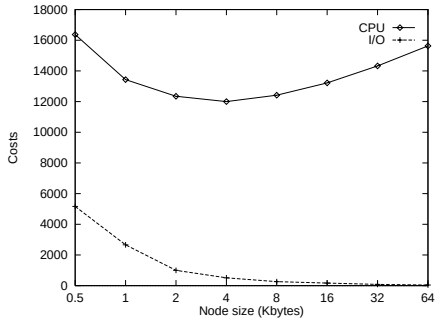
$$\begin{aligned} &\Pr\{N_{r_i} \text{ accessed}\} = \\ &= \Pr\{\mu_{i-1} - r_Q < d(Q, \mathbf{O_v}) \leq \mu_i + r_Q\} = \\ &= F_Q(\mu_i + r_Q) - F_Q(\mu_{i-1} - r_Q) \approx \\ &\approx F(\mu_i + r_Q) - F(\mu_{i-1} - r_Q) \end{aligned} \tag{19}$$

where, as in Eq. 5, the uncertainty is due to the position of the vantage point and the approximation relies on Assumption 1. Figure 7 shows the probability to access the second child of the root in a 3-way *vp*-tree.

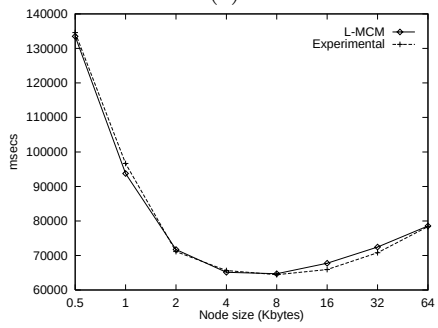
The homogeneity assumption also allows us to estimate the cutoff values without actually building the tree. In fact,

[3] The extension to nearest neighbors queries follows the same principles and is not presented here for the sake of brevity.

[4] Since the *vp*-tree is not paged, in the following we will assume that the index is stored in main memory, thus ignoring I/O costs.



(a)



(b)

Figure 5: (a) Predicted I/O and CPU costs for variable node size. (b) Overall estimated and real costs (in msecs)

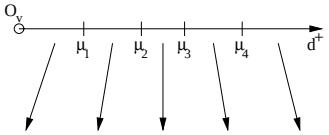


Figure 6: A node of a 5-way *vp*-tree

each μ_i can be estimated as the i/m quantile of $F()$, that is $F^{-1}(i/m)$.[5] It follows that Eq. 19 can be rewritten as:

$$\begin{aligned} &\Pr\{N_{r_i} \text{ accessed}\} \approx \\ &\quad \approx \quad F(F^{-1}(i/m) + r_Q) - F(F^{-1}((i-1)/m) - r_Q) \end{aligned} \tag{20}$$

Therefore, among the m children of the root,

$$\sum_{i=1}^{m} F(F^{-1}(i/m) + r_Q) - F(F^{-1}((i-1)/m) - r_Q) \tag{21}$$

nodes have to be accessed, on the average.

Above arguments are not directly applicable to the lower levels of the tree, because of constraints on the distance distribution. In fact, suppose the i-th child of the root, N_{r_i}, is accessed. The distance between the objects in the corresponding sub-tree is bounded by the triangle inequality to be lower than or equal to $2\mu_i$, as Figure 8 shows.

Since the probability of accessing the j-th child of N_{r_i}, denoted $N_{r_{i,j}}$, can be computed as:

$$\begin{aligned} &\Pr\{N_{r_{i,j}} \text{accessed}\} = \\ &\quad = \quad \Pr\{N_{r_{i,j}} \text{accessed} | N_{r_i} \text{accessed}\} \cdot \Pr\{N_{r_i} \text{accessed}\} \end{aligned}$$

[5] For simplicity of notation, here we assume that $F()$ is invertible.

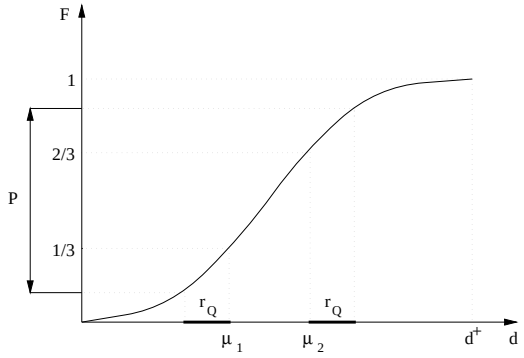


Figure 7: P is the probability to access the second child of the root in a 3-way *vp*-tree

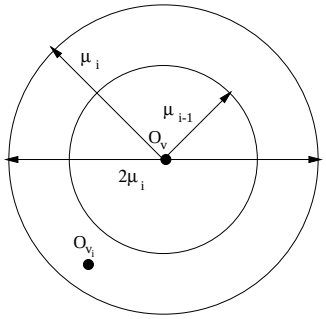


Figure 8: The distance between the objects in the sub-tree rooted at N_{r_i} cannot exceed $2\mu_i$

the problem is to determine $\Pr\{N_{r_{i,j}} \text{accessed} | N_{r_i} \text{accessed}\}$. For this Eq. 20 cannot be directly applied, since the maximum distance is now bounded by $2\mu_i$. Therefore, the distance distribution has to be "normalized" to the new bound, thus obtaining the distance distribution:

$$F_i(x) = \begin{cases} \dfrac{F(x)}{\min\{1, F(2\mu_i)\}} & \text{if } x \leq 2\mu_i \\ 1 & \text{if } x > 2\mu_i \end{cases} \tag{22}$$

