Searching in Metric Spaces with User-Defined and Approximate Distances

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

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

Abstract

Metric access methods (MAMs), such as the M-tree, are powerful index structures for supporting similarity queries on metric spaces, which represent a common abstraction for those searching problems that arise in many modern application areas, such as multimedia, data mining, decision support, pattern recognition, and genomic databases. As compared to multi-dimensional (spatial) access methods (SAMs), MAMs are more general, yet they are reputed to lose in flexibility, since it is commonly deemed that they can only answer queries using the same distance function used to build the index. In this paper we show that this limitation is only apparent – thus MAMs are far more flexible than believed – and extend the M-tree so as to be able to support user-defined distance criteria, approximate distance functions to speed up query evaluation, as well as dissimilarity functions which are *not* metrics. The so-extended M-tree, also called QIC-M-tree, can deal with *three* distinct distances at a time: 1) a *query (user-defined) distance*, 2) an *index distance* (used to build the tree), and 3) a *comparison (approximate) distance* (used to quickly discard from the search uninteresting parts of the tree). We develop an analytical cost model that accurately characterizes the performance of QIC-M-tree and validate such model through extensive experimentation on real metric data sets. In particular, our analysis is able to predict the best evaluation strategy (i.e. which distances to use) under a variety of configurations, by properly taking into account relevant factors such as the distribution of distances, the cost of computing distances, and the actual index structure. We also prove that the overall saving in CPU search costs when using an approximate distance can be estimated by using information on the data set only – thus such measure is independent of the underlying access method – and show that performance results are closely related to a novel "indexing" error measure. Finally, we show how our results apply to other MAMs and query types.

1 Introduction

In this paper we tackle the problem of evaluating similarity queries in metric spaces. Similarity queries are a major trend in many modern database applications, such as multimedia, data mining, pattern recognition, molecular biology, and many others. In its essence, the problem is to find in a given collection of objects those objects that are most "similar" to a particular *query object*. In all these scenarios, the similarity of a pair of objects is typically evaluated by computing their relative *distance* in a suitably defined *space* (e.g. weighted Euclidean distance in a 10-dimensional vector space, edit distance over a string space, etc.), which depends on both the objects' nature and the matching criterion. Basic requirements for solving similarity queries are:

Correctness: All objects satisfying the query should be returned in the result; no *false dismissals* are allowed.[1]

Efficiency: The cost to be paid to solve the query (typically measured as the overall time needed to produce the results) should be as low as possible.

[1] *Approximate* similarity queries [CP01], which allow for missing some qualifying objects, are beyond the scope of this paper.

In order to efficiently answer similarity queries, many index structures have been proposed in recent years. Disregarding specific implementation details, they can be broadly classified in two categories, depending on the type of spaces and distances they support. The first category includes Spatial Access Methods (SAMs), such as the X-tree [BKK96] and the SR-tree [KS97] (see [GG98] for a survey on SAMs), that have been designed so as to improve performance when indexed objects are represented as high-dimensional vectors. This is the case for color histograms [FEF+94] and texture descriptors [MM96] for content-based image retrieval, Fourier vectors for temporal sequences [AFS93], and information on scientific data, only to name a few. In such cases, the number of dimensions of the space can vary from a few dozens to several hundreds. The second category of index structures consists of Metric Access Methods (MAMs), such as the M-tree [CPZ97], the mvp-tree [BÖ97], and the Slim-tree [TTSF00] (see also [CNBYM] for a recent survey), which aim to solve similarity queries on generic *metric spaces*, where the only requirement is that the distance function is a metric, thus objects need not to be necessarily represented as vectors. Unlike SAMs, a MAM organizes objects using only their relative distances – thus no geometric operations based on objects' "positions" are used – and exploits the triangle inequality to prune unrelevant parts of the search space.

Although MAMs have an intrinsic broader applicability than SAMs (since vector spaces, the only spaces to which SAMs apply, are a proper subset of metric spaces), a current limit to MAMs utilization originates from their lack of flexibility in supporting queries with a distance function different from the one used to build the index structure. A relevant case where these two distance functions may differ stems from the need to support *user-defined distance functions*, when the criterion used to assess the (dis)similarity between objects can be chosen at query time [SK97].

A further limitation to the efficient resolution of similarity queries in metric spaces is the possible high cost of distance computations, which can sometimes make MAMs CPU-bound. For instance, this is the case for the comparison of long strings which arise in genomic databases and for "ellipsoid" queries in high-dimensional vector spaces. Even if some optimizations to alleviate this problem have been already incorporated into MAMs [CPZ97], they are not sufficient yet to guarantee adequate performance levels on complex metric spaces [CPZ98a, CNBYM].

In this paper we start by showing that the limitation on the applicability of metric index structures is only apparent. The keys for adding flexibility to MAMs are the so-called *Lower-Bounding property*, which is also commonly used with SAMs, and "distance scaling", which allow MAMs to be applied in a wider variety of situations. We then provide an extension of the M-tree (which we take as the reference MAM throughout the paper) so as to process queries using distance functions based on user preferences. In particular, any distance function which is lower-bounded by the distance used to build the tree can be used to correctly query the index without any false dismissal, i.e. without losing any relevant object.

Turning to performance issues, we introduce the concept of *comparison* (approximate) distance function, to quickly prune the nodes of the tree that do not contain relevant objects. The so-extended M-tree, also called *QIC-M-tree*, can therefore deal with *three* distinct distances at a time:

1. the *query (user-defined) distance* (according to which the actual result must be computed),
2. the *index distance* (used to build the tree), and
3. the *comparison distance* (used to quickly discard uninteresting index paths).

It is worth pointing out that this enlarged scenario also includes the possibility to have *non-metric* query and/or comparison distance functions, a fact which indeed further broadens the scope of applicability of the QIC-M-tree.

In order to estimate performance of QIC-M-tree when solving similarity (range and k-nearest neighbor) queries, we present a cost model to analytically predict search costs given statistics about the data set and the index at hand. Specifically, since in a generic metric space no concept of "position" can be used, the only information exploited by the cost model is the *distance distribution*, i.e. the distribution of pairwise objects' distances. Experiments with real metric data sets clearly demonstrate the high accuracy of the cost model. Besides being applicable to predict query costs, the cost model can also be used to help with some relevant query processing issues, such as:

1. When several comparison distances are available, which one is the most effective for a given query?

2. On the other side, is the use of a comparison distance always effective?

We also show that our results are not peculiar to the QIC-M-tree but can also be applied to other types of (properly extended) MAMs: In particular, it is demonstrated how the saving that can be obtained by using a comparison distance *is independent of the underlying access method*, and can be derived from statistics on the data set only.

The paper is organized as follows. In Section 2 we give background information on MAMs and on user-defined distances. Section 3 introduces the basic concepts underlying our approach. In Section 4 we concentrate on range queries and present the algorithm for correctly solving them with QIC-M-tree. Section 4.1 introduces a cost model for analytically predicting index performance. Section 5 considers k-nearest neighbor queries. In Section 6 we validate the proposed cost model and provide extensive experimental results over real metric data sets. Section 7 shows how our results can also be adapted to other metric index structures. In Section 8 we discuss how to extend MAMs so as to support other query types. Section 9 presents a discussion on related works and Section 10 concludes. Relevant symbols used throughout the paper are given in Table 1.

Symbol	Description
$\mathcal{U}$	set of objects
$\mathcal{O} \subseteq \mathcal{U}$	data set
$M = \|\mathcal{O}\|$	number of objects in the data set
$d : \mathcal{U} \times \mathcal{U} \rightarrow \Re^+$	distance function
d_Q	query distance
d_I	index distance
d_C	comparison distance
$Q \in \mathcal{U}$	query object
ϵ	radius of a range query
k	number of objects requested by a k-nearest neighbors query
$S_{1\rightarrow 2}$	scaling factor of distance d_2 wrt d_1
d^+	finite upper bound on values of d
$F_{\mathbf{x}}(\cdot)$	distribution of the random variable $\mathbf{x}$
$f_{\mathbf{x}}(\cdot)$	density of $\mathbf{x}$
$E[\mathbf{x}]$	expected value of $\mathbf{x}$
$\#_{op}$	number of *op* operations needed to solve a query
$cost_{op}$	time needed for a single *op* operation
$time_{op} = \#_{op} \cdot cost_{op}$	time spent for performing operation *op* during search
L	height of the M-tree
N	node of the M-tree
$O^{[N]} \in \mathcal{U}$	routing object of node N
$r^{[N]}$	covering radius of node N
n_l	number of nodes at level l of the M-tree
$n = \sum_{l=1}^{L} n_l$	total number of nodes of the M-tree
$\overline{r_l}$	average covering radius for nodes at level l of the M-tree

Table 1: Summary of symbols.

2 Background

A metric space is a pair $(\mathcal{U}, d)$, where $\mathcal{U}$ is a set of objects and $d : \mathcal{U} \times \mathcal{U} \rightarrow \Re^+$ is a *metric*, i.e. a non-negative and symmetric binary function that also satisfies the triangle inequality: $d(O_i, O_j) \leq d(O_i, O_k) + d(O_k, O_j)$ for all objects $O_i, O_j, O_k \in \mathcal{U}$. Similarity queries are of two basic types: Given a data set $\mathcal{O} = \{O_1, \ldots, O_M\} \subseteq \mathcal{U}$, a range query, $\texttt{range}(\mathcal{O}, Q, \epsilon, d)$, selects all the objects O in $\mathcal{O}$ whose distance $d(Q, O)$ from the *query object*

$Q \in \mathcal{U}$ is not larger than ϵ. A k-nearest neighbor (k-NN) query, $\texttt{NN}(\mathcal{O}, Q, k, d)$, retrieves the k objects in $\mathcal{O}$ that have minimum distance d from Q.

2.1 Searching with Metric Access Methods

Given a set of objects, a MAM organizes them using a distance function d_I that measures their mutual dissimilarities. For the purpose of this paper, we call d_I the *index distance function*.

The unifying model presented in [CNBYM] indeed shows how all existing MAMs organize objects into a set of disjoint partitions, on top of which an index is built to drive the search to only those partitions which can contain relevant objects. For the case of metric trees (examples include the M-tree [CPZ97], the *Vantage Point* tree (vp-tree) [Yia93, Chi94], the mvp-tree [BÖ97, BÖ99], and the GNAT [Bri95]), partitions correspond to leaves of the tree and index (internal) nodes correspond to (possibly overlapping) regions of the metric space. During the search, only those nodes are accessed that may lead to objects satisfying the query.

The M-tree [CPZ97] has been the first *dynamic* metric tree to be introduced, in the sense that it supports random insertions and deletions without the need of costly re-organizations to avoid performance degradation. The recent Slim-tree [TTSF00] is also a dynamic MAM, which extends the M-tree with a new node splitting algorithm and a procedure to make the tree more compact. For the sake of definiteness, in the following we will focus our analysis on the M-tree, postponing discussion about other MAMs to Section 7.

2.1.1 The M-tree

The M-tree is a paged, dynamic, and balanced tree whose fixed-size nodes are mapped to disk pages and where indexed objects are stored in the leaf nodes.[2] Each node N corresponds to a *region* of the indexed metric space $(\mathcal{U}, d_I)$, defined as $Reg(N) = \{O \in \mathcal{U} | d_I(O^{[N]}, O) \leq r^{[N]}\}$, where $O^{[N]}$ is called the *routing object* of node N and $r^{[N]}$ is its *covering radius*. All the objects in the sub-tree rooted at N are then guaranteed to belong to $Reg(N)$, thus their distance from $O^{[N]}$ does not exceed $r^{[N]}$. Both $O^{[N]}$ and $r^{[N]}$ are stored, together with a pointer to node N, $ptr(N)$, in an entry of the parent node of N. In order to save distance computations, the M-tree also stores pre-computed distances between a routing object and its parent. Thus, if N_p is the parent node of N, the entry for N in node N_p also includes the value of $d_I(O^{[N_p]}, O^{[N]})$.

Figure 1 shows an example of (a part of) an M-tree in the 2-D space equipped with a quadratic (elliptic) distance function (see also Section 2.2): In particular, it has to be noted that all objects contained in nodes N_3 and N_4 are also contained in the region associated with the parent node N_1.

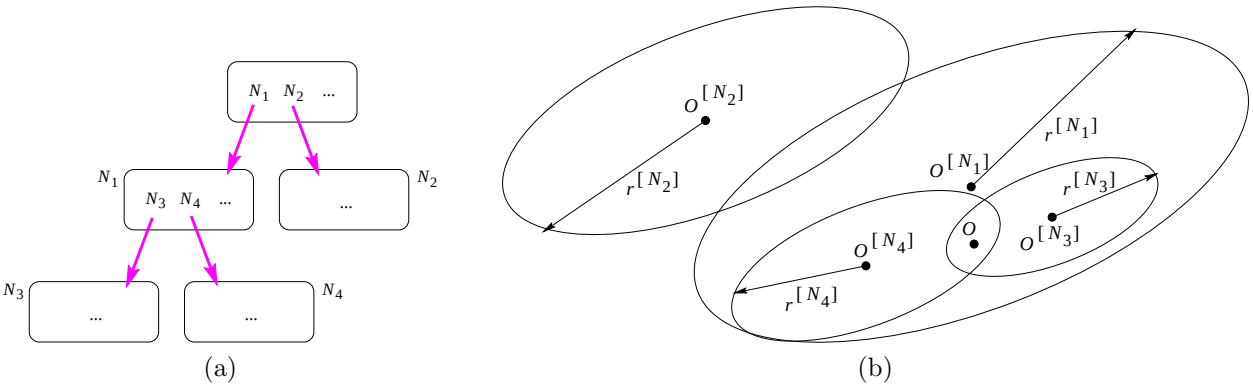


Figure 1: An example of (a part of) an M-tree (a) and regions associated to each node in the 2-D space with a quadratic distance function (b). For each node N, the routing object $O^{[N]}$ and the covering radius $r^{[N]}$ are shown.

[2] The source code of the M-tree is publicly available at URL: `http://www-db.deis.unibo.it/Mtree`.

Besides range and k-NN queries (covered in Sections 4 and 5, respectively), the M-tree also supports *approximate* NN queries [CP00] and *complex* similarity queries (where more than one query object is specified) [CPZ98b] (see also Section 8.2). Performance of the M-tree for the basic query types can also be analytically predicted [CPZ98a].

2.2 User-Defined Distance Functions

In many situations it is desirable that users can decide at query time which is the "best" distance function to use. For instance, consider the case of color-based image retrieval [SK97], where it is assumed that objects are represented as histograms of colors (i.e. vectors). In this case, the (dis)similarity between two histograms can be computed by multiplying the difference between each couple of histogram bins by the similarity between respective colors. This leads to a quadratic distance function:

$$d_{qf}[A](O_1, O_2) = \sqrt{(O_1 - O_2)^T A (O_1 - O_2)} = \sqrt{\sum_i \sum_j a_{ij}(O_{1i} - O_{2i})(O_{1j} - O_{2j})} \quad (1)$$

where element a_{ij} of the symmetric positive definite matrix A is the similarity between colors i and j.[3]

The fact that each user can specify a different A matrix to compare color histograms can be the rule, rather than the exception. Indeed, different users could have different query requirements and the same user should also be given the possibility to change the way to query the system, based on his/her current preferences. Another common situation where the user distance can change over time is when relevance feedback is supported: In this case the distance can vary at each iteration of the feedback cycle depending on the relevance judgments provided by the user [PC99, BCW01].

The following Definition concisely captures cases like these.

Definition 1 (Class of distance functions)
Let $d[\theta]$ be a distance function specified by a vector θ of parameter values and let $\Theta = \{\theta\}$ be the space of possible parameter vectors. The set $d[\Theta] = \{d[\theta] | \theta \in \Theta\}$ is called a class of distance functions *with parameter θ.* □

Example 1
L_p ($p = 1, 2, \ldots, \infty$) norms are the most commonly used distance functions for D-dimensional vector spaces, and are defined as:

$$L_p(O_1, O_2) = \left(\sum_{j=1}^{D} |O_{1j} - O_{2j}|^p \right)^{\frac{1}{p}} \qquad (1 \leq p < \infty)$$

$$L_\infty(O_1, O_2) = \max_{j=1}^{D} \{|O_{1j} - O_{2j}|\}$$

The set $P = \{1, 2, \ldots, \infty\}$ of all possible p values leads to the definition of the $d_L[P]$ class of L_p metrics. □

Example 2
User-adaptable similarity queries based on quadratic form distance functions (see Equation 1) form the class $d_{qf}[\mathcal{A}]$, where $\mathcal{A}$ is the space of all symmetric positive definite matrices. Each matrix $A \in \mathcal{A}$ leads to "ellipsoid queries" in the D-dimensional vector space, where the shape and the orientation of the ellipsoid depend on the similarity matrix A. □

Example 3
The family of edit distances $d_{edit}[\Gamma]$ is commonly used to measure the dissimilarity of strings. In particular, the "unweighted" edit distance counts the minimum number of atomic edit operations (insertion, deletion,

[3] The fact that A has to be positive definite is not really necessary, depending on the data distribution, see [HSE+95].

and substitution of one symbol) needed to transform a string into another one. With weighted edit distances, each atomic operation has its own "cost" (or *weight*).

Formally, let Σ be a finite alphabet, let Σ^* be the set of all finite-length strings over Σ, and let λ be the *null* symbol. An *edit transformation* from string X to string Y is a sequence $S = S_1 S_2 \ldots S_m$ of elementary edit operations, i.e. $S_j = (A_j \rightarrow B_j)$, where $A_j, B_j \in \Sigma \cup \{\lambda\}$. A *weight function* γ is used to assign to each operation $A \rightarrow B$ a non-negative cost, $\gamma(A \rightarrow B)$. The cost of an edit transformation S can be defined as $\gamma(S) = \sum_{j=1}^{m} \gamma(S_j)$. The weighted edit distance induced by γ is then:

$$d_{edit}[\gamma](X, Y) = \min\{\gamma(S) | S \text{ is an edit transformation from } X \text{ to } Y\}$$

The complexity of computing $d_{edit}[\gamma](X, Y)$ is $O(len(X) \times len(Y))$, where $len(X)$ and $len(Y)$ are the length of strings X and Y, respectively [WF74]. This makes edit distance computation highly CPU demanding for the case of long strings (e.g. strings corresponding to human chromosomes contain millions of characters, each representing a nucleic acid type).

It is known that if the two following conditions hold:

1. $\gamma(A \rightarrow B) = \gamma(B \rightarrow A) > 0$ if $A \neq B$, and
2. $\gamma(A \rightarrow A) = 0$, $\forall A, B \in \Sigma \cup \{\lambda\}$

then $d_{edit}[\gamma]$ is a metric over the Σ^* space, *regardless* of the specific values of the weights. The unweighted edit distance, simply denoted d_{edit} from now on, is obtained by setting $\gamma(A \rightarrow B) = 1$ $\forall A, B \in \Sigma \cup \{\lambda\}$ with $A \neq B$. Also important is the case where no symbol substitutions are permitted. The corresponding distance is then defined by a weight function $\gamma_{i,d}$ such that $\gamma_{i,d}(A \rightarrow B) = \infty$ when $A \neq B$ and $A, B \neq \lambda$, otherwise it assigns cost 1 to each insertion or deletion of symbols. It is immediate to show that setting $\gamma_{i,d}(A \rightarrow B) = 2$ does not alter at all the distance function, since a symbol substitution can always be obtained by one deletion followed by an insertion, each of which costs 1. □

3 New Scenarios for Metric Access Methods

Current limits to MAMs applicability are their lack of flexibility in supporting queries with a distance function different from the one used to build the index and the high distance computation costs to be paid in complex metric domains.

The basic tool for extending MAMs is the lower-bounding property between distances. Such concept has been already profitably used by SAMs to support user-adaptable distance functions [SK97, SK98].

Definition 2 (Lower-bounding distance function)
Let d_1 and d_2 be two distance functions over $\mathcal{U}$, we say that d_1 is a lower-bounding distance function *of d_2 ($d_1 \preceq d_2$ for short) if d_1 is an underestimate of d_2, that is:*

$$d_1(O_i, O_j) \leq d_2(O_i, O_j) \qquad \forall O_i, O_j \in \mathcal{U} \tag{2}$$

□

In order to support queries with user-defined distance functions, it is useful to extend above definition to *classes* of distance functions. As a preparatory step, assume $d_1 \not\preceq d_2$, yet there exists a real value $S_{1\rightarrow 2} > 0$ such that $\forall O_i, O_j \in \mathcal{U}$ it is $d_1(O_i, O_j) \leq S_{1\rightarrow 2}\, d_2(O_i, O_j)$. In this case, we call $S_{1\rightarrow 2}$ the *scaling factor* of d_2 with respect to (wrt) d_1 and write $d_1 \preceq S_{1\rightarrow 2}\, d_2$. Clearly, if there exists $S_{1\rightarrow 2}$ such that $d_1 \preceq S_{1\rightarrow 2}\, d_2$ holds, infinitely many other values for the scaling factor exist. It is therefore advisable to consider the minimum of such values, since it makes d_1 a tight lower-bound of $S_{1\rightarrow 2}\, d_2$. This concept is captured by the following

Definition 3 (Optimal scaling factor)
Let d_1 and d_2 be two distance functions over $\mathcal{U}$, and let $S_{1\rightarrow 2}$ the minimum value such that $d_1 \preceq S_{1\rightarrow 2}\, d_2$ holds. We call $S_{1\rightarrow 2}$ the optimal scaling factor of d_2 wrt d_1. □

We are now ready to generalize the concept of lower-bounding distance function to a class of distance functions.

Definition 4 (Lower-bound of a class of distance functions)
If d_1 is a distance function and $d_2[\Theta]$ is a class of distance functions, then d_1 is a lower-bounding distance function *for the $d_2[\Theta]$ class, written $d_1 \preceq d_2[\Theta]$, iff for each θ there exists a scaling $S_{1\to 2}[\theta]$ such that $d_1 \preceq S_{1\to 2}[\theta]\ d_2[\theta]$.* □

The following Claim, which will be proved to hold for the M-tree (see Theorem 2 and Corollary 2) as well as for other specific MAMs (see Section 7), exploits above definitions to support user-defined distance functions.

Claim 1
Let $\mathcal{M}$ be a MAM built on a data set $\mathcal{O} \subseteq \mathcal{U}$ using the d_I metric, and d_Q a distance function over $\mathcal{U}$, such that $d_I \preceq S_{I\to Q}\ d_Q$, then $\mathcal{M}$ can also correctly process range and k-NN queries based on d_Q. □

Although we have not a formal proof that the Claim holds for *any* MAM, all our results as well as the logic used to prove them strongly support the opinion that this is likely to be the case.

Next Proposition shows that we can also use approximate (cheap) distances as a pre-test before computing the actual query distance on data objects. This can be exploited whenever a threshold on query distance values is available (this is the case for range queries, as well as for k-NN queries where the threshold is dynamically modified during the search). Note that in this case the presence of a MAM is unrelevant.

Proposition 1
Let d_C be a distance function over $\mathcal{U}$, such that $d_C \preceq S_{C\to Q}\ d_Q$, then d_C, also called a comparison distance, *can be used as a pre-test (so as to avoid computing the actual d_Q) when checking whether an object should be included in the result.* □

Proof: If the threshold on query distance values is ϵ, and the pre-test yields $d_C(Q,O) > S_{C\to Q}\ \epsilon$, then $d_Q(Q,O) \geq d_C(Q,O)/S_{C\to Q} > \epsilon$, thus object O can be safely excluded from the result. □

The use of a comparison distance is not restricted to data objects. Indeed, when a MAM is available, d_C can also be used as a pre-test for the index distance d_I.

Claim 2
Let $\mathcal{M}$ be a MAM built on a data-set $\mathcal{O} \subseteq \mathcal{U}$ using the d_I distance, and d_C a distance function over $\mathcal{U}$, such that $d_C \preceq S_{C\to I}\ d_I$, then d_C can be used as a pre-test (so as to avoid computing the actual d_I) when searching the index. □

Observations made for Claim 1 still apply here. As a remark concerning performance, it is intuitive that a "good" comparison distance should satisfy two contrasting requirements:

1. d_C should be computationally cheap to evaluate with respect to d_I and d_Q;

2. d_C should be a good approximation of both $S_{C\to I}\ d_I$ and $S_{C\to Q}\ d_Q$.

Above Claims and Proposition allow for a variety of scenarios, the more general being the one where *three* distinct distance functions (d_I, d_Q, and d_C) are used. Overall, this leads to what we call the *QIC* (pronounced "quick") approach to similarity query processing in metric spaces.

It has also to be remarked that in Claims 1 and 2 and Proposition 1 no metric assumption is made on distances d_Q and d_C, thus *MAMs can also be used to answer queries where the user similarity criterion is not a metric one.*

3.1 Remarks on SAMs and the Filter & Refine Approach

As said, the lower-bounding property is also used by SAMs to deal with user-defined distance functions. As an example, consider the case when the query function d_Q is a quadratic form and a SAM using rectilinear regions (like the R-tree family [Gut84, BKSS90, BKK96]) is used: Since, particularly in high-dimensional spaces, computing whether an elliptic region and a box intersect is a very expensive task, the query region is approximated using its minimum bounding box or sphere, thus defining a new distance function that lower-bounds d_Q [ABKS98]. The same approach is also used when the user metric on the object domain is replaced with an approximate lower-bounding distance function on the *feature* domain, where features are vectors of suitable dimensionality that provide a simplified representation of objects' content (this is the technique used in the GEMINI approach [ZCF+97, Chapter 12]); then a SAM is used to index objects features. Both techniques lead to a two-steps *filter & refine* (F&R) query processing strategy, where the filter step uses the SAM with the approximate distance function and the refine step discards, using the actual object distance, the *false drops*, i.e. objects included in the approximate result but that do not satisfy the original query.

Unlike above strategies, in the QIC approach a query is completely solved within the MAM, thus no objects transformations or refinement steps are needed. Nonetheless, when reputed convenient from a performance point of view, a MAM-based F&R strategy can still be adopted, as we discuss in Section 6.2.4.

3.2 Some Relevant Cases

In the following we enumerate some relevant cases to which the QIC approach applies.

L_p Norms: The following corollary is based on well-known results on L_p norms.

Corollary 1
Every $L_{p'}$ norm is a lower-bounding distance function for the class of L_p norms, $L_{p'} \preceq d_L[P]$. The optimal scaling factor equals 1 if $p' > p$, and $D^{1/p'-1/p}$ if $p' < p$, where D is the dimensionality of the vector space. □

Proof: The result follows immediately from the inequalities $L_{p'} \preceq L_p$, if $p' > p$, and $L_{p'} \preceq D^{1/p'-1/p} L_p$, if $p' < p$. □

Quadratic Form Distance Functions: In [HSE+95] it is proved that L_2 is a lower-bounding distance function for the class of quadratic form distance functions, $L_2 \preceq d_{qf}[\mathcal{A}]$, and that the optimal scaling factor $S_{L_2 \to qf}[A]$ is given by the inverse of the square root of the minimum eigenvalue of A, $S_{L_2 \to qf}[A] = 1/\sqrt{\min_j\{\lambda_j\}}$.[4]

Weighted Edit Distances: The following Theorem proves that any weighted edit distance is a lower-bounding distance function for its class.

Theorem 1
Let Γ be the set of all weight functions γ such that $d_{edit}[\gamma]$ is a weighted edit distance over Σ^*, the set of all finite-length strings over a finite alphabet Σ, and let $\gamma_I \in \Gamma$. Then, $d_{edit}[\gamma_I] \preceq d_{edit}[\Gamma]$ and the optimal scaling for $d_{edit}[\gamma]$ is given by $S_{\gamma_I \to \gamma} = 1/\min_{A,B,A \neq B}\{\gamma(A \to B)/\gamma_I(A \to B)\}$. □

Proof: Given in Appendix B. □

The Multiset Distance: Given a string X, let $x = ms(X)$ denote the multiset (bag) of symbols in X. For instance, $ms(\texttt{"tree"}) = \{\{\texttt{t}, \texttt{r}, \texttt{e}, \texttt{e}\}\}$. The following can be easily proved to be a metric on multisets:

$$d_{ms}(x, y) = \max\{|x - y|, |y - x|\} \tag{3}$$

[4] An alternative formulation of this result can be found in Appendix A.

where the difference has a bag semantics (e.g. $\{\{\mathtt{a},\mathtt{a},\mathtt{a},\mathtt{b}\}\} - \{\{\mathtt{a},\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{c}\}\} = \{\{\mathtt{a}\}\}$), and $|\cdot|$ counts the number of elements in a multiset (e.g. $|\{\{\mathtt{a},\mathtt{a}\}\}| = 2$). In practice, $d_{ms}(x,y)$ first "drops" common elements, then takes the maximum considering the number of "residual" elements. For instance:

$$d_{ms}(\{\{\mathtt{a},\mathtt{a},\mathtt{a},\mathtt{b}\}\} - \{\{\mathtt{a},\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{c}\}\}) = \max\{|\{\{\mathtt{a}\}\}|, |\{\{\mathtt{c},\mathtt{c}\}\}|\} = 2$$

It is immediate to observe that $d_{ms}(x,y)$ is a lower-bounding distance for the unweighted edit distance, i.e. $d_{ms}(x,y) \leq d_{edit}(X,Y)\ \forall X,Y \in \Sigma^*$, thus $d_{ms} \preceq d_{edit}$. It then easily follows from Theorem 1 that $d_{ms} \preceq d_{edit}[\Gamma]$, and the scaling factor of $d_{edit}[\gamma]$ wrt d_{ms} is $1/\min_{A,B,A\neq B}\{\gamma(A \to B)\}$.

As a specific example, if a metric index is built using the Euclidean distance, i.e. $d_I \equiv L_2$, it also supports queries with the "city-block" distance, $d_Q \equiv L_1$ (with scaling factor 1), with the "max metric", $d_Q \equiv L_\infty$ (with scaling factor $D^{1/2-1/\infty} = \sqrt{D}$), as well as with any quadratic form distance function. As another example, one could have an index built with a weighted edit distance that, say, reflects the typing error probabilities due to a "default" keyboard layout, and then issue queries using *any* other weighted distance, e.g. considering user's own keyboard. Further, since computing $d_{ms}(ms(X), ms(Y))$ is in $O(len(X)+len(Y)+|\Sigma|)$, d_{ms} can indeed be effectively used as a comparison distance for edit distances.[5]

3.2.1 Non-Metric Distances

Following are two deliberately simple examples aiming to illustrate how a MAM can be also used in the case when d_Q is not a metric. For this assume that one wants to support both "subset" and "superset" queries on strings. Namely, for subset queries the distance function is:

$$d_{\subseteq}(X,Y) = \begin{cases} len(Y) - len(X) & \text{if } X \text{ is a substring of } Y \\ \infty & \text{otherwise} \end{cases}$$

Similarly, for superset queries it is:

$$d_{\supseteq}(X,Y) = \begin{cases} len(X) - len(Y) & \text{if } Y \text{ is a substring of } X \\ \infty & \text{otherwise} \end{cases}$$

Clearly, since neither $d_{\subseteq}$ nor $d_{\supseteq}$ are symmetric, they are not metrics. However it is easy to see that both $d_{edit} \preceq d_{\subseteq}$ and $d_{edit} \preceq d_{\supseteq}$ hold. Thus, a MAM built using $d_I \equiv d_{edit}$ would be able to answer queries based on both $d_{\subseteq}$ and $d_{\supseteq}$.

Another relevant case to consider concerns the Earth Mover's Distance (EMD), which has been proved to be an effective way to compare distributions of values [RTG98]. The EMD extends quadratic form distance functions based on histograms and as such it has been successfully applied for image retrieval. In general, the EMD is *not* a metric, yet [RTG98] shows how under particular circumstances it is indeed possible to find a metric d_{EMD} such that $d_{EMD} \preceq EMD$ holds. Again, this would make it possible to build an index using $d_I \equiv d_{EMD}$ for answering queries with $d_Q \equiv EMD$.

4 Range Queries

To show how the M-tree, which, we recall, we take as the reference MAM, can correctly process range queries expressed with a distance function d_Q which is different from the d_I used to build the tree, it is first useful to review the basic logic of the search algorithm for the basic case $d_C \equiv d_I \equiv d_Q$ (see also [CPZ97]).

Given the range query $\mathtt{range}(\mathcal{O}, Q, \epsilon, d_I)$, the M-tree is recursively descended and sub-trees are accessed iff the region associated with their root node overlaps the query region (see Figure 2). For a given node N with routing object $O^{[N]}$ and covering radius $r^{[N]}$, this amounts to check if $d_I(Q, O^{[N]}) > \epsilon + r^{[N]}$ holds, since from the triangle inequality it follows:

$$d_I(Q, O^{[N]}) > \epsilon + r^{[N]} \implies d_I(Q,O) > \epsilon \qquad \forall O \in Reg(N) \tag{4}$$

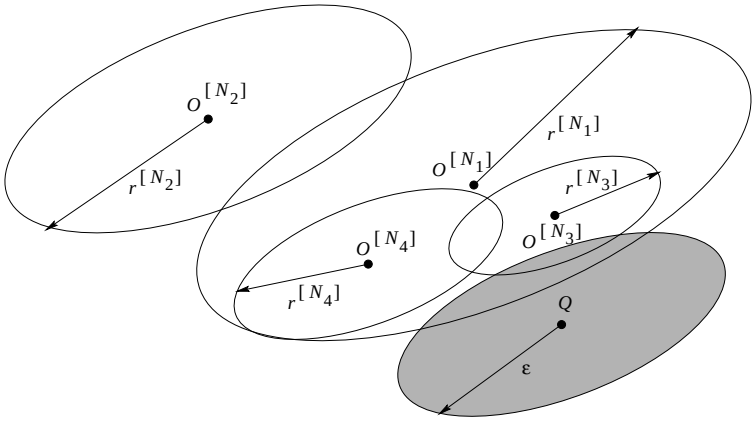


Figure 2: Searching in an M-tree. N_2 and N_4 are pruned since their regions do not overlap the query region.

The recursive `RangeSearch` M-tree algorithm is shown in Figure 3. It is assumed that the search starts from the root node. Lines 4. and 9. both exploit the triangle inequality so as to avoid to compute unnecessary distances (since the involved distance values either have been already computed, $d_I(Q, O^{[N]})$, or are stored in the considered node, $d_I(O^{[N]}, O_i)$, see Lemma 3.2 in [CPZ97]). In particular, consider line 4.: If $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O_i)| > \epsilon$, then from the triangle inequality it is derived $d_I(Q, O_i) > \epsilon$, thus object O_i does not belong to the result.

Algorithm `RangeSearch` (node N, query object Q, threshold ϵ)

1. Let $O^{[N]}$ be the routing object of node N;
2. If N is a leaf node then:
3. For each object O_i in N do:
4. If $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O_i)| \leq \epsilon$ then:
5. Compute $d_I(Q, O_i)$;
6. If $d_I(Q, O_i) \leq \epsilon$ then add O_i to the result set;
7. else: // N is an internal node
8. For each child node N_c of N do:
9. If $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O^{[N_c]})| \leq \epsilon + r^{[N_c]}$ then:
10. Compute $d_I(Q, O^{[N_c]})$;
11. If $d_I(Q, O^{[N_c]}) \leq \epsilon + r^{[N_c]}$ then:
12. Fetch node N_c and call `RangeSearch`(N_c, Q, ϵ);
13. End.

Figure 3: Algorithm for processing range queries with M-tree.

The `QIC-RangeSearch` algorithm, shown in Figure 4, applies when both a query distance d_Q and an approximate distance d_C are used to answer the query `range`$(\mathcal{O}, Q, \epsilon, d_Q)$. With respect to `RangeSearch`, the following modifications are needed:

- On internal nodes, the index distance d_I is used together with a "scaled threshold" $S_{I \to Q}\ \epsilon$ (see line 15.). Indeed, since covering radii are computed using d_I, the index distance has to be used to check whether a node could be pruned from the search.
- The same applies at lines 4. and 11. where, as in `RangeSearch`, already available distances are used to

[5] Other distances can be used to approximate the edit distance [KS01]; the multiset distance d_{ms} has, however, the advantage that it does not require further processing of strings and is very fast to compute.

prune objects from the result. In particular, $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O_i)| > S_{I \to Q}\ \epsilon$ implies $d_I(Q, O_i) > S_{I \to Q}\ \epsilon$, thus $d_Q(Q, O_i) > \epsilon$.

- At line 6. the pre-test based on the comparison distance d_C is executed.
- d_C is also used in line 13. to save d_I computations (note that in this case, the $S_{C \to I}$ scaling factor has to be used).

Algorithm `QIC-RangeSearch` (node N, query object Q, distances d_Q, d_C, threshold ϵ, scaling factors $S_{C \to I}$, $S_{C \to Q}$, $S_{I \to Q}$)

```
1.  Let O^[N] be the routing object of node N;
2.  If N is a leaf node then:
3.      For each object O_i in N do:
4.          If |d_I(Q, O^[N]) − d_I(O^[N], O_i)| ≤ S_{I→Q} ε then:
5.              Compute d_C(Q, O_i);
6.              If d_C(Q, O_i) ≤ S_{C→Q} ε then:
7.                  Compute d_Q(Q, O_i);
8.                  If d_Q(Q, O_i) ≤ ε then add O_i to the result set;
9.  else: // N is an internal node
10.     For each child node N_c of N do:
11.         If |d_I(Q, O^[N]) − d_I(O^[N], O^[N_c])| ≤ S_{I→Q} ε + r^[N_c] then:
12.             Compute d_C(Q, O^[N_c]);
13.             If d_C(Q, O^[N_c]) ≤ S_{C→I}(S_{I→Q} ε + r^[N_c]) then:
14.                 Compute d_I(Q, O^[N_c]);
15.                 If d_I(Q, O^[N_c]) ≤ S_{I→Q} ε + r^[N_c] then:
16.                     Fetch node N_c and call QIC-RangeSearch(N_c, Q, d_Q, d_C, ε, S_{C→I}, S_{C→Q}, S_{I→Q});
17. End.
```

Figure 4: The `QIC-RangeSearch` algorithm.