Thus, the probability of accessing $N_{r_{i,j}}$ is obtained by substituting $F_i()$, as given by Eq. 22, for $F()$ in Eq. 20:

$$\begin{aligned} &\Pr\{N_{r_{i,j}} \text{accessed} | N_{r_i} \text{accessed}\} \approx \\ &\quad \approx \quad F_i(F_i^{-1}(j/m) + r_Q) - F_i(F_i^{-1}((j-1)/m) - r_Q) \end{aligned} \tag{23}$$

Following this approach it is possible to compute the probability of accessing every node of the *vp*-tree, thus obtaining a cost formula similar to Eq. 7, with $e(N_{r_i}) = 1$. The intuitive complexity of such a formula would suggest, as done for M-tree, to derive a level-based cost model. However, due to the "asymmetry" of the *vp*-tree – the probability of accessing a node depends on the specific path to the node itself – this appears to be a difficult problem.

6 Conclusions

In this work we have presented a cost model, N-MCM, for estimating CPU and I/O costs for processing range and nearest neighbors queries over generic metric spaces, when objects

are indexed using an M-tree. A simplified version of the model, L-MCM, has also been introduced, with the major aim of minimizing the amount of statistics of the tree to be kept. N-MCM is the first model for queries on metric spaces, and is based on the idea of using the distance distribution of objects and on the assumption of a probabilistically "homogeneous" metric space. This concept has been formalized, by introducing the HV index, and shown to be valuable to characterize both real and synthetic datasets. Experimental results show that both N-MCM and L-MCM accurately estimates costs, with errors rarely exceeding 10%. Finally, we also applied the same principles to show how a cost model for *vp*-trees could be derived.

From a both theoretical and practical point of view, our work raises some questions, to which we will try to answer in the prosecution of the research:

- A cost model which does not use tree statistics at all, but only relies on information derivable from the dataset, is the major challenge we are dealing with. The key problem appears to be formalizing the correlation between covering radii and the distance distribution.
- For non-homogeneous spaces ($HV \ll 1$) our model is not guaranteed to perform well. This suggests an approach which keeps several "viewpoints", and properly combines them to predict query costs. This would allow a cost model based on query "position" (relative to the viewpoints) to be derived, thus being able to change estimates depending on the specific query object.
- We plan to extend our cost model to deal with "complex" similarity queries [11] - queries consisting of more than one similarity predicate.
- We also intend to develop a complete model for *vp*-trees, following the arguments presented in Section 5, and to validate it through an experimental evaluation as done with M-tree.
- Finally, we plan to exploit concepts of fractal theory, which, we remind, is in principle applicable to generic metric spaces.

Acknowledgements

The authors would like to thank the PODS '98 Program Committee members for their helpful comments about the original version of the paper.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[2] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 299–310, Zurich, Switzerland, September 1995.

[3] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 78–86, Tucson, AZ, May 1997.

[4] T. Bozkaya and M. Özsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 357–368, Tucson, AZ, May 1997.

[5] T. Bozkaya, N. Yazdani, and M. Özsoyoglu. Matching and indexing sequences of different lengths. In *Proceedings of the 6th International Conference on Information and Knowledge Management (CIKM'97)*, pages 128–135, Las Vegas, NE, November 1997.

[6] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 574–584, Zurich, Switzerland, September 1995.

[7] W. Chen and K. Aberer. Efficient querying on genomic databases by using metric space indexing techniques. In *1st International Workshop on Query Processing and Multimedia Issues in Distributed Systems (PMIDS'97)*, Toulouse, France, September 1997.

[8] T. Chiueh. Content-based image indexing. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 582–593, Santiago, Chile, September 1994.

[9] P. Ciaccia and M. Patella. Bulk loading the M-tree. In *Proceedings of the 9th Australasian Database Conference (ADC'98)*, pages 15–26, Perth, Australia, February 1998.

[10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, Athens, Greece, August 1997.

[11] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, pages 9–23, Valencia, Spain, March 1998.

[12] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'94)*, pages 4–13, Minneapolis, MN, May 1994.

[13] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, MA, June 1984.

[14] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*,

pages 562–573, Zurich, Switzerland, September 1995. `http://gist.cs.berkeley.edu:8000/gist/`.

[15] D. P. Huttenlocher, G.A. Klanderman, and W. J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[16] I. Kamel and C. Faloutsos. On packing R-trees. In *Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM'93)*, pages 490–499, Washington, DC, November 1993.

[17] D. Maio and D. Maltoni. A structural approach to fingerprint classification. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume C, pages 578–585, Wien, Austria, August 1996.

[18] B. Mandelbrot. *The Fractal Geometry of Nature*. W.H.Freeman, New York, 1977.

[19] A. Papadopoulos and Y. Manolopoulos. Performance of nearest-neighbor queries in R-trees. In *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*, pages 394–408, Delphi, Greece, January 1997.

[20] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'96)*, pages 161–171, Montreal, Canada, June 1996.