Theorem 2 (The QIC Theorem)
The `QIC-RangeSearch` algorithm returns the correct answer to the query $\texttt{range}(\mathcal{O}, Q, \epsilon, d_Q)$. □

Proof: Given in Appendix B. □

4.1 Cost Model

Here we present an analytical cost model to evaluate the performance of the `QIC-RangeSearch` algorithm. The cost model exploits statistical information about the data set and the index tree built on it:[6]

Data Set Statistics: We use the *distance distribution* of objects, which, since its introduction in [CPZ98a], has been successfully adopted to predict performance of metric structures [TTF00, TTSF00, CNBYM]. Formally, let $F_{\mathbf{d}}(\cdot)$ be the distance distribution corresponding to distance d, that is:

$$F_{\mathbf{d}}(x) = \Pr\{d(\mathbf{O_i}, \mathbf{O_j}) \leq x\} = \Pr\{\mathbf{d} \leq x\} \qquad x \in [0, d^+]$$

where $\mathbf{O_i}$ and $\mathbf{O_j}$ are randomly chosen objects of $\mathcal{O}$, d^+ is a finite upper bound on distance values, and $\mathbf{d} = d(\mathbf{O_i}, \mathbf{O_j})$ is a random variable with distribution $F_{\mathbf{d}}(\cdot)$. With $f_{\mathbf{d}}(x) = \frac{d}{dx} F_{\mathbf{d}}(x)$ we denote the probability density function (pdf) of $F_{\mathbf{d}}(x)$.

[6] A *tree independent* cost model would necessarily rely on optimality assumptions (e.g. the fact that node regions do not overlap) that are completely unrealistic given the state-of-the-art for MAM construction algorithms [TTSF00]. In the following, therefore, we will assume that costs are estimated given an M-tree built over the considered data set using a particular construction algorithm (which is completely uninfluential to our purposes).

Index Statistics: For each level l of the tree ($l = 1, 2, \ldots, L$, where L is the tree height), the number of nodes n_l and the average covering radius $\overline{r_l}$ are maintained. This is the same approximation considered in the Level-based cost model of [CPZ98a] and leads to a negligible $O(L) = O(\log M)$ storage overhead.

In general, the time needed for solving the query $\texttt{range}(\mathcal{O}, Q, \epsilon, d_Q)$ is composed by two terms: The I/O time, needed for fetching index nodes into main memory, and the CPU time, which can be further split into the time needed for computing distances and the overhead time spent by the algorithm for recursion and management of structures. Since the latter is orders of magnitude lower than the distance computation cost [CNBYM], in the following we will only consider the I/O time and the CPU time due to distance computations.

The time spent in performing each operation is obtained by multiplying the expected number of operations by the average cost of a single operation:

$$\begin{aligned} time_{total} = time_{I/O} + time_{CPU} &\simeq time_{I/O} + time_{d_Q} + time_{d_I} + time_{d_C} \\ time_{I/O}(\epsilon; d_I; S_{I\to Q}) &= \#_{I/O}(\epsilon; d_I; S_{I\to Q}) \cdot cost_{I/O} \\ time_{d_C}(\epsilon; d_I, d_C; S_{I\to Q}) &= \#_{d_C}(\epsilon; d_I; S_{I\to Q}) \cdot cost_{d_C} \\ time_{d_I}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to I}) &= \#_{d_I}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to I}) \cdot cost_{d_I} \\ time_{d_Q}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q}) &= \#_{d_Q}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to Q}) \cdot cost_{d_Q} \end{aligned} \tag{5}$$

Since average costs for each single operation can be easily computed by sampling the data set, we concentrate on the estimation of the number of atomic operations needed to solve a query. The dependencies of cost components on the parameters are explicited in Equations 5. It is important to note that:

- No number (#) directly depends on (the distribution of) the query distance d_Q; this can also be argued by looking at the `QIC-RangeSearch` algorithm, since computing d_Q only serves to check whether an object has to be inserted in the result set.
- The number of I/Os, $\#_{I/O}$, only depends on the query radius ϵ, on the index distance d_I, and on the scaling factor $S_{I\to Q}$. In particular, it is independent of the specific d_C (and on scaling factors $S_{C\to Q}$ and $S_{C\to I}$), since the use of a comparison distance can only affect the number of distances to be computed but not the number of nodes to be retrieved. Same arguments apply to $\#_{d_C}$, the number of d_C computations.
- On the other hand, the number of computed index and query distances, $\#_{d_I}$ and $\#_{d_Q}$ respectively, depend also on d_C and the corresponding scaling factor.

In order to predict the number of I/O accesses, we consider the probability that a node N of the M-tree, with routing object $O^{[N]}$ and covering radius $r^{[N]}$, has to be accessed by the query $\texttt{range}(\mathcal{O}, Q, \epsilon, d_Q)$. This is the case iff $d_I(Q, O^{[N]}) \leq S_{I\to Q}\ \epsilon + r^{[N]}$ (line 15. of `QIC-RangeSearch` algorithm). Hence, it is

$$\Pr\{\text{node } N \text{ is accessed}\} = \Pr\left\{ d_I(Q, \mathbf{O^{[N]}}) \leq S_{I\to Q}\ \epsilon + r^{[N]} \right\} = F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + r^{[N]})$$

which, considering the level-based index statistics, leads to

$$\boxed{\#_{I/O}(\epsilon; d_I; S_{I\to Q}) = \sum_{l=1}^{L} n_l \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_l})} \tag{6}$$

For each accessed node, the test at line 4. (for a leaf) or 11. (for an internal node) has to be performed on all its entries; therefore, for each level, the number of such tests is

$$n_{l+1} \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_l})$$

where $n_{L+1} \stackrel{\text{def}}{=} M$ is the number of indexed objects. Since d_C is computed only when the test fails, in order to estimate $\#_{d_C}$ we need the probability that $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O^{[N_c]})| \leq S_{I\to Q}\ \epsilon + r^{[N_c]}$, where node N_c is a child of node N (obviously, for leaf nodes it is $r^{[N_c]} = 0$). This can be obtained from the distance distribution of the random variable $\mathbf{z} = |d_I(Q, \mathbf{O^{[N]}}) - d_I(\mathbf{O^{[N]}}, \mathbf{O^{[N_c]}})|$.

Lemma 1
The probability that the test at line 11. fails is

$$F_{\mathbf{z}}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}}) = \\ = \int_0^{\overline{r_l}-\overline{r_{l+1}}} \frac{f_{\mathbf{d_I}}(y)}{F_{\mathbf{d_I}}(\overline{r_l}-\overline{r_{l+1}})F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_l})} \left(F_{\mathbf{d_I}}(y + S_{I\to Q}\ \epsilon + \overline{r_{l+1}}) - F_{\mathbf{d_I}}(y - S_{I\to Q}\ \epsilon - \overline{r_{l+1}})\right)\ dy \quad (7)$$

□

Proof: Given in Appendix B. □

Above Lemma immediately leads to

$$\boxed{\#_{d_C}(\epsilon; d_I; S_{I\to Q}) = \sum_{l=1}^{L} n_{l+1} \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_l}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}})} \quad (8)$$

where $\overline{r_{L+1}} = 0$ and $F_{\mathbf{z}}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}})$ is obtained from Equation 7.

The probability that $d_I(Q, O^{[N_c]})$ is computed equals the probability that the test at line 13. fails:

$$\Pr\left\{d_I(Q, \mathbf{O^{[N_c]}}) \text{ is computed}\right\} = \\ = \Pr\left\{d_C(Q, \mathbf{O^{[N_c]}}) \le S_{C\to I}(S_{I\to Q}\ \epsilon + r^{[N_c]})\right\} = F_{\mathbf{d_C}}(S_{C\to I}(S_{I\to Q}\ \epsilon + r^{[N_c]}))$$

Therefore, the total number of d_I computations can be estimated as

$$\boxed{\begin{aligned}&\#_{d_I}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to I}) = \\ &\quad = \sum_{l=1}^{L-1} n_{l+1} \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_l}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}}) \cdot F_{\mathbf{d_C}}(S_{C\to I}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}}))\end{aligned}} \quad (9)$$

Likewise, the number of computed query distances d_Q (line 6.) is estimated as

$$\boxed{\#_{d_Q}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to Q}) = M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \cdot F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon)} \quad (10)$$

Of course, when no comparison distance is used, it is $F_{\mathbf{d_C}}(x) = 1$ and $cost_{d_C} = 0$, thus we obtain

$$\#_{d_I}(\epsilon; d_I, \_, ; S_{I\to Q}, 1) = \sum_{l=1}^{L-1} n_{l+1} \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon + \overline{r_{l+1}}) \quad (11)$$

$$\#_{d_Q}(\epsilon; d_I, \_; S_{I\to Q}, 1) = M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \quad (12)$$

As anticipated, the overall cost for a range query *does not depend on the distance distribution of the query distance* $F_{\mathbf{d_Q}}$, but only on the scaling factors. In Equations 6, 8, 9, and 10, the values of the scaling factors appear as multiplicative factors for the query radius ϵ. Therefore, in order to reduce costs, the scaling factors should be taken as low as possible, which confirms the need for optimal scaling factors advocated in Section 3.

5 Nearest Neighbor Queries

The basic principles of the k-NN search algorithm for the case when $d_C \equiv d_I \equiv d_Q$ are summarized as follows (see also [CPZ97]). The algorithm, described in Figure 5, uses a priority queue, `PQ`, of pointers to nodes of the tree. These are kept ordered by increasing values of $d_I^{min}(Q, Reg(N))$, which is the

minimum distance between the query point Q and the region of node N, computed as $d_I^{min}(Q, Reg(N)) = \max\{0, d_I(Q, O^{[N]}) - r^{[N]}\}$. As proved in [BBKK97], accessing nodes in increasing order of $d_I^{min}(Q, Reg(N))$ leads to minimal I/O costs, as compared to other scheduling criteria (e.g. the one proposed in [RKV95]).

The `RL` (Result List) array, with k entries of type $[O_j, d_I(Q, O_j)]$, is used to store the k closest objects found so far by the algorithm. At the beginning of the search, when less than k objects have been retrieved, missing entries are set to $[\_, \infty]$. The largest of the k distances in `RL`, denoted ϵ_k, is used as a *dynamic search threshold* for the pruning criterion of Equation 4. This means that if $d_I^{min}(Q, Reg(N)) \geq \epsilon_k$ holds for node N, then N can be pruned from the search, since it cannot contain any object closer to Q than the current k-th NN in `RL`. In particular, when this is verified for the first node in the `PQ` queue the search is interrupted (line 5. of the algorithm). Also note that because exactly k objects are to be returned, nodes can also be pruned even when their minimum distance from Q equals ϵ_k (whereas for range queries this would not be correct). Clearly, the same holds for objects (see lines 8. and 10.).

Algorithm `k-NNSearch` (query object Q, integer k)

```
1.  Initialize PQ with a pointer to the root node of the M-tree;
2.  Let RL[j] = [_, ∞], j = 1, ..., k; Let ε_k = ∞;
3.  While PQ ≠ ∅ do:
4.      Extract the first entry from PQ, referencing node N;
5.      If d_I^min(Q, Reg(N)) ≥ ε_k then exit, else read N;
6.      If N is a leaf node then:
7.          For each object O_i in N do:
8.              If |d_I(Q, O^[N]) − d_I(O^[N], O_i)| < ε_k then:
9.                  Compute d_I(Q, O_i);
10.                 If d_I(Q, O_i) < ε_k then:
11.                     Update RL performing an ordered insertion of [O_i, d_I(Q, O_i)];
12.                     Update ε_k;
13.     else: // N is an internal node
14.         For each child node N_c of N do:
15.             If |d_I(Q, O^[N]) − d_I(O^[N], O^[N_c])| < ε_k + r^[N_c] then:
16.                 Compute d_I(Q, O^[N_c]);
17.                 If d_I(Q, O^[N_c]) < ε_k + r^[N_c] then:
18.                     Update PQ performing an ordered insertion of [ptr(N_c), d_I^min(Q, Reg(N_c))];
19. End.
```

Figure 5: Optimal algorithm for k-NN search.

Application of the QIC Theorem to k-NN search leads to the `QIC-k-NNSearch` algorithm shown in Figure 6. The major differences with the basic case are:

- The `RL` array keeps *actual* (i.e. query) distance values from Q. Consequently, the ϵ_k threshold is set using query/user distances.
- Lines 10. and 19. both perform a cheap filter test using the comparison distance d_C, as much as done in the `QIC-RangeSearch` algorithm.

Corollary 2
The `QIC-k-NNSearch` algorithm returns the correct answer to the query $\texttt{NN}(\mathcal{O}, Q, k, d_Q)$. □

Proof: Correctness directly follows from the QIC Theorem, which applies here with $\epsilon = \epsilon_k$. □

5.1 Cost Model

Results presented in Section 4.1 can also be exploited for predicting search performance of k-NN queries. The number of each operation type is obtained by integrating the values given by Equations 6, 8, 9, and 10

Algorithm `QIC-k-NNSearch` (query object Q, distances d_Q, d_C, integer k, scaling factors $S_{C\to I}$, $S_{C\to Q}$, $S_{I\to Q}$)

1. Initialize `PQ` with a pointer to the root node of the M-tree;
2. Let `RL`$[j] = [\_, \infty]$, $j = 1, \ldots, k$; Let $\epsilon_k = \infty$;
3. While `PQ` $\neq \emptyset$ do:
4. Extract the first entry from `PQ`, referencing node N;
5. If $d_I^{min}(Q, Reg(N)) \geq S_{I\to Q}\ \epsilon_k$ then exit, else read N;
6. If N is a leaf node then:
7. For each object O_i in N do:
8. If $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O_i)| < S_{I\to Q}\ \epsilon_k$ then:
9. Compute $d_C(Q, O_i)$;
10. If $d_C(Q, O_i) < S_{C\to Q}\ \epsilon_k$ then:
11. Compute $d_Q(Q, O_i)$;
12. If $d_Q(Q, O_i) < \epsilon_k$ then:
13. Update `RL` performing an ordered insertion of $[O_i, d_Q(Q, O_i)]$;
14. Update ϵ_k;
15. else: // N is an internal node
16. For each child node N_c of N do:
17. If $|d_I(Q, O^{[N]}) - d_I(O^{[N]}, O^{[N_c]})| < S_{I\to Q}\ \epsilon_k + r^{[N_c]}$ then:
18. Compute $d_C(Q, O^{[N_c]})$;
19. If $d_C(Q, O^{[N_c]}) < S_{C\to I}(S_{I\to Q}\ \epsilon_k + r^{[N_c]})$ then:
20. Compute $d_I(Q, O^{[N_c]})$;
21. If $d_I(Q, O^{[N_c]}) < S_{I\to Q}\ \epsilon_k + r^{[N_c]}$ then:
22. Update `PQ` performing an ordered insertion of $[ptr(N_c), d_I^{min}(Q, Reg(N_c))]$;
23. End.

Figure 6: The QIC-k-NNSearch algorithm.

over all possible values of ϵ, each weighted by the probability $p_k(\epsilon)$ that the distance of the k-th NN is equal to ϵ. The probability $P_k(\epsilon)$ that such distance is lower than ϵ is given in [CPZ98a] as

$$P_k(\epsilon) = 1 - \sum_{i=0}^{k-1} \binom{M}{i} F_{\mathbf{d_Q}}(\epsilon)^i (1 - F_{\mathbf{d_Q}}(\epsilon))^{M-i} \tag{13}$$

The density function can be obtained by taking the derivative of Equation 13:

$$p_k(\epsilon) = \frac{d}{d\epsilon} P_k(\epsilon) = \sum_{i=0}^{k-1} \binom{M}{i} F_{\mathbf{d_Q}}(\epsilon)^{i-1} f_{\mathbf{d_Q}}(\epsilon)(1 - F_{\mathbf{d_Q}}(\epsilon))^{M-i-1}(M \cdot F_{\mathbf{d_Q}}(\epsilon) - i) \tag{14}$$

In order to simplify the evaluation of cost formulae, we can consider a k-NN query as a range query with radius $E[\epsilon_{\mathbf{k}}]$ equal to the expected distance between Q and its k-th nearest neighbor. This can be estimated by taking into account the pdf $p_k(\epsilon)$ given in Equation 14. Therefore, it is

$$\begin{aligned} E[\epsilon_{\mathbf{k}}] &= \int_0^{d_Q^+} \epsilon \cdot p_k(\epsilon)\, d\epsilon = |\epsilon\ P_k(\epsilon)|_0^{d_Q^+} - \int_0^{d_Q^+} P_k(\epsilon)\, d\epsilon = d_Q^+ - \int_0^{d_Q^+} P_k(\epsilon)\, d\epsilon = \\ &= d_Q^+ - \int_0^{d_Q^+} \left(1 - \sum_{i=0}^{k-1} \binom{M}{i} F_{\mathbf{d_Q}}(\epsilon)^i (1 - F_{\mathbf{d_Q}}(\epsilon))^{M-i}\right) d\epsilon = \\ &= d_Q^+ - d_Q^+ + \int_0^{d_Q^+} \sum_{i=0}^{k-1} \binom{M}{i} F_{\mathbf{d_Q}}(\epsilon)^i (1 - F_{\mathbf{d_Q}}(\epsilon))^{M-i} d\epsilon = \\ &= \int_0^{d_Q^+} \sum_{i=0}^{k-1} \binom{M}{i} F_{\mathbf{d_Q}}(\epsilon)^i (1 - F_{\mathbf{d_Q}}(\epsilon))^{M-i} d\epsilon \end{aligned} \tag{15}$$

Then, formulae obtained for range queries can be directly applied by using $\epsilon = E[\epsilon_{\mathbf{k}}]$. Of course, in this case, the cost for a k-NN query does depend on the distance distribution of d_Q over the considered data set $\mathcal{O}$, since such distribution influences the value of $E[\epsilon_{\mathbf{k}}]$.

6 Performance Evaluation

The purpose of this Section is to investigate the actual performance of QIC-M-tree as well as to validate the proposed cost models. The experimental results we provide are all obtained from *real* data sets, which are also synthetically described in Table 2, together with the distances used in the experiments:

`Airphoto60`: This data set consists of 274,424 60-dimensional vectors. Each vector contains texture information extracted from a tile of size 64×64 that is part of a large aerial photograph (there are 40 airphotos in the data set). Each tile is analyzed by means of 30 Gabor filters, and for each filter the mean and the standard deviation of the output are stored in the feature vector. This data set was given us by B.S. Manjunath [Man] and was also used in [CP00].

`Airphoto8`: This is obtained from the previous data set by projecting vectors on the first 8 coordinates.

`BibleWords`: It consists of all the 12,569 distinct words occurring in the English King James version of the Holy Bible (as provided by [Gut]).

`BibleLines`: The 74,645 variable-length lines of the Holy Bible.

`BibleLines20`: We took the Holy Bible and segmented it into lines of length 20, which resulted in a total of 161,212 lines.

`Corel`: This data set contains 68,040 32-dimensional color histograms extracted from a collection of Corel images. The value for each dimension in the histogram represents the density of the relative color in the entire image. This data set was first used in [ORC+98] and we downloaded it from [KDD].

Name	No. of objects	Dimensionality	d_Q	d_I	d_C
`Airphoto60`	274,141	60	L_p	L_2, L_p	$L_2[D']$
`Airphoto8`	274,141	8	L_p, $d_{qf}[A]$	L_2, L_p	–
`BibleWords`	12,569	–	$d_{edit}[\gamma]$	d_{edit}	d_{ms}
`BibleLines`	74,645	–	$d_{edit}[\gamma]$	d_{edit}	d_{ms}
`BibleLines20`	161,212	–	$d_{edit}[\gamma]$	d_{edit}	d_{ms}
`Corel`	67,358	32	$d_{qf}[A]$	L_2, $d_{qf}[A]$	–

Table 2: Data sets.

We ran all the experiments on a Linux PC with a Pentium III 450 MHz processor, 256 MB of main memory, and a 9 GB disk. The node size of the QIC-M-trees was always set to 8 Kbytes. For each experiment we average results obtained over all executed queries. The performance measures we are interested in are those in Equations 5 (I/O and CPU costs, and number of corresponding operations). From these, other performance measures are derived:

- The number of computed query distances, normalized with respect to the overall number of indexed objects M, is called *query distance selectivity*, sel_{d_Q}:

$$sel_{d_Q} = \frac{\#_{d_Q}(\epsilon; d_I, d_C; S_{I \to Q}, S_{C \to Q})}{M}$$

- In a similar way, the *index distance selectivity*, sel_{d_I}, and the *I/O selectivity*, $sel_{I/O}$, are defined as the number of computed d_Is and fetched nodes, respectively, divided by the total number of nodes, i.e.

$$sel_{d_I} = \frac{\#_{d_I}(\epsilon; d_I, d_C; S_{I \to Q}, S_{C \to I})}{n}$$
$$sel_{I/O} = \frac{\#_{I/O}(\epsilon; d_I; S_{I \to Q})}{n}$$

- When $d_Q \neq d_I$ we expect that performance deteriorates with respect to the case $d_Q \equiv d_I$. The *(percent) cost overhead*, oh_{pm}, measures such degradation of performance, where pm is any of the performance indicators in Equations 5:

$$oh_{pm} = \frac{pm(\langle d_Q \neq d_I \rangle) - pm(\langle d_Q \equiv d_I \rangle)}{pm(\langle d_Q \equiv d_I \rangle)} \, (*100) \tag{16}$$

- When using a cheap comparison distance d_C, we expect that index performance would improve with respect to the case where d_C is not used. The *(percent) cost saving*, sav_{pm}, measures performance improvement:

$$sav_{pm} = \frac{pm(\langle d_C \text{ is not used} \rangle) - pm(\langle d_C \text{ is used} \rangle)}{pm(\langle d_C \text{ is not used} \rangle)} \, (*100) \tag{17}$$

6.1 Validation of the Model

Our first experiments aim to verify the accuracy of the cost model proposed in Section 4.1. To this end, we provide comparison between predicted and real costs for range queries with increasing search radius on three data sets: The `AirPhoto8` data set with $d_I \equiv L_2$ and $d_Q \equiv L_1$, the `Corel` data set with $d_I \equiv L_2$ and a quadratic query distance function, and the `BibleWords` data set with $d_I \equiv d_{edit}$ and with a weighted edit query distance.[7] Graphs in Figures 7, 8, and 9 show estimated and actual costs as a function of the query selectivity (i.e. the fraction of indexed objects retrieved by the query) and demonstrate that the cost model indeed succeeds in predicting search costs, with errors that rarely exceeds 10%-20%. These values are particularly good, especially if one considers the minimal amount of statistics on the tree structure needed by the cost model.

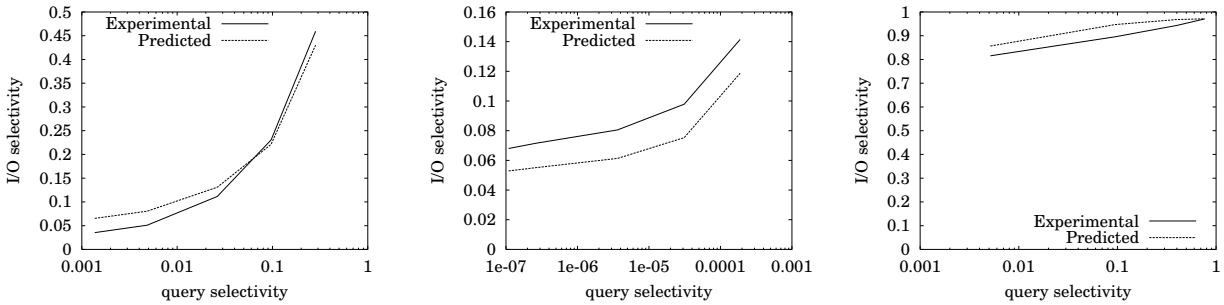


Figure 7: Real and predicted I/O selectivity for the `AirPhoto8` (a), the `Corel` (b), and the `BibleWords` (c) data sets.

To show the accuracy of the model also when a comparison distance is used, we consider $d_Q \equiv d_I \equiv d_{edit}$ and $d_C \equiv d_{ms}$ for the three `Bible` data sets. Figure 10 shows predicted and actual values of sel_{d_Q}, plotted against the query selectivity. Regardless of whether or not d_C is used, errors are minimal along all the range of considered query selectivities.

[7] Experiments on k-NN queries exhibit similar results and are omitted here for brevity.

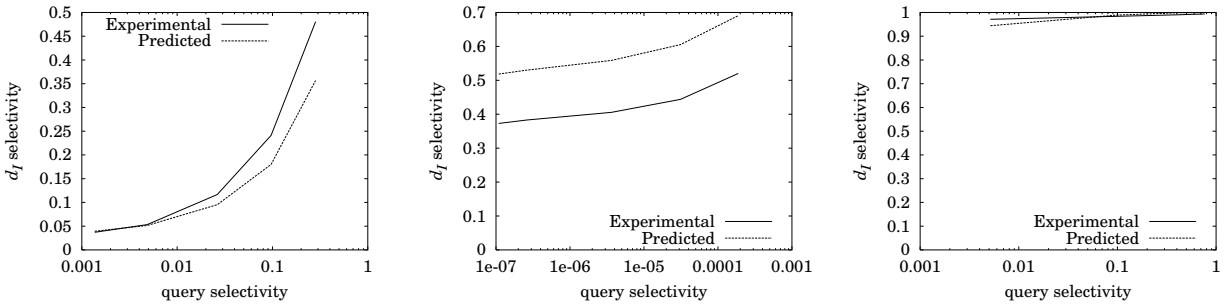


Figure 8: Real and predicted index distance selectivity for the `AirPhoto8` (a), the `Corel` (b), and the `BibleWords` (c) data sets.

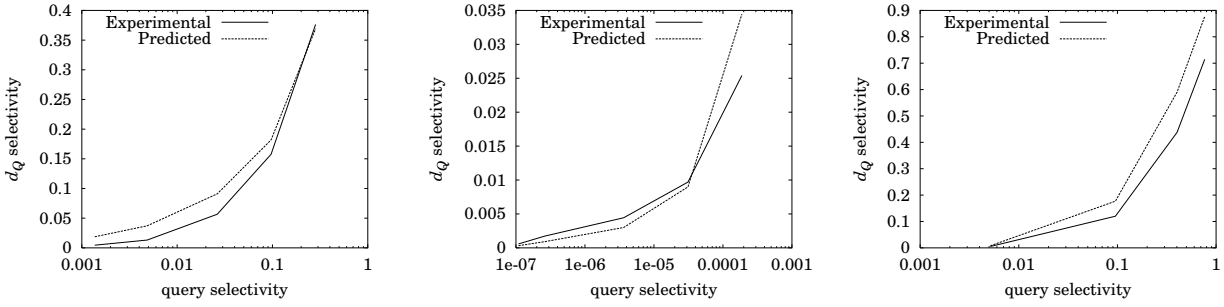


Figure 9: Real and predicted query distance selectivity for the `AirPhoto8` (a), the `Corel` (b), and the `BibleWords` (c) data sets.

6.1.1 Predicting the Effectiveness of a Comparison Distance

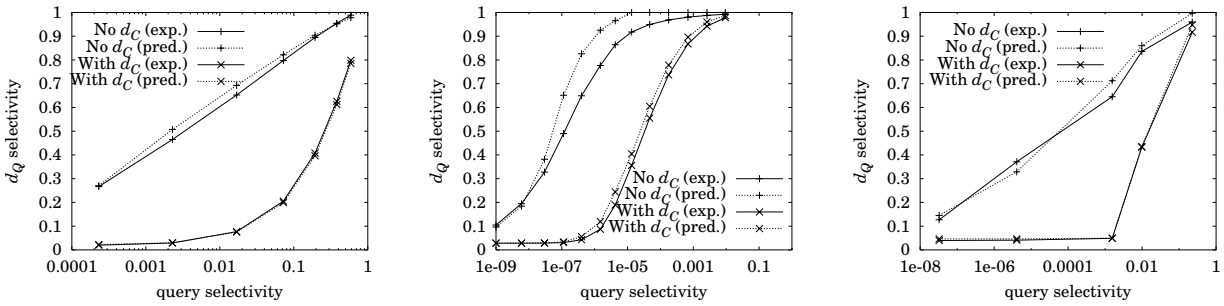
Our last experiment suggests that the cost model can also be used to predict the saving in search costs that can be achieved when using a comparison distance d_C. Indeed, from Equations 10 and 12, it follows

$$
\begin{aligned}
sav_{d_Q} &= \frac{\#_{d_Q}(\epsilon; d_I, \_; S_{I\to Q}, 1) - \#_{d_Q}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to Q})}{\#_{d_Q}(\epsilon; d_I, \_; S_{I\to Q}, 1)} = \\
&= \frac{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) - M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \cdot F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon)}{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon)} = \\
&= \frac{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon)\ (1 - F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon))}{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon)}
\end{aligned}
$$

which leads to

$$
\boxed{sav_{d_Q} = 1 - F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon)} \tag{18}
$$

Considering CPU costs, an easy-to-evaluate formula based on Equation 18 can be obtained on the assumptions that $cost_{d_I} \approx cost_{d_Q}$ and $\#_{d_I} \ll \#_{d_Q}$ (i.e. the number of distance computed at higher levels of the tree is negligible with respect to the number of distance computations performed at leaf level). It then follows that $time_{CPU} \approx time_{d_Q} + time_{d_C}$, and the saving in CPU time, sav_{CPU}, can be estimated from



(a) (b) (c)

Figure 10: Real and predicted query distance selectivity with and without d_{ms} for the `BibleWords` (a), `BibleLines20` (b), and `BibleLines` (c) data sets.

Equations 8, 10, and 12 as

$$
\begin{aligned}
sav_{CPU} &= \frac{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) - time_{CPU}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q}, S_{C\to I})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1)} \approx \\
&\approx 1 - \frac{time_{d_Q}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q}) + time_{d_C}(\epsilon; d_I, d_C; S_{I\to Q})}{time_{d_Q}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1)} = \\
&= 1 - \frac{\#_{d_Q}(\epsilon; d_I, d_C; S_{I\to Q}, S_{C\to Q}) \cdot cost_{d_Q} + \#_{d_C}(\epsilon; d_I; S_{I\to Q}) \cdot cost_{d_C}}{\#_{d_Q}(\epsilon; d_I, \_; S_{I\to Q}, 1) \cdot cost_{d_Q}} = \\
&= 1 - \frac{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \cdot F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon) \cdot cost_{d_Q}}{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \cdot cost_{d_Q}} \\
&\quad - \frac{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \cdot cost_{d_C}}{M \cdot F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + \overline{r_L}) \cdot F_{\mathbf{z}}(S_{I\to Q}\ \epsilon) \cdot cost_{d_Q}}
\end{aligned}
$$

Thus

$$
\boxed{sav_{CPU} \approx 1 - F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon) - \frac{cost_{d_C}}{cost_{d_Q}} \qquad (\text{when } \#_{d_I} \ll \#_{d_Q})} \tag{19}
$$

from which we obtain that, in order to reduce search costs, the comparison distance d_C should be cheap (in order to minimize the ratio $cost_{d_C}/cost_{d_Q}$), but also selective enough for the considered query radius ϵ to reduce the term $F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon)$ in Equation 19. These two requirements are often contrasting, thus reducing the time complexity of d_C will lead to a "looser" approximation of d_Q (see also Section 6.1.2).

From Equation 19 we can also easily compute the maximum search radius, ϵ_{max}, for which it is convenient to use the d_C comparison distance: Such value can be obtained when the saving equals 0. Supposing for simplicity that $F_{\mathbf{d_C}}$ is invertible, it is:

$$
\boxed{\epsilon_{max} \approx \frac{F^{-1}_{\mathbf{d_C}}(1 - cost_{d_C}/cost_{d_Q})}{S_{C\to Q}}} \tag{20}
$$

Finally, since I/O costs do not change when using d_C, the saving in overall time will have the same behavior of the CPU saving, although scaled by a factor equal to

$$
\beta = \frac{time_{CPU}}{time_{CPU} + time_{I/O}}
$$

where $time_{CPU}$ refers to the case where d_C is not used, that is $time_{CPU}(\epsilon; d_I, \_; S_{I\to Q}, 1, 1)$:

$$
\boxed{sav_{total} = \beta \cdot sav_{CPU}} \tag{21}
$$

It could be argued that since in Equations 18, 19, and 20 no tree statistics are used, *above results do not depend on the actual tree*, but only on the considered data set.

Theorem 3
For any metric access method, whenever it is $time_{d_I} \ll time_{d_Q}$*, the saving in CPU time for solving a range query* $\texttt{range}(\mathcal{O}, Q, \epsilon, d_Q)$ *when using a comparison distance* d_C *is given by Equation 19. On the other hand, the saving in overall time can be computed from Equation 21, with the* β *parameter depending on the specific MAM.* □

Proof: Given in Appendix B. □

To evaluate the accuracy of these approximations of the model, we consider the `Bible` data sets with $d_Q \equiv d_I \equiv d_{edit}$ and $d_C \equiv d_{ms}$ (Table 3 reports the average cost for computing such distances).

Data set	$cost_{d_{ms}}$ (s)	$cost_{d_{edit}}$ (s)
`BibleWords`	7.78×10^{-6}	26.1×10^{-6}
`BibleLines20`	17.7×10^{-6}	231.6×10^{-6}
`BibleLines`	66.1×10^{-6}	1300×10^{-6}

Table 3: Distance computation times for the `Bible` data sets.

Graphs in Figure 11 show predicted and real values of the CPU saving, along with the value obtained from Equation 19, where the cost due to the index distance d_I is ignored. Besides demonstrating the accuracy of the cost model, the graphs also show that, when the query radius ϵ is low, costs due to d_I indeed reduce the saving, since they are similar to costs due to d_Q; when the search radius increases, however, $time_{d_I}$ has a minor impact, and the CPU saving can be very well estimated using Equation 19.

Figure 11 also proves the reliability of Equation 20. In particular, by using the corresponding distance distributions (not shown here for brevity) and the distance computation times in Table 3, it is obtained

- $\epsilon_{max}(\texttt{BibleWords}) = F^{-1}_{\mathbf{d_{ms}}}(1 - 7.78/26.1) \approx F^{-1}_{\mathbf{d_{ms}}}(0.703) \approx 6$ (for which the query selectivity is $F_{\mathbf{d_{edit}}}(\epsilon_{max}) = 0.38$),
- $\epsilon_{max}(\texttt{BibleLines20}) = F^{-1}_{\mathbf{d_{ms}}}(1 - 17.7/232) \approx F^{-1}_{\mathbf{d_{ms}}}(0.924) \approx 11$ ($F_{\mathbf{d_{edit}}}(\epsilon_{max}) = 0.0018$), and
- $\epsilon_{max}(\texttt{BibleLines}) = F^{-1}_{\mathbf{d_{ms}}}(1 - 66.1/1300) \approx F^{-1}_{\mathbf{d_{ms}}}(0.949) \approx 39$, ($F_{\mathbf{d_{edit}}}(\epsilon_{max}) = 0.17$).

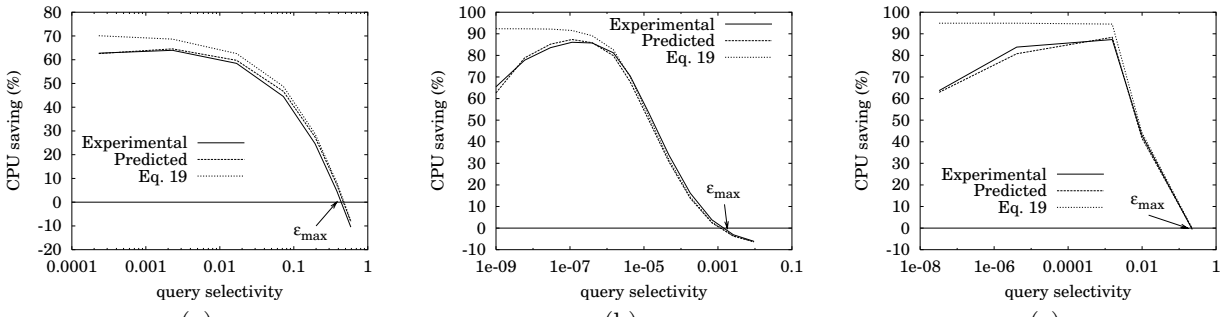


Figure 11: Real and predicted CPU saving for the `BibleWords` (a), `BibleLines20` (b), and `BibleLines` (c) data sets. Also shown are the ϵ_{max} values which limit the usefulness of using d_C.

6.1.2 Choosing the Best Comparison Distance

Another issue worth investigating is how to choose the most suitable comparison distance when many of them are available. We study the problem with reference to the `AirPhoto60` data set with $d_Q \equiv d_I \equiv L_2[D = 60]$ and $d_C \equiv L_2[D']$, i.e. the Euclidean distance computed only on the first $D' < D$ dimensions.[8] In this way, we are able to vary the computation time of d_C along with the similarity between the distance distributions $F_{\mathbf{d_Q}}$ and $F_{\mathbf{d_C}}$. Intuitively, the distance computation time should linearly increase with D', whereas the difference between distance distributions should reduce. This is confirmed by Figure 12, where the distance distributions and average computation times are plotted for different values of D'. In Figure 12 (a), the distance distribution $F_{\mathbf{L_2[D']}}(x)$ is plotted as a function of the distance distribution $F_{\mathbf{L_2[D]}}(x)$, i.e. of the correct query distance, to emphasize differences between the distance distributions for different values of x.

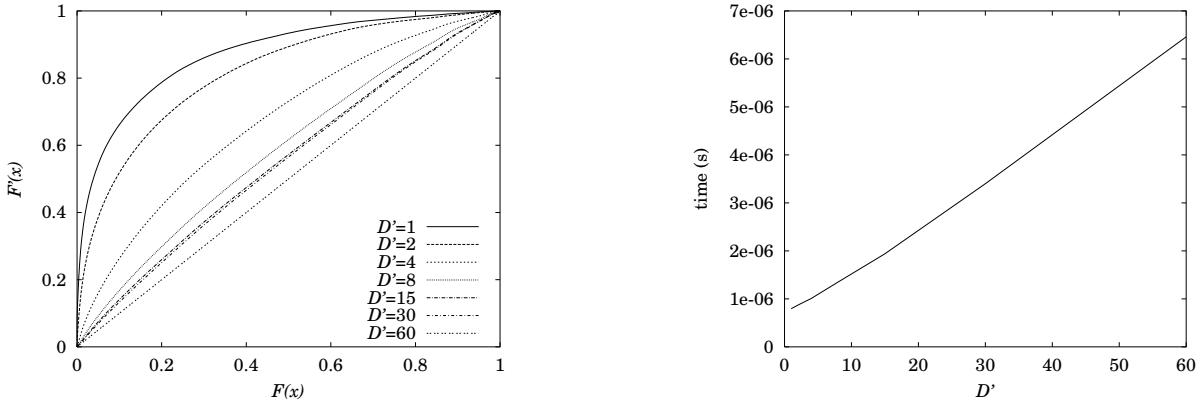


Figure 12: Distance distribution (a) for different comparison distances with respect to the L_2 metric and average computation time (b) as a function of the number D' of considered dimensions.

We execute several range queries, by varying the search radius ϵ, and analyze the saving in CPU times, sav_{CPU}, for several values of D'. Figure 13 shows the "CPU saving loss", i.e. the amount of saving which is lost when using a non-optimal D' in place of the optimal D' for that query selectivity. In our experiments it turns out that the optimal number of dimensions to be retained varies between 6 and 10, depending on the query selectivity. As Figure 13 clearly demonstrates, choosing D' far from the optimal values can considerably reduce the effectiveness of using d_C (e.g. see the curve for $D' = 30$). Predictions yielded by Equation 19 are almost always able to select the optimal D' and, whenever this is not the case, they never lead to a CPU saving loss higher than 2%.

6.2 Querying with User-Defined Functions

In this Section we report experimental results for the case when $d_Q \neq d_I$, and compare the performance of `QIC-RangeSearch` algorithm (exemplified in Figure 14 (a)) with results obtained from two alternate approaches:

1. The first approach (which we call "unfeasible", UNF for short) consists in building from scratch an M-tree for each new query by using the current d_Q distance (Figure 14 (b)). In some sense, UNF provides the reference performance level against which to compare `QIC-RangeSearch` performance.

2. The second approach is indeed a filter & refine (F&R) variant of the QIC solution. In this case a "classic" M-tree built using d_I is queried with the query $\texttt{range}(\mathcal{O}, Q, S_{I \to Q}\ \epsilon, d_I)$, thus scaling the query threshold, after that a second refinement step discards the resulting false drops (Figure 14 (c)).

[8] Of course, when $D' < D$ it is $L_2[D'](O_1, O_2) \leq L_2[D](O_1, O_2)$.

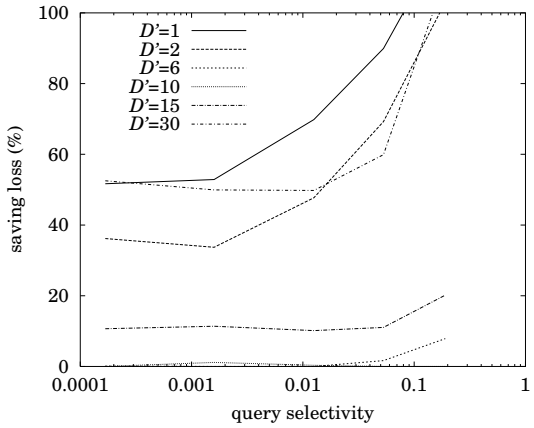


Figure 13: CPU saving loss for several D' as a function of the query selectivity.

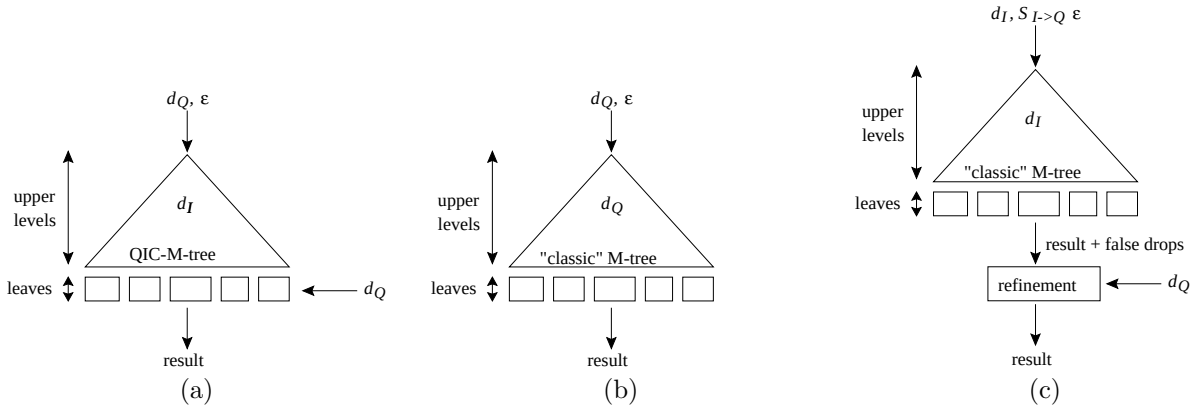


Figure 14: Three possible solutions for querying an M-tree with a user-defined function: (a) QIC, (b) UNF, and (c) F&R.

6.2.1 Comparing the Distance Functions

The goal of this set of experiments is to investigate how search performance is influenced when $d_I \neq d_Q$. Intuitively, the more d_Q and d_I are "similar", the lower the search costs needed to solve a query with d_Q using a QIC-M-tree built on d_I. This leads us to define a measure able to quantify how much a distance function is similar to (i.e. how well it approximates) another distance function. Until now, the problem has been addressed only for specific cases, thus leading to solutions that cannot be generalized to arbitrary distance functions. For instance, in [ABKS98] the authors measure the quality of approximating ellipsoid queries with simpler geometric figures by introducing the concept of ellipsoid *sphericity*. In [SYKU01], the *flatness* of an ellipsoid, computed as the variance of eigenvalues of the query matrix A, is used to discriminate between different query distances.

To start with, consider two distance functions d_1 and d_2 such that $d_1 \preceq d_2$. In order to define how much d_1 and d_2 differ, it is important that we refer our measure to a specific data set $\mathcal{O}$. This is because the effect of using d_1 in place of d_2 will vary depending on the distribution of the indexed objects, as also confirmed by analytical results [CPZ98a]. For arbitrary distance functions we therefore consider the corresponding distance distributions they induce on $\mathcal{O}$. Note that since $d_1 \preceq d_2$, it is $d_1^+ \leq d_2^+$ as well as $F_{\mathbf{d_1}}(x) \geq F_{\mathbf{d_2}}(x)$ $\forall x \in [0, d_2^+]$.

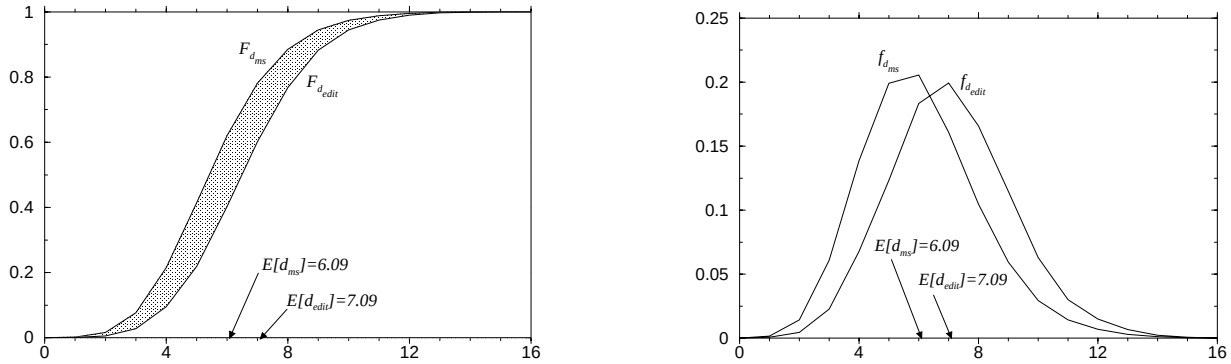
The error measure we propose to compare d_1 and d_2 corresponds to the (normalized) integral difference between $F_{\mathbf{d_1}}(\cdot)$ and $F_{\mathbf{d_2}}(\cdot)$.

Definition 5 (Error Measure for Distance Functions)
The normalized error measure, $Err(d_1|d_2)$, of distance function d_1 with respect to d_2 on a data set $\mathcal{O}$ (for simplicity, the dependency on $\mathcal{O}$ is understood in the notation) is:

$$Err(d_1|d_2) = \frac{1}{d_2^+} \int_0^{d_2^+} (F_{\mathbf{d_1}}(x) - F_{\mathbf{d_2}}(x))\, dx \tag{22}$$

where it is assumed that $d_1 \preceq d_2$. Note that $0 \leq Err(d_1|d_2) \leq 1$ and $Err(d_1|d_2) = 0$ iff $F_{\mathbf{d_1}}(\cdot) = F_{\mathbf{d_2}}(\cdot)$. □

Figure 15 (a) refers to the case where $d_1 \equiv d_{ms}$, $d_2 \equiv d_{edit}$, and distances refer to the `BibleWords` data set. From Definition 5 it follows that $Err(d_{ms}|d_{edit})$ equals the normalized volume of the shaded area in the figure.



(a) (b)

Figure 15: The distance distributions (a) and pdf's (b) of d_{ms} and d_{edit} for the `BibleWords` data set.

The following result shows that $Err(d_1|d_2)$ can be indeed easily expressed in terms of the expected values of the $\mathbf{d_1}$ and $\mathbf{d_2}$ random variables.

Lemma 2
If $\mathbf{d_1}$ ($\mathbf{d_2}$) is distributed according to $F_{\mathbf{d_1}}(\cdot)$ (respectively $F_{\mathbf{d_2}}(\cdot)$), then:

$$Err(d_1|d_2) = \frac{E[\mathbf{d_2}] - E[\mathbf{d_1}]}{d_2^+} \tag{23}$$

□

Proof: Given in Appendix B. □

Similar arguments show that when $d_1 \preceq S_{1\to 2}\, d_2$ it is:

$$Err(d_1|d_2) = \frac{E[\mathbf{d_2}] - E[\mathbf{d_1}]/S_{1\to 2}}{d_2^+} \tag{24}$$

In Figure 15 (b) we plot the pdf's of d_{ms} and d_{edit} for the `BibleWords` data set. From Lemma 2 it is $Err(d_{ms}|d_{edit}) = (7.09 - 6.09)/16 \approx 0.0627$.

The above error measure is by no means the only possible one to compare two distributions. With respect to more complex measures, such as the *Kullback-Leibler distance*,[9] $Err(d_1|d_2)$ has however the advantages of having low computational costs and an extremely easy-to-grasp interpretation.

Turning back to our scenario, the specific error measure we consider is defined as follows.

[9]The Kullback-Leibler distance [CT91], also called *relative entropy*, of $f_{\mathbf{1}}(\cdot)$ with respect to $f_{\mathbf{2}}(\cdot)$ is defined as $\int_x f_{\mathbf{2}}(x) \log \frac{f_{\mathbf{2}}(x)}{f_{\mathbf{1}}(x)}\, dx$, and provides a measure of how much information we get about $f_{\mathbf{2}}(\cdot)$ if we know $f_{\mathbf{1}}(\cdot)$.

Definition 6 (Indexing Error)
The normalized indexing error *with respect to a data set $\mathcal{O}$ when using an index distance $d_I \preceq S_{I \to Q}\, d_Q$ is:*

$$Err_I \stackrel{\text{def}}{=} Err(d_I|d_Q) = \frac{E\left[\mathbf{d_Q}\right] - E\left[\mathbf{d_I}\right] / S_{I \to Q}}{d_Q^+} \tag{25}$$

□

6.2.2 The Effect of Different Distance Distributions

To investigate the effect of the difference in distance distributions between d_Q and d_I, as well as the relevance of the indexing error measure, we measure performance degradation when the query distance is dissimilar with respect to the index distance, thus we keep d_I fixed and vary d_Q. In the first set of experiments we index the `BibleWords` data set using $d_I \equiv d_{edit}$ and query the QIC-M-tree with different $d_Q \equiv d_{edit}[\gamma]$ functions. Since computation times for d_{edit} and $d_{edit}[\gamma]$ are the same, differences in performance only depend on the distance distributions. We contrast results obtained from the QIC algorithm with the UNF approach.

To construct different weighing functions γ, all edit operations are initially given a unitary cost, then this default situation is perturbed by giving to some cost the value 2. The fraction of edit operations having cost 2, therefore, characterizes each distance function.[10] Note that, for all considered weight functions, it is $S_{I \to Q}[\gamma] = 1$. Table 4 shows the average distance values and the indexing errors for different $d_{edit}[\gamma]$ functions.

Fraction of 2 in γ	$E\left[\mathbf{d_Q}\right]$	Err_I
0	7.093	6×10^{-5}
0.25	8.50	0.044
0.5	9.90	0.088
0.75	11.6	0.14
1	14.2	0.22

Table 4: Indexing error values for different metrics with respect to the unweighted edit distance for the `BibleWords` data set ($E\left[\mathbf{d_{edit}}\right] = 7.092$, $d_Q^+ = 32$).

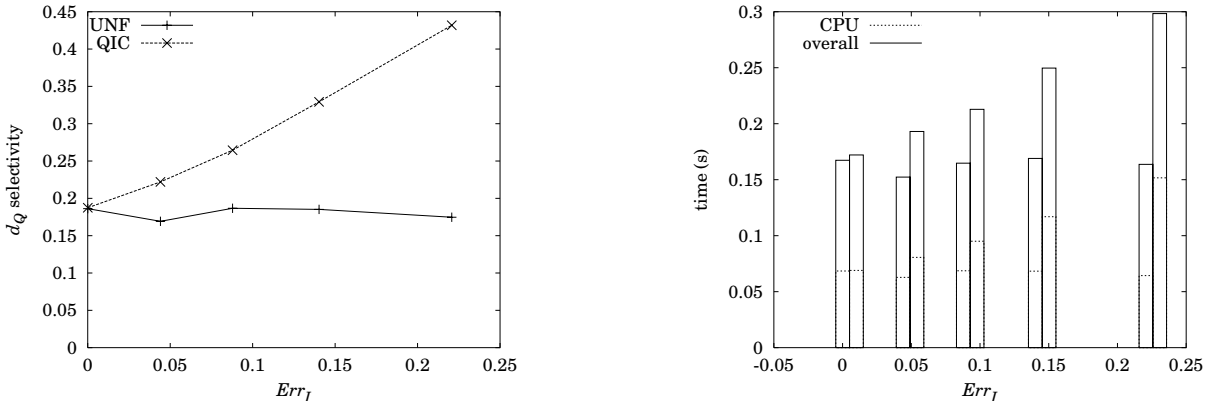
Figure 16 (a) shows the query distance selectivity sel_{d_Q} as a function of the indexing error Err_I for a 1-NN query for the QIC and the UNF solutions, whereas Figure 16 (b) reports actual search times. It can be seen that the search cost is almost the same for different weighing functions when using the UNF approach, whereas it is linearly correlated with Err_I when using QIC. Indeed, when costs of edit operations increase, the distribution $F_{\mathbf{d_Q}}$ shifts toward the right, thus the indexing error Err_I grows; therefore, the expected distance between Q and its 1-NN increases as well and so necessarily do search costs.

The inverse correlation between search performance and the indexing error is also confirmed by the second series of experiments we present. Here we consider the `AirPhoto8` data set with $d_I \equiv L_2$ and query the index with differently weighted Euclidean distances. To construct different weighing functions, we assign a weight higher than 1 to an increasingly number of dimensions. Average distance values and indexing errors are shown in Table 5 for each query distance.

In Figure 17 (a) the cost overhead for solving a 1-NN query is plotted as a function of the indexing error. Again, we see that a direct correlation between performance degradation and indexing error exists: The higher the difference between d_Q and d_I, the higher search costs when using a tree built on d_I to solve a query with d_Q.

The fact that search performance increases when the index and the query distance are more similar is also demonstrated by recent studies conducted for SAMs. In [SYKU01], the Spatial Transformation Technique is introduced to decrease CPU costs for SAMs when queried with quadratic form distances, and experimental studies show how search performance deteriorates as the *flatness* (see Section 6.2.1) of the

[10] For simplicity, we refer to the situation of only one operation (resp. all operations except one) having cost 2 as 0 (resp. 1).



(a) (b)

Figure 16: Query distance selectivity (a) and overall costs (b) for 1-NN queries as a function of Err_I for the `BibleWords` data set with different querying functions with respect to the d_{edit} index distance.

No. of weights > 1	$E[\mathbf{d_Q}]$	Err_I
1	45.08	0.023
2	49.63	0.036
3	54.67	0.050
4	63.20	0.075
5	66.78	0.085
6	70.15	0.095
7	73.32	0.104

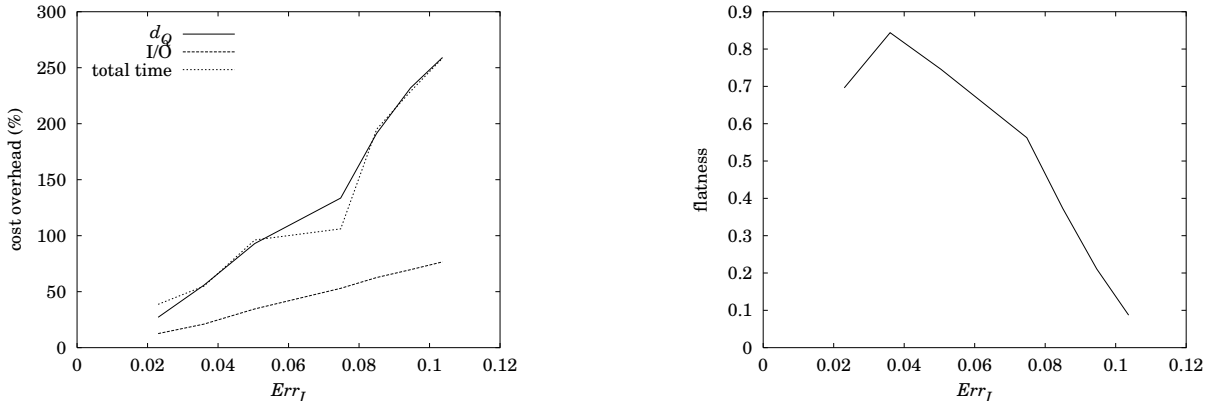
Table 5: Indexing error values for different metrics with respect to the L_2 distance for the `AirPhoto8` data set ($E[\mathbf{L_2}] = 37.03$, $d^+_{L_2} = 350$).

query matrix increases; intuitively, as the query ellipsoid flattens, its relative distance distribution becomes more and more different with respect to that of the Euclidean distance, which is taken as the "default" index distance. It has to be noted that the flatness measure is independent of axis rotation, thus it cannot capture performance deterioration due to differences in distance distributions. On the other hand, the indexing error does not take into account information on space coordinates, thus it is able to reflect rotations of the query ellipsoid (see Figure 18). This is confirmed by Figure 17 (b), where the correlation between the flatness and the indexing error is plotted for the query distances used in the previous experiments. Moreover, in this case, it has to be noted that *performance is reduced with decreasing flatness values*, contrary to intended purposes of the flatness measure.[11]

6.2.3 The Effect of Distance Computation Costs

Although the "unfeasible" (UNF) solution, for which by definition it is $d_I \equiv d_Q$, apparently represents the most favorable situation to deal with, indexing objects with a d_I cheaper to evaluate than d_Q could in principle lead to a better performance. This can be the case when the increase in the number of distances to be evaluated is more than compensated by the reduction in the cost of a single distance computation. To this end, in this second series of experiments we index the `Corel` data set with $d_I \equiv L_2$ and query the index with four d_Qs having different computation costs. The query distances, for which costs and indexing errors are given in Table 6, are:

[11] It has to be noted that results in [SYKU01] are however correct, since in that paper the flatness of query ellipsoids is varied only along certain fixed axes.



(a) (b)

Figure 17: Cost overhead as a function of the indexing error (a) and correlation between flatness and indexing error (b).

Figure 18: The two query ellipsoids have the same flatness value, but, clearly, different distance distributions with respect to the considered data set.

L_{random}**:** A weighted Euclidean distance, with randomly chosen weights.

L_σ**:** A weighted Euclidean distance, where the weight of each coordinate is equal to the inverse of the standard deviation of the data set values for that coordinate.

A_{random}**:** A quadratic function, where the A matrix is a randomly chosen positive definite symmetric one.

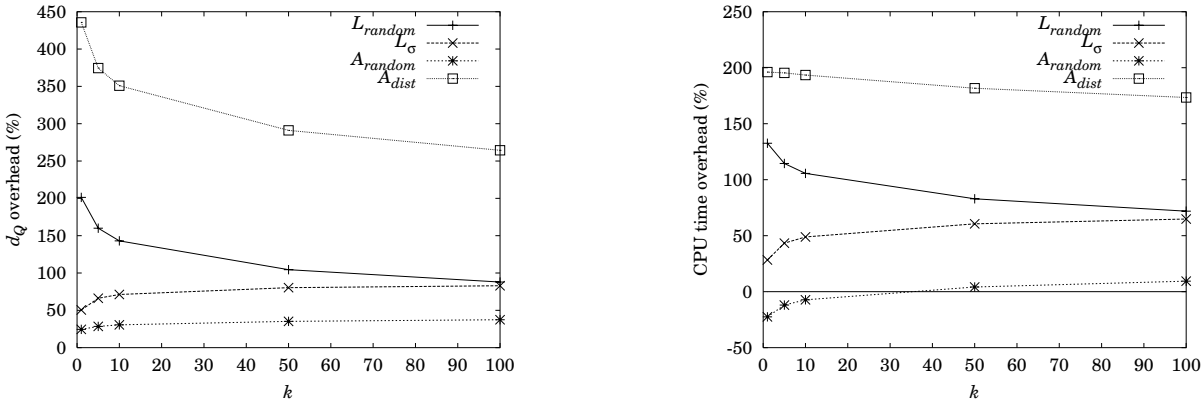
A_{dist}**:** A quadratic function, where each coefficient a_{ij} of the A matrix represents the similarity in the RGB space between color i and color j.

Query distance d_Q	$cost_{d_Q}$ (s)	$E[\mathbf{d_Q}]$	$S_{I \to Q}$	Err_I
L_{random}	4.18×10^{-6}	0.409	22.74	0.295
L_σ	4.18×10^{-6}	1.79	0.433	0.0898
A_{random}	60.4×10^{-6}	2.01	0.337	0.0650
A_{dist}	60.4×10^{-6}	0.619	173.09	0.422

Table 6: Indexing error values and computation times for different metrics with respect to the Euclidean distance for the `Corel` data set ($cost_{L_2} = 4.05 \times 10^{-6}$, $E[\mathbf{L_2}] = 0.573$).

In Figure 19 we plot the query distance and the CPU time overhead ($oh_{\#_{d_Q}}$ and $oh_{time_{CPU}}$, respectively) for a k-NN query, as a function of k.

As expected, the increase in the number of d_Q computations is again directly related to the indexing error (see Figure 19 (a)), with all cases presenting a positive overhead. On the other hand, when turning



(a) (b)

Figure 19: Distance (a) and CPU time (b) overhead for k-NN queries for the `Corel` data set for different querying functions.

to consider the CPU time overhead (see Figure 19 (b)), the higher cost of quadratic functions becomes apparent. In particular, in the case of A_{random} and for $k \leq 30$, the QIC approach is even more convenient than UNF.

6.2.4 Comparison with F&R

Finally, we compare the QIC and the filter & refine approaches. To this end, we use the `AirPhoto` data sets with $d_Q \equiv L_2$ and the L_1, L_3, and L_∞ metrics as index distances. Distance computation costs and indexing errors with respect to d_Q are given in Table 7.

Index distance d_I	$cost_{d_I}$ (s)	Err_I
L_1	0.75×10^{-6}	0.0523
L_2	1.25×10^{-6}	0
L_3	1.5×10^{-6}	0.0348
L_∞	1.0×10^{-6}	0.0576

(a)

Index distance d_I	$cost_{d_I}$ (s)	Err_I
L_1	2.76×10^{-6}	0.1223
L_2	4.8×10^{-6}	0
L_3	5.5×10^{-6}	0.0635
L_∞	4.51×10^{-6}	0.0971

(b)

Table 7: Distance computation times and indexing errors for the `Airphoto8` (a) and the `Airphoto60` (b) data sets.

Figure 20 shows CPU times for solving a range query using different index distances. From the graphs it can be observed that using the QIC approach always leads to lower search CPU costs except when querying the `AirPhoto60` data set with $d_I \equiv L_1$ and query selectivities less than 3%. This is due to the combination of two factors: the low query selectivity, which alleviates the cost of the refinement phase, and the cheapness of L_1 wrt L_2 (see Table 7 (b)), which reduces the cost of distance computations at the leaf level of the tree.

To better analyze the additional cost of the refinement phase, Figure 21 (a) shows the distance distributions of the `AirPhoto60` data set for different index distances plotted as functions of the query selectivity. In order to find the predicted number of query distances to be computed for retrieving a given percentage of objects, one should look at the value of the index distance distribution for that value of the query selectivity, and multiply it by the number of indexed objects (as an example, if the user would select 10% of the data set using the L_1 metric as the index distance, the filter phase would retrieve around 30% of the indexed objects). The (percent) difference between distance distributions values for the index and the query distance represents the false drop percentage (this is also shown in Figure 21 (b), again for the `AirPhoto60` data set). Similar results are also obtained for the `AirPhoto8` data set.

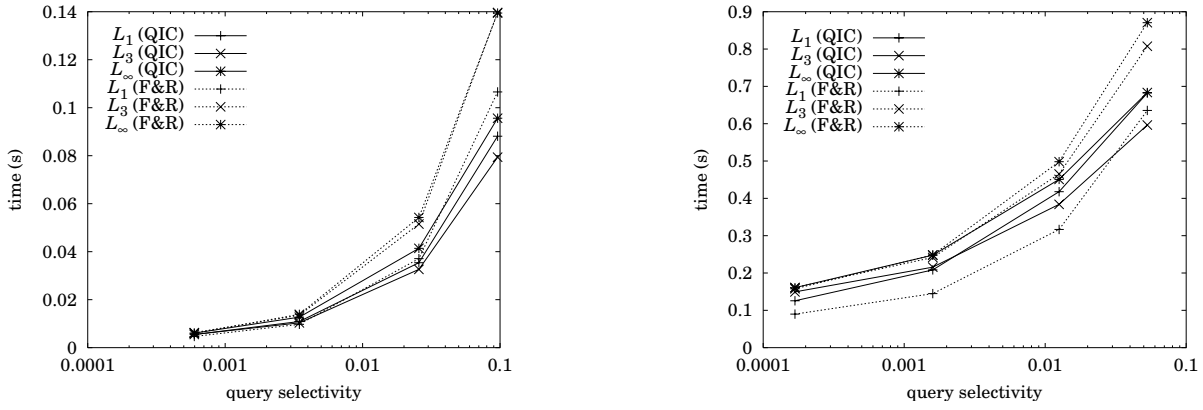


Figure 20: Comparison of CPU search costs for the QIC and the filter & refine strategies when using different index distance functions for the `AirPhoto8` (a) and the `AirPhoto60` (b) data sets.

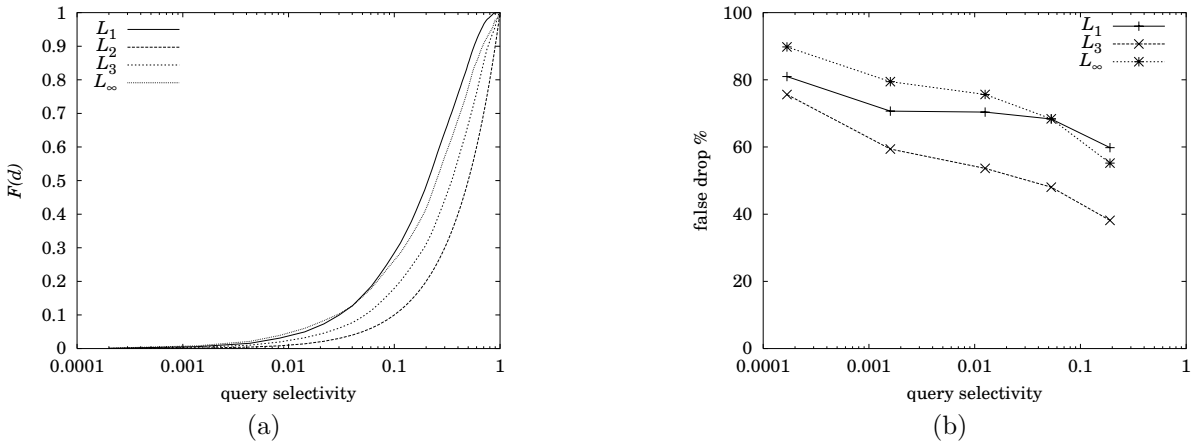


Figure 21: Distance distribution of different index distances (a) and percentage of false drops (b) as a function of the query selectivity for the `AirPhoto60` data set.

In order to predict which strategy is most suitable for the situation at hand, the cost model can be used. Since the index I/O cost is the same for QIC and F&R, and so is the number of computed index distances, it is sufficient to consider differences arising from the number of query distance computations. For QIC this cost evaluates to $\#_{d_Q} \cdot cost_{d_Q}$, whereas for F&R the cost is $\#_{d_Q} \cdot cost_{d_I} + M \cdot F_{\mathbf{d_I}}(S_{I \to Q}\ \epsilon) \cdot cost_{d_Q}$, since the number of objects returned by the index can be estimated as $M \cdot F_{\mathbf{d_I}}(S_{I \to Q}\ \epsilon)$. As a consequence, F&R has a lower cost than QIC as long as

$$\#_{d_Q} \cdot cost_{d_I} + M \cdot F_{\mathbf{d_I}}(S_{I \to Q}\ \epsilon) \cdot cost_{d_Q} \leq \#_{d_Q} \cdot cost_{d_Q} \tag{26}$$

This is the case iff

$$\frac{M \cdot F_{\mathbf{d_I}}(S_{I \to Q}\ \epsilon)}{\#_{d_Q}} = \frac{F_{\mathbf{d_I}}(S_{I \to Q}\ \epsilon)}{sel_{d_Q}} \leq 1 - \frac{cost_{d_I}}{cost_{d_Q}} \tag{27}$$

where $\#_{d_Q}$ can be predicted using Equation 10 (or Equation 12).

As an example, for a search radius $\epsilon = 5$ (corresponding to a query selectivity of 1.25%) and $d_I \equiv L_1$, from Equation 12 we estimate $sel_{d_Q} = 23.94\%$. The scaling factor to be used in Equation 27 is $D^{1/1-1/2} = \sqrt{60}$ (see Corollary 1 in Section 3.2). Since $F_{\mathbf{L_1}}(\sqrt{60} \cdot 5) = 5.315\%$ (Figure 21 (a)), we obtain $\frac{5.315}{23.94} = 0.222 <$

$\left(1 - \frac{2.76}{4.8}\right) = 0.42$ (see Table 7 (b)). For the same search radius, using $d_I \equiv L_\infty$ yields $sel_{d_Q} = 21.56\%$, and $F_{\mathbf{L_1}}(5) = 6.293\%$ (see Figure 21 (a)). Therefore, it is $\frac{6.293}{21.56} = 0.292 > \left(1 - \frac{4.51}{4.8}\right) = 0.05$ (see Table 7 (b)). Thus, when $d_I \equiv L_1$, it is cheaper to use the filter & refine approach, whereas if $d_I \equiv L_\infty$ we should use the QIC solution. This fully agrees with results in Figure 20 (b).

6.3 Summary of Experiments

The results of our experimental analysis can be summarized as follows:

- The cost model can accurately predict performance of QIC-M-tree, delivering estimates which are robust with respect to parameter changes.
- The effectiveness of using a comparison distance d_C can also be reliably estimated, and consequently the maximum query radius that limits the usefulness of d_C. These results are not limited to the QIC-M-tree, but are also valid for other MAMs.
- Further, when several d_Cs are available, the best one to adopt for the case at hand can be analytically chosen.
- The indexing error defined by Equation 25 is very effective in accounting for the performance degradation when the query distance is different from the index distance.
- The QIC approach can even lead to better performance than the so-called UNF solution, i.e. when $d_I \equiv d_Q$. This is possible when $cost_{d_I} < cost_{d_Q}$.
- Whenever $d_I \neq d_Q$, both QIC and its F&R variant are viable alternatives. Using the cost model, we can predict which of them is the most suitable to use in a given context.

7 Extending to Other Metric Access Methods

The QIC approach that we have extensively described with reference to the M-tree can also be profitably adapted to work with other MAMs. This is trivially true for the Slim-tree [TTSF00], since this MAM uses the same organizing principles of the M-tree. Below we provide basic intuition on how both vp-tree [Yia93, Chi94] and GNAT [Bri95] can benefit from our findings, whereas we leave out the mvp-tree [BÖ97], since its analysis would closely follow that of the vp-tree. For simplicity, we consider basic (i.e. binary) versions of vp-tree and GNAT, and only refer to the case of range queries. For clarity purpose the notation we use is here adapted to be consistent with the rest of this paper. As a representative of non-tree-structured methods, we also consider the LAESA algorithm [MOV94].

7.1 vp-tree

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 $[O^{[v]}, \mu, ptr(N_l), ptr(N_r)]$, where $O^{[v]}$ is the vantage point (i.e. an object of the data set), μ is (an estimate of) the median of the distances between $O^{[v]}$ and all the objects reachable from the node, and $ptr(N_l)$ and $ptr(N_r)$ are pointers to the left and right child node, respectively. The left child node, N_l, indexes the objects whose distance from $O^{[v]}$ is less than or equal to μ, thus $Reg(N_l) = \{O \in \mathcal{U} | d_I(O^{[v]}, O) \leq \mu\}$. The right child node, N_r, indexes the objects whose distance from $O^{[v]}$ is greater than μ, that is, $Reg(N_r) = \{O \in \mathcal{U} | d_I(O^{[v]}, O) > \mu\}$. The same partitioning principle is then recursively applied to the N_l and N_r child nodes.

Given a range query $\texttt{range}(\mathcal{O}, Q, \epsilon, d_Q)$, consider a vp-tree built using the distance d_I, with $d_I \preceq_{S_{I \to Q}} d_Q$, and a node N. From Theorem 2 it is immediate to conclude that we can skip reading the left child node N_l if $d_I(Q, O^{[v]}) > S_{I \to Q}\, \epsilon + \mu$ holds. As for the right child node N_r, arguments are almost similar. In particular, we can use the pruning condition $d_I(Q, O^{[v]}) \leq \mu - S_{I \to Q}\, \epsilon$, since:

$$d_I(Q, O^{[v]}) \leq \mu - S_{I \to Q}\, \epsilon \implies d_Q(Q, O) > \epsilon \;\; \forall O \in Reg(N_r)$$

This is derived as follows, where the "trick" is to use triangle inequality in the opposite way as usually done (since we want to prune a node which is "far" from the vantage point):

$$\begin{aligned}
d_Q(Q,O) &\geq d_I(Q,O)/S_{I\to Q} && \text{(by the hyp. } d_I \preceq S_{I\to Q}\ d_Q\text{)}\\
&\geq d_I(O^{[v]},O)/S_{I\to Q} - d_I(Q,O^{[v]})/S_{I\to Q} && \text{(triangle inequality)}\\
&> \mu/S_{I\to Q} - d_I(Q,O^{[v]})/S_{I\to Q} && \text{(since } O \in Reg(N_r)\text{)}\\
&\geq \mu/S_{I\to Q} - (\mu - S_{I\to Q}\ \epsilon)/S_{I\to Q} && \text{(by hypothesis)}\\
&= \epsilon
\end{aligned}$$

Finally, if also a comparison distance d_C is used, appropriately scaling the search threshold still guarantees correctness of the search algorithm.

7.2 GNAT

We now turn to consider GNAT. In the simplest version, each node N of this structure has the format:

$$[O^{[1]}, ptr(N_1), O^{[2]}, ptr(N_2),\\ d_I^{min}(O^{[1]}, Reg(N_2)), d_I^{max}(O^{[1]}, Reg(N_2)), d_I^{min}(O^{[2]}, Reg(N_1)), d_I^{max}(O^{[2]}, Reg(N_1))]$$

where $O^{[1]}$ and $O^{[2]}$ are two *split points*, and $ptr(N_1)$ and $ptr(N_2)$ are pointers to the corresponding child nodes. The d_I^{min} and d_I^{max} values are respectively the minimum and the maximum distance between $O^{[1]}$ ($O^{[2]}$) and the indexed objects assigned to the child node N_2 (N_1, respectively). The partitioning of objects is based on a *generalized hyperplane* rule: if $d_I(O^{[1]}, O_i) \leq d_I(O^{[2]}, O_i)$ then assign O_i to node N_1, else assign it to node N_2.

The QIC Theorem can be easily adapted to work with GNAT. The pruning step of GNAT search algorithm uses a split point, say $O^{[2]}$, to verify if the node associated to another split point, say $O^{[1]}$, can be pruned. After computing $d_I(Q, O^{[2]})$ the pruning test is [Bri95]:

$$[d_I(Q,O^{[2]}) - \epsilon, d_I(Q,O^{[2]}) + \epsilon] \cap [d_I^{min}(O^{[2]}, Reg(N_1)), d_I^{max}(O^{[2]}, Reg(N_1))] = \emptyset$$

Note that the role of $O^{[2]}$ in the test is similar to that of a vantage point, since all distances refer to it. For our purposes, above test can be more conveniently rewritten as:

$$d_I(Q,O^{[2]}) + \epsilon < d_I^{min}(O^{[2]}, Reg(N_1)) \qquad \text{or} \qquad d_I(Q,O^{[2]}) - \epsilon > d_I^{max}(O^{[2]}, Reg(N_1))$$

We now show that if above tests are replaced by the following ones:

$$d_I(Q,O^{[2]}) + S_{I\to Q}\ \epsilon < d_I^{min}(O^{[2]}, Reg(N_1)) \tag{28}$$

$$d_I(Q,O^{[2]}) - S_{I\to Q}\ \epsilon > d_I^{max}(O^{[2]}, Reg(N_1)) \tag{29}$$

then pruning is still correct in the case $d_I \preceq S_{I\to Q}\ d_Q$. Consider first inequality (28). We have, for each point $O \in Reg(N_1)$:

$$\begin{aligned}
d_Q(Q,O) &\geq d_I(Q,O)/S_{I\to Q} && \text{(by the hyp. } d_I \preceq S_{I\to Q}\ d_Q\text{)}\\
&\geq d_I(O^{[2]},O)/S_{I\to Q} - d_I(Q,O^{[2]})/S_{I\to Q} && \text{(triangle inequality)}\\
&> d_I(O^{[2]},O)/S_{I\to Q} + (S_{I\to Q}\ \epsilon - d_I^{min}(O^{[2]}, Reg(N_1)))/S_{I\to Q} && \text{(by hypothesis)}\\
&\geq d_I^{min}(O^{[2]}, Reg(N_1))/S_{I\to Q} - d_I^{min}(O^{[2]}, Reg(N_1))/S_{I\to Q} + \epsilon && \text{(by def. of } d_I^{min}(O^{[2]}, Reg(N_1))\text{)}\\
&= \epsilon
\end{aligned}$$

Similarly, when inequality (29) holds:

$$\begin{aligned} d_Q(Q,O) &\geq d_I(Q,O)/S_{I\to Q} && \text{(by the hyp. } d_I \preceq S_{I\to Q}\, d_Q) \\ &\geq d_I(Q,O^{[2]})/S_{I\to Q} - d_I(O^{[2]},O)/S_{I\to Q} && \text{(triangle inequality)} \\ &> (d_I^{max}(O^{[2]},Reg(N_1)) + S_{I\to Q}\,\epsilon)/S_{I\to Q} - d_I(O,O^{[2]})/S_{I\to Q} && \text{(by hypothesis)} \\ &\geq d_I^{max}(O^{[2]},Reg(N_1))/S_{I\to Q} + \epsilon - d_I^{max}(O^{[2]},Reg(N_1))/S_{I\to Q} && \text{(by def. of } d_I^{max}(O^{[2]},Reg(N_1))) \\ &= \epsilon \end{aligned}$$

Again, introducing a comparison distance d_C does not lead to major modifications.

7.3 LAESA

The Linear Approximating Eliminating Search Algorithm [MOV94] works by pre-computing the distances between the indexed objects and (properly chosen) *pivot* objects $\mathcal{P} = \{P_1, \ldots, P_s\} \subseteq O$. Space and construction time for LAESA are both $O(m \cdot s)$. Consider a range query $\texttt{range}(\mathcal{O}, Q, \epsilon, d_I)$. At query time, first the distances $d_I(Q, P_i)$ are computed, then (without computing any further distance) all objects O for which $|d_I(Q,P_i) - d_I(P_i,O)| > \epsilon$ holds for at least one pivot are discarded, whereas remaining objects are directly compared against Q.

When $d_I \preceq S_{I\to Q}\, d_Q$, then the correct pruning condition is:

$$|d_I(Q,p_i) - d_I(p_i,O)| > S_{I\to Q}\,\epsilon$$

since:

$$\begin{aligned} d_Q(Q,O) &\geq d_I(Q,O)/S_{I\to Q} && \text{(by the hyp. } d_I \preceq S_{I\to Q}\, d_Q) \\ &\geq |d_I(Q,p_i) - d_I(p_i,O)|\,/S_{I\to Q} && \text{(triangle inequality)} \\ &> S_{I\to Q}\,\epsilon/S_{I\to Q} && \text{(by hypothesis)} \\ &= \epsilon \end{aligned}$$

Even with LAESA, the presence of d_C can be easily accommodated.

8 Other Query Types

In this Section we briefly describe how other query types supported by MAMs can be managed by the QIC approach.

8.1 Sorted Access Queries

Given a data set $\mathcal{O}$ and a query Q, a *sorted access query* requests to output, one by one, the objects of $\mathcal{O}$ in order of increasing distance from Q. The user is also given the possibility to stop the output whenever he/she wants, so that it is not necessary to retrieve all the M indexed objects.

For solving sorted access queries, we consider the algorithm presented in [HS99], which is already part of the M-tree repertoire. Essentially, the algorithm is similar to `k-NNSearch` with the major difference that no pruning of nodes (and objects) can take place and that the `RL` result list has a variable size, since it contains all the objects that have been accessed but have not been already delivered as results.

In light of this, the algorithm can be immediately extended by using d_I on internal nodes and d_Q on objects (stored in leaf nodes). Further, the scaling factor $S_{I\to Q}$ should be used when comparing the distance from Q of the top element of `RL` with the distance from Q to the region of the most promising node (i.e. the top element of the priority queue `PQ`).

As a final remark, we observe that, since no nodes or objects can be pruned, *the use of a comparison distance is meaningless*. Finally, it can be argued that the performance of the algorithm for retrieving k objects can be still predicted using the cost model in Section 5.1, with the advise that the optimizations of lines 8. and 19. cannot be used.

8.2 Complex Similarity Queries

Complex similarity queries [Fag96, CPZ98b, BMSW01] are another query type supported by MAMs that can be extended to deal with user-defined and approximate distance functions. For the purpose of our work, complex queries can be modelled as conventional range or k-NN queries where a *combining function* [Fag96], CF, combines the distance values[12] of an (indexed) object O with respect to $p \geq 1$ query objects $\mathcal{Q} = \{Q_1, \ldots, Q_p\}$.[13] Then, the "distance" d_Q between an object O and $\mathcal{Q}$ is obtained as $CF(d_Q(Q_1, O), \ldots, d_Q(Q_p, O))$.[14]

In [CPZ98b] it is proved that when $d_Q \equiv d_I$, under the assumption of monotonicity of the combining function, *any* index tree based on a recursive and conservative decomposition of the space can process complex queries *as a whole*, i.e. without decomposing them into p simple queries. The basic rationale of the solution in [CPZ98b] is that, in order to be able to prune a node N of the tree, one has to compute a lower bound of CF and check that such bound is higher than the query threshold ϵ. Because of the monotonicity of CF, if one computes $d_I^{min}(Q_i, Reg(N))$ for each query object Q_i, and then plugs such values as arguments of CF, then it is guaranteed that

$$CF\left(d_I^{min}(Q_1, Reg(N)), \ldots, d_I^{min}(Q_p, Reg(N))\right) > \epsilon \implies d_I(\mathcal{Q}, O) > \epsilon \qquad \forall O \in Reg(N)$$

The following Corollary generalizes the above to the case $d_Q \neq d_I$, also considering the presence of a comparison distance d_C.

Corollary 3
Let $\mathcal{Q} = \{Q_1, \ldots, Q_p\}$, be a set of query objects, CF a monotonic non-decreasing combining function, $\mathcal{T}$ a tree-structured MAM built on the data set $\mathcal{O}$ using d_I, d_Q a query distance, and d_C a comparison distance, with $d_I \preceq S_{I \to Q}\, d_Q$, $d_C \preceq S_{C \to I}\, d_I$, and $d_C \preceq S_{C \to Q}\, d_Q$. Then a node N of $\mathcal{T}$ can be pruned if

$$CF(d_I^{min}(Q_1, Reg(N))/S_{I \to Q}, \ldots, d_I^{min}(Q_p, Reg(N))/S_{I \to Q}) > \epsilon' \tag{30}$$

where ϵ' is

- *the user supplied query radius, if the query is a range query;*
- *the k-th lower distance d_Q encountered so far between the complex predicate and indexed objects, if the query is a k-NN query. If less than k objects have been evaluated, then $\epsilon' = d_Q^+$.*

Moreover, let $d_C^*(Q_i, Reg(N))$ be a lower bound of $S_{C \to I}\, d_I^{min}(Q_i, Reg(N))$. Then, node N can be pruned if $CF(d_C^*(Q_1, Reg(N))/(S_{C \to I}\ S_{I \to Q}), \ldots, d_C^*(Q_p, Reg(N))/(S_{C \to I}\ S_{I \to Q})) > \epsilon'$, and object O can be discarded from the result if $CF(d_C(Q_1, O)/S_{C \to Q}, \ldots, d_C(Q_p, O)/S_{C \to Q}) > \epsilon'$. □

Proof: Given in Appendix B. □

It has to be observed that above Theorem relies on the possibility of (efficiently) evaluating $d_C^*(Q_i, Reg(N))$. For the specific case of the QIC-M-tree, let

$$d_C^*(Q_i, Reg(N)) = \max\left\{d_C(Q_i, O^{[N]}) - S_{C \to I}\, r^{[N]}, 0\right\}$$

Then it is immediate to see that

$$d_C^*(Q_i, Reg(N)) \leq \max\left\{S_{C \to I}\, d_I(Q_i, O^{[N]}) - S_{C \to I}\, r^{[N]}, 0\right\} = S_{C \to I}\, d_I^{min}(Q_i, Reg(N))$$

[12] In [Fag96, CPZ98b] the combining function was applied to produce a *similarity score*; we adapted this concept to better suit our scenario, being understood that high similarity values correspond to low distances and vice versa.

[13] Note that complex queries are different from *multiple queries* [BEKS00]; in a complex query, multiple predicates are combined into a single query, whereas multiple queries are defined as sets of (simple) queries issued simultaneously, each with its own result.

[14] Nothing would prevent to use a different query distance d_{Q_i} for each query object Q_i. We do not consider this case in the following.

9 Related Work

At the best of our knowledge, this is the first work that investigates the use of metric access methods to answer similarity queries based on user-defined distance functions. Indeed, the complain that MAMs are bound to use a given distance function was raised several times (e.g. see [SK98, AKS98, KSF+98, CM99, SYKU01]). For the same reason, the issue of supporting adaptable similarity criteria for searching in generic metric spaces has not been investigated before. Thus, being unable to compare our approach with direct competitors, in the following we restrict ourselves to briefly discuss other approaches and techniques that somewhat have been influential to our work.

The paper by Seidl and Kriegel [SK97] was the first to consider the possibility of using Spatial Access Methods to efficiently support user-adaptable similarity queries in vector spaces. Their basic algorithm for answering elliptic queries, which is based on computing intersections of ellipsoids and boxes, is however only suitable in low D-dimensional spaces, because of its $O(D^2)$ complexity. To overcome such limitation, the authors propose to apply dimensionality reduction techniques and to exploit the lower-bounding lemma (first introduced in [AFS93] to deal with similarity queries over time sequences) to implement a filter & refine query processing algorithm.

Several other approaches have been proposed to increase performance of SAMs for user-adaptable similarity queries in high-dimensional spaces. Basically, all of them either rely on a transformation of the original space to a low-dimensional space equipped with the Euclidean distance (this is the case, for example, of the Spatial Transformation Technique in [SYKU01]), or approximate the query region with a simpler figure, e.g. a hyper-sphere or a hyper-box [ABKS98]. In both cases, the so-obtained problem is considerably simpler to be solved with respect to the original one and the lower-bounding property is still used to guarantee the correctness of search algorithms.

When turning to consider similarity queries in generic metric spaces, three major trends for speeding-up query evaluation can be observed. The first one, well represented by the so-called GEMINI approach [ZCF+97, Chapter 12], is to extract from each object a *feature vector* and to replace the original object distance with a distance in the so-resulting feature (vector) space. Again, this allows SAMs to be used and, provided the feature distance lower-bounds the object distance, the filter & refine processing algorithm to be applied. The problem of finding a distance in a feature space that lower-bounds the object distance has however to be solved using domain-specific considerations (see, for example, [FEF+94] for the case of color images). Alternatively, and without the need of a domain knowledge, one could try to map objects into points of a (low-dimensional) vector space so that relative object distances are preserved as much as possible in the resulting target space. FastMap [FL95] is a well-known representative of this approach. In [HS00] it is however demonstrated that mappings achieved through FastMap do not satisfy the lower-bounding property unless the original space is (isometric to) the Euclidean space: Correctness of search algorithms has therefore to be given up. Other mappings are proposed in [HS00] that satisfy the lower-bounding property, but they require high computation costs, and therefore cannot be used to improve search performance. Finally, several techniques have been proposed to reduce the number of distance computations at query time without resorting to a vector space mapping [CNBYM, FTTF01]. Such techniques use optimizations that are very similar to those used for the M-tree, in that pre-computed distances between indexed objects and a set of suitable objects, called *pivots*,[15] are used together with the triangle inequality to exclude objects from the result.

10 Final Discussion

In recent times, metric spaces have become a popular paradigm for similarity retrieval. Several techniques have been proposed to efficiently search in generic metric spaces where the (dis)similarity criterion is known in advance. However, none of such techniques is able to support user-defined distance functions which are not a priori available. Moreover, in complex domains where computing a single distance is a very expensive task, the search can become CPU-bound, thus requiring alternative solutions to improve query processing

[15] These are called *foci* in [FTTF01].

performance. We have addressed the problems of supporting user-defined distance functions with a metric access method (MAM), and of using approximate distances to reduce the CPU cost of similarity search. In particular, we have detailed how an appropriately extended M-tree built using an index distance d_I can:

- Use any query distance d_Q for which d_I is a (scaled) lower bound.
- Use any (cheap) comparison distance d_C which lower-bounds d_I with the purpose of quickly pruning objects and tree nodes.

We have also shown how our so-called QIC (Query, Index, and Comparison) approach can be extended to other MAMs (namely, the vp-tree, the GNAT, and LAESA). Our claim is that *any* MAM can be QICly extended.

The major contributions of this paper can be summarized as follows:

1. Definition of QIC algorithms for the M-tree to solve range and k-NN queries.
2. Introduction of a cost model to analytically estimate search performance of QIC-M-tree. The accuracy of the model has been evaluated through extensive experimentations.
3. Application of the model to predict the usefulness of the comparison distance and to choose the best comparison distance, in case several of them are available.
4. Application of the model to choose the best query processing strategy between QIC and its filter & refine variant.
5. We have shown that differences in distance distribution between d_Q and d_I impact search costs, and have introduced the indexing error, which is a measure able to characterize this decrease in performance.
6. Extension of the QIC approach to other MAMs.
7. Extension of QIC-M-tree to other query types, namely sorted access queries and complex queries.

Our work generates a number of interesting follow-ups. First, it is now possible to compare SAMs and MAMs in a larger variety of scenarios. This gives more freedom to a designer in choosing the best access method for the problem at hand. In this light, it has to be remarked that recent studies have demonstrated how even in vector spaces MAMs can outperform SAMs [CPZ97, BDBP00]. Second, as an interesting design problem, it could be worth investigating the case where multiple QIC-extended MAMs, each with its own index distance, are available at the same time, and the problem becomes that of choosing, for a given query, the most suitable one to use (see also [SYKU01]). For this task, the indexing error defined in Section 6.2.1 could be profitably used.

The basic lesson we have learned is that MAMs are not tightly bound to use a single distance function. From this, one could argue whether other basic ways of extending MAMs, besides the one considered in this paper, exist. At present, we envision two interesting research directions:

- For the sake of generality, in this paper we have made no assumptions on the metric nature of d_Q (as well as of d_C). If one assumes that d_Q is a metric and that there exists $S_{Q\to I}$ such that $d_Q \preceq S_{Q\to I}\ d_I$, then new opportunities for pruning the search space are available. Indeed, with reference to the QIC-M-tree range search algorithm, it is easy to demonstrate that $d_Q(Q, O^{[N]}) > \epsilon + S_{Q\to I}\ r^{[N]}$ is now a (new) sufficient condition for pruning node N. In other terms, rather than just "changing the shape" of the query region, one could also change the shape of index regions. It has to be observed that this pruning criterion is somewhat orthogonal to the one we have considered in this paper (i.e. $d_I(Q, O^{[N]}) > S_{I\to Q}\ \epsilon + r^{[N]}$), so that performance improvements are expected. As a simple example, Figure 22 illustrates the case $d_I \equiv L_2$ and $d_Q \equiv L_1$. Same arguments apply to the comparison distance.

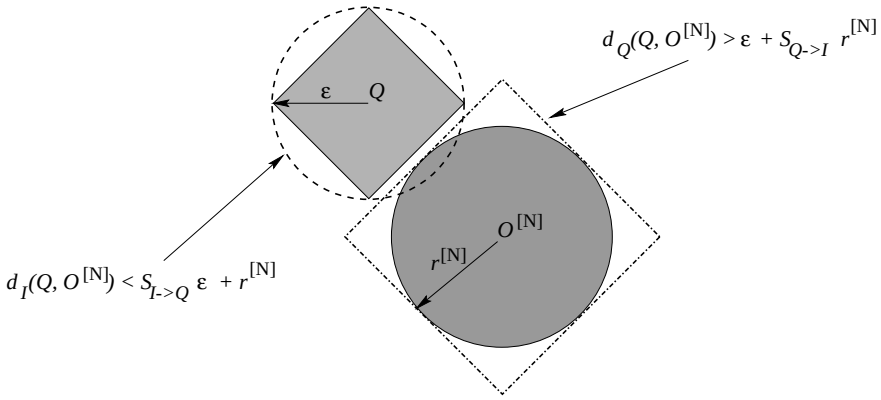


Figure 22: Searching with $d_Q \neq d_I$: Either the query region or the node region are inflated. In the former case (QIC approach) the index node has to be accessed, whereas in the latter it can be pruned.

- Another challenging research issue is to revise the design principles of a MAM so that from the very beginning it is suited to support a class of distance functions. As a concrete example, consider the class of L_p norms and the M-tree basic structure: If each entry of an M-tree node not only stores the covering radius computed according to a *primary* d_I (say, L_2) but also the covering radii for other L_p norms (e.g. L_1, L_∞), then this additional information could be directly exploited when answering queries using, say, L_1. In some sense, this is related to the above-discussed extension where explicit covering radii are maintained rather than using only the bounds provided by the $S_{Q \to I}$ scaling factor(s).

References

[ABKS98] Mihael Ankerst, Bernhard Braunmüller, Hans-Peter Kriegel, and Thomas Seidl. Improving adaptable similarity query processing by using approximations. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 206–217, New York City, NY, August 1998.

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organizations and Algorithms, (FODO'93)*, pages 69–84, Chicago, IL, October 1993. Springer-Verlag, LNCS, Vol. 730.

[AKS98] Mihael Ankerst, Hans-Peter Kriegel, and Thomas Seidl. A multistep approach for shape similarity search in image databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):996–1004, 1998.

[BBKK97] Stefan Berchtold, Christian Böhm, Daniel A. Keim, and Hans-Peter 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.

[BCW01] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. FeedbackBypass: A new approach to interactive similarity query processing. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 201–210, Rome, Italy, September 2001.

[BDBP00] Stefano Berretti, Alberto Del Bimbo, and Pietro Pala. Retrieval by shape similarity with perceptual distance and effective indexing. *IEEE Transactions on Multimedia*, 2(4):225–239, December 2000.

[BEKS00] Bernhard Braunmüller, Martin Ester, Hans-Peter Kriegel, and Jörg Sander. Efficiently supporting multiple similarity queries for mining in metric databases. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 256–267, San Diego, CA, March 2000.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 28–39, Mumbai (Bombay), India, September 1996.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard 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.

[BMSW01] Klemens Böhm, Michael Mlivoncic, Hans-Jörg Schek, and Roger Weber. Fast evaluation techniques for complex similarity queries. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 211–220, Rome, Italy, September 2001.

[BÖ97] Tolga Bozkaya and Meral 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.

[BÖ99] Tolga Bozkaya and Meral Özsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, September 1999.

[Bri95] Sergey 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.

[Chi94] Tzi-cker 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.

[CM99] Kaushik Chakrabarti and Sharad Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, pages 440–447, Sydney, Australia, March 1999.

[CNBYM] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Proximity searching in metric spaces. To appear in ACM Computing Surveys.

[CP00] Paolo Ciaccia and Marco Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 244–255, San Diego, CA, March 2000.

[CP01] Paolo Ciaccia and Marco Patella. Approximate similarity queries: A survey. Technical Report CSITE-08-01, CSITE–CNR, 2001. Available at URL `http://www-db.deis.unibo.it/MMDBGroup/TRs.html`.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel 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.

[CPZ98a] Paolo Ciaccia, Marco Patella, and Pavel Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 59–68, Seattle, WA, June 1998.

[CPZ98b] Paolo Ciaccia, Marco Patella, and Pavel 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.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley & Sons, New York, 1991.

[Fag96] Ronald Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'96)*, pages 216–226, Montreal, Canada, June 1996.

[FEF+94] Christos Faloutsos, Will Equitz, Myron Flickner, Wayne Niblack, Dragutin Petkovic, and Ron Barber. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, July 1994.

[FL95] Christos Faloutsos and King-Ip Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, CA, June 1995.

[FTTF01] Roberto Figueira Santos Filho, Agma J. M. Traina, Caetano Traina Jr., and Christos Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *Proceedings of the 17th International Conference on Data Engineering (ICDE 2001)*, pages 623–630, Heidelberg, Germany, April 2001.

[GG98] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[Gut] Project Gutenberg official home site. `http://www.gutenberg.net/`.

[Gut84] Antonin 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.

[HS99] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, June 1999.

[HS00] Gísli R. Hjaltason and Hanan Samet. Contractive embedding methods for similarity searching in metric spaces. Technical Report CS-TR-4102, Computer Science Department, University of Maryland, College Park, MD, February 2000.

[HSE+95] Jim Hafner, Harpreet Sawhney, Will Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):729–736, July 1995.

[KDD] The UCI Knowledge Discovery in Databases archive. University of California, Irvine, Department of Information and Computer Science. `http://kdd.ics.uci.edu`.

[KS97] Norio Katayama and Shin'ichi Satoh. 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*, pages 369–380, New York, NY, May 1997.

[KS01] Tamer Kahveci and Ambuj K. Singh. An efficient index structure for string databases. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 351–360, Rome, Italy, September 2001.

[KSF+98] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot Siegel, and Zenon Protopapas. Fast and effective retrieval of medical tumor shapes. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):889–904, 1998.

[Man] B.S. Manjunath. The airphoto data set. `http://vivaldi.ece.ucsb.edu/users/wei/gaborfeatures/airphotoFeatures/`.

[MM96] B.S. Manjunath and Wei-Ying Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, August 1996.

[MOV94] María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, January 1994.

[ORC+98] Michael Ortega, Yong Rui, Kaushik Chakrabarti, Kriengkrai Porkaew, Sharad Mehrotra, and Thomas S. Huang. Supporting ranked boolean similarity queries in MARS. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):905–925, January 1998.

[PC99] Kriengkrai Porkaew and Kaushik Chakrabarti. Query refinement for multimedia similarity retrieval in MARS. In *Proceedings of the 7th ACM International Conference on Multimedia*, volume 1, pages 235–238, Orlando, FL, November 1999.

[RKV95] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, CA, May 1995.

[RTG98] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the 6th International Conference on Computer Vision ICCV'98*, pages 59–66, Mumbai, India, January 1998.

[SK97] Thomas Seidl and Hans-Peter Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 506–515, Athens, Greece, August 1997.

[SK98] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 154–165, Seattle, WA, June 1998.

[SYKU01] Yasushi Sakurai, Masatoshi Yoshikawa, Ryoji Kataoka, and Shunsuke Uemura. Similarity search for adaptive ellipsoid queries using spatial transformation. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 231–240, Rome, Italy, September 2001.

[TTF00] Caetano Traina Jr., Agma J. M. Traina, and Christos Faloutsos. Distance exponent: A new concept for selectivity estimation in metric trees. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, page 195, San Diego, CA, March 2000.

[TTSF00] Caetano Traina Jr., Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT 2000)*, pages 51–65, Konstanz, Germany, March 2000.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974.

[Yia93] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, Austin, TX, January 1993.

[ZCF+97] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V.S. Subrahmanian, and Roberto Zicari, editors. *Advanced Database Systems*. Morgan Kaufmann, San Francisco, 1997.

A Lower-bounding Metric for Quadratic Form Distance Functions

In this Appendix we prove that the Euclidean distance L_2 is a lower-bounding distance function for all quadratic form distance functions. Theorem 4 is an alternative formulation of a result presented in [HSE+95] to better suit our scenario.

Theorem 4 ([HSE+95])
Let $\mathcal{A}$ be the set of all $D \times D$ symmetric matrices A such that $d_{qf}[A]$ is a quadratic form distance function over the D-dimensional vector space. Then $L_2 \preceq d_{qf}[\mathcal{A}]$ and the optimal scaling for $d_{qf}[A]$ is given by $S_{L_2 \to qf}[A] = 1/\sqrt{\min_j\{\lambda_j\}}$, where $\min_j\{\lambda_j\}$ is the minimum eigenvalue of A. □

Proof: Since A is symmetric, it is $A = U\Lambda U^T$, where Λ is a diagonal matrix whose elements $\lambda_{jj} \equiv \lambda_j$ are the eigenvalues of A, and U is an orthonormal matrix, i.e. $UU^T = U^TU = I$. Then we have:

$$d_{qf}[A](O_1, O_2) = \sqrt{(O_1 - O_2)^T A (O_1 - O_2)} = \sqrt{(O_1 - O_2)^T U \Lambda U^T (O_1 - O_2)} =$$
$$= \sqrt{(U^T(O_1 - O_2))^T \Lambda (U^T(O_1 - O_2))} = \sqrt{(O'_1 - O'_2)^T \Lambda (O'_1 - O'_2)}$$

where $O'_1 = U^TO_1$ and $O'_2 = U^TO_2$. Thus it is

$$d_{qf}[A](O_1, O_2) = \sqrt{\sum_{j=1}^{D} \lambda_j \cdot (O'_{1j} - O'_{2j})^2} \geq \sqrt{\min_j\{\lambda_j\} \sum_{j=1}^{D} (O'_{1j} - O'_{2j})^2} = \sqrt{\min_j\{\lambda_j\}} L_2(O'_1, O'_2)$$

However, since U is orthonormal, it is

$$L_2(O'_1, O'_2) = \sqrt{(U^T(O_1 - O_2))^T (U^T(O_1 - O_2))} = \sqrt{(O_1 - O_2)^T U U^T (O_1 - O_2)} =$$
$$= \sqrt{(O_1 - O_2)^T (O_1 - O_2)} = L_2(O_1, O_2)$$

from which it is derived that $d_{qf}[A](O_1, O_2)/\sqrt{\min_j\{\lambda_j\}} \geq L_2(O_1, O_2)$. □

B Proofs

Proof of Theorem 1. For some sequence $S = S_1S_2 \ldots S_m$ of elementary edit operations, $S_j = (A_j \to B_j)$, it is $d_{edit}[\gamma](X, Y) = \sum_{j=1}^{m} \gamma(S_j) = \sum_{j=1}^{m} \gamma(A_j \to B_j)$. Then we have:

$$d_{edit}[\gamma](X, Y) = \sum_{j=1}^{m} \gamma(A_j \to B_j) = \sum_{j=1}^{m} \gamma_I(A_j \to B_j) \frac{\gamma(A_j \to B_j)}{\gamma_I(A_j \to B_j)} \geq$$
$$\geq \sum_{j=1}^{m} \gamma_I(A_j \to B_j) \min_{A,B,A \neq B} \left\{ \frac{\gamma(A \to B)}{\gamma_I(A \to B)} \right\} = \min_{A,B,A \neq B} \left\{ \frac{\gamma(A \to B)}{\gamma_I(A \to B)} \right\} \sum_{j=1}^{m} \gamma_I(A_j \to B_j) \geq$$
$$\geq \min_{A,B,A \neq B} \left\{ \frac{\gamma(A \to B)}{\gamma_I(A \to B)} \right\} d_{edit}[\gamma_I](X, Y)$$

where in the last step the inequality (rather than equality) is due to the fact that with γ_I there is no guarantee that S is still a minimum cost transformation from X to Y. Note that since the minimum of cost ratios is taken over all (non-null) values of the weight functions, the result does not depend on the specific X and Y input strings. □

Proof of Theorem 2. The proof amounts to show that the test at line 6. does not lead to discard any qualifying object from the result, and that the tests at lines 13. and 15. do not prune form the search any leaf that contains at least one qualifying object. For the test at line 13. it has to be proved that:

$$d_C(Q, O^{[N]}) > S_{C\to I}(S_{I\to Q}\ \epsilon + r^{[N]}) \implies d_Q(Q, O) > \epsilon \quad \forall O \in Reg(N)$$

which can be seen as a generalization of Equation 4. The derivation is as follows:

$$\begin{aligned}
d_Q(Q,O) &\geq d_I(Q,O)/S_{I\to Q} && \text{(by hyp. } d_I \preceq S_{I\to Q}\ d_Q\text{)}\\
&\geq d_I(Q,O^{[N]})/S_{I\to Q} - d_I(O^{[N]},O)/S_{I\to Q} && \text{(triangle inequality)}\\
&\geq d_I(Q,O^{[N]})/S_{I\to Q} - r^{[N]}/S_{I\to Q} && \text{(def. of covering radius)}\\
&\geq d_C(Q,O^{[N]})/(S_{C\to I}\ S_{I\to Q}) - r^{[N]}/S_{I\to Q} && \text{(by hyp. } d_C \preceq S_{C\to I}\ d_I\text{)}\\
&> S_{C\to I}(S_{I\to Q}\ \epsilon + r^{[N]})/(S_{C\to I}\ S_{I\to Q}) - r^{[N]}/S_{I\to Q} && \text{(by hypothesis)}\\
&= (S_{C\to I}\ S_{I\to Q}\ \epsilon)/(S_{C\to I}\ S_{I\to Q}) + r^{[N]}/S_{I\to Q} - r^{[N]}/S_{I\to Q} = \epsilon
\end{aligned}$$

Similarly for the test at line 15. it is

$$\begin{aligned}
d_Q(Q,O) &\geq d_I(Q,O^{[N]})/S_{I\to Q} - r^{[N]}/S_{I\to Q} && \text{(from the previous proof)}\\
&> (S_{I\to Q}\ \epsilon + r^{[N]})/S_{I\to Q} - r^{[N]}/S_{I\to Q} && \text{(by hypothesis)}\\
&= \epsilon
\end{aligned}$$

Finally, the test at line 6. follows immediately

$$\begin{aligned}
d_Q(Q,O) &\geq d_C(Q,O)/S_{C\to Q} && \text{(by hyp. } d_C \preceq S_{C\to Q}\ d_Q\text{)}\\
&> S_{C\to Q}\ \epsilon/S_{C\to Q} && \text{(by hypothesis)}\\
&= \epsilon
\end{aligned}$$

□

Proof of Lemma 1. To compute the distance distribution of the random variable $\mathbf{z} = |d_I(\mathbf{Q}, \mathbf{O^{[N]}}) - d_I(\mathbf{O^{[N]}}, \mathbf{O^{[N_c]}})|$, we can consider it as a function of the two RVs $\mathbf{x} = d_I(Q, \mathbf{O^{[N]}})$ and $\mathbf{y} = d_I(\mathbf{O^{[N]}}, \mathbf{O^{[N_c]}})$. With z a given number, denote with D_z the region of the xy plane such that $|x - y| \leq z$; thus, it is

$$F_{\mathbf{z}}(z) = \Pr\{\mathbf{z} \leq z\} = \Pr\{(\mathbf{x}, \mathbf{y}) \in D_z\} = \iint_{D_z} f_{\mathbf{xy}}(x, y)\ dx\ dy \tag{31}$$

If we suppose that $\mathbf{x}$ and $\mathbf{y}$ are statistically independent, then it is

$$f_{\mathbf{xy}}(x, y) = f_{\mathbf{x}}(x) \cdot f_{\mathbf{y}}(y) \tag{32}$$

Therefore, in order to find the distribution of the $\mathbf{z}$ RV, we have to find the density of the two RVs $\mathbf{x}$ and $\mathbf{y}$. The distribution of $\mathbf{x}$ can be obtained from the distance distribution of d_I, by taking into account that, since we already accessed node N, it is $\mathbf{x} = d_I(Q, \mathbf{O^{[N]}}) \leq S_{I\to Q}\ \epsilon + r^{[N]}$. Therefore it is

$$F_{\mathbf{x}}(x) = \begin{cases} \frac{F_{\mathbf{d_I}}(x)}{F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + r^{[N]})} & \text{if } x \leq S_{I\to Q}\ \epsilon + r^{[N]}, \\ 1 & \text{otherwise.} \end{cases} \tag{33}$$

$$f_{\mathbf{x}}(x) = \begin{cases} \frac{f_{\mathbf{d_I}}(x)}{F_{\mathbf{d_I}}(S_{I\to Q}\ \epsilon + r^{[N]})} & \text{if } x \leq S_{I\to Q}\ \epsilon + r^{[N]}, \\ 0 & \text{otherwise.} \end{cases} \tag{34}$$

Likewise, the distribution of $\mathbf{y}$ can be obtained from the distance distribution of d_I, by observing that it is $\mathbf{y} = d_I(\mathbf{O^{[N]}}, \mathbf{O^{[N_c]}}) \leq r^{[N]} - r^{[N_c]}$. Hence

$$F_{\mathbf{y}}(y) = \begin{cases} \frac{F_{\mathbf{d_I}}(y)}{F_{\mathbf{d_I}}(r^{[N]} - r^{[N_c]})} & \text{if } y \leq r^{[N]} - r^{[N_c]}, \\ 1 & \text{otherwise.} \end{cases} \tag{35}$$

$$f_{\mathbf{y}}(y) = \begin{cases} \frac{f_{\mathbf{d_I}}(y)}{F_{\mathbf{d_I}}(r^{[N]} - r^{[N_c]})} & \text{if } y \leq r^{[N]} - r^{[N_c]}, \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

By taking into account Equations 32, 34, and 36, Equation 31 can be written as:

$$\begin{aligned} F_{\mathbf{z}}(z) &= \int_0^{r^{[N]} - r^{[N_c]}} \int_{y-z}^{y+z} f_{\mathbf{xy}}(x, y)\, dx\, dy = \\ &= \int_0^{r^{[N]} - r^{[N_c]}} \int_{y-z}^{y+z} f_{\mathbf{x}}(x)\, f_{\mathbf{y}}(y)\, dx\, dy = \\ &= \int_0^{r^{[N]} - r^{[N_c]}} \frac{f_{\mathbf{d_I}}(y)}{F_{\mathbf{d_I}}(r^{[N]} - r^{[N_c]})} \int_{y-z}^{y+z} \frac{f_{\mathbf{d_I}}(x)}{F_{\mathbf{d_I}}(S_{I \to Q}\, \epsilon + r^{[N]})}\, dx\, dy = \\ &= \int_0^{r^{[N]} - r^{[N_c]}} \frac{f_{\mathbf{d_I}}(y)}{F_{\mathbf{d_I}}(r^{[N]} - r^{[N_c]}) F_{\mathbf{d_I}}(S_{I \to Q}\, \epsilon + r^{[N]})} \left(F_{\mathbf{d_I}}(y + z) - F_{\mathbf{d_I}}(y - z)\right)\, dy \end{aligned}$$

Thus, the probability that d_C has to be computed for an object in a node at level l, is

$$\begin{aligned} &F_{\mathbf{z}}(S_{I \to Q}\, \epsilon + \overline{r_{l+1}}) = \\ &\quad = \int_0^{\overline{r_l} - \overline{r_{l+1}}} \frac{f_{\mathbf{d_I}}(y)}{F_{\mathbf{d_I}}(\overline{r_l} - \overline{r_{l+1}}) F_{\mathbf{d_I}}(S_{I \to Q}\, \epsilon + \overline{r_l})} \left(F_{\mathbf{d_I}}(y + S_{I \to Q}\, \epsilon + \overline{r_{l+1}}) - F_{\mathbf{d_I}}(y - S_{I \to Q}\, \epsilon - \overline{r_{l+1}})\right)\, dy \end{aligned}$$

□

Proof of Theorem 3. For sav_{CPU}, let $\#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1)$ be the number of query distances computed when d_C is not used. Clearly, when using d_C, $\#_{d_C}(\epsilon; d_I; S_{I \to Q}) = \#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1)$ comparison distances have to be computed between the query and objects in the data set. Then, a query distance $d_Q(Q, O_i)$ has to be computed iff O_i has not been pruned by d_C, i.e. iff $d_C(Q, O_i) \leq S_{C \to Q}\, \epsilon$: This happens with probability $\Pr\{d_C(Q, O_i) \leq S_{C \to Q}\, \epsilon\} = F_{\mathbf{d_C}}(S_{C \to Q}\, \epsilon)$. Therefore, it is

$$\begin{aligned} sav_{CPU} &= \frac{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I \to Q}, 1, 1) - time_{CPU}(\epsilon; d_Q, d_I, d_C; S_{I \to Q}, S_{C \to Q}, S_{C \to I})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I \to Q}, 1, 1)} = \\ &= 1 - \frac{time_{CPU}(\epsilon; d_Q, d_I, d_C; S_{I \to Q}, S_{C \to Q} S_{C \to I})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I \to Q}, , 11)} \approx \\ &\approx 1 - \frac{time_{d_Q}(\epsilon; d_Q, d_I, d_C; S_{I \to Q}, S_{C \to Q}) + time_{d_C}(\epsilon; d_I, d_C; S_{I \to Q})}{time_{d_Q}(\epsilon; d_Q, d_I, \_; S_{I \to Q}, 1} = \\ &= 1 - \frac{\#_{d_Q}(\epsilon; d_I, d_C; S_{I \to Q}, S_{C \to Q}) \cdot cost_{d_Q} + \#_{d_C}(\epsilon; d_I; S_{I \to Q}) \cdot cost_{d_C}}{\#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1) \cdot cost_{d_Q}} = \\ &= 1 - \frac{\#_{d_Q}(\epsilon; d_I, d_C; S_{I \to Q}, S_{C \to Q}) \cdot cost_{d_Q} + \#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1) \cdot cost_{d_C}}{\#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1) \cdot cost_{d_Q}} = \\ &= 1 - \frac{F_{\mathbf{d_C}}(S_{C \to Q}\, \epsilon) \cdot \#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1) \cdot cost_{d_Q} + \#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1) \cdot cost_{d_C}}{\#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, S_{C \to I}) \cdot cost_{d_Q}} = \\ &= 1 - \frac{\#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1)(cost_{d_Q} \cdot F_{\mathbf{d_C}}(S_{C \to Q}\, \epsilon) + cost_{d_C})}{\#_{d_Q}(\epsilon; d_I, \_; S_{I \to Q}, 1) \cdot cost_{d_Q}} = \\ &= 1 - F_{\mathbf{d_C}}(S_{C \to Q}\, \epsilon) - \frac{cost_{d_C}}{cost_{d_Q}} \end{aligned}$$

As for sav_{total}, it is sufficient to note that

$$
\begin{aligned}
sav_{total} &= \frac{time_{total}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) - time_{total}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q} S_{C\to I})}{time_{total}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1)} = \\
&= \frac{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) + time_{I/O}(\epsilon; d_I; S_{I\to Q})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) + time_{I/O}(\epsilon; d_I; S_{I\to Q})} \\
&\quad - \frac{time_{CPU}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q}, S_{C\to I}) + time_{I/O}(\epsilon; d_I; S_{I\to Q})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) + time_{I/O}(\epsilon; d_I; S_{I\to Q})} = \\
&= \frac{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) - time_{CPU}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q}, S_{C\to I})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) + time_{I/O}(\epsilon; d_I; S_{I\to Q})} = \\
&= \frac{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1)}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) + time_{I/O}(\epsilon; d_I; S_{I\to Q})} \\
&\quad \cdot \frac{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1) - time_{CPU}(\epsilon; d_Q, d_I, d_C; S_{I\to Q}, S_{C\to Q} S_{C\to I})}{time_{CPU}(\epsilon; d_Q, d_I, \_; S_{I\to Q}, 1, 1)} = \\
&= \beta \cdot sav_{CPU} \approx \beta \left(1 - F_{\mathbf{d_C}}(S_{C\to Q}\ \epsilon) - \frac{cost_{d_C}}{cost_{d_Q}}\right)
\end{aligned}
$$

□

Proof of Lemma 2. Without loss of generality assume that both $F_{\mathbf{d_1}}(\cdot)$ and $F_{\mathbf{d_2}}(\cdot)$ are continuous, and let $f_{\mathbf{d_1}}(\cdot)$ and $f_{\mathbf{d_2}}(\cdot)$ be their corresponding probability density functions (pdf's). Integrating by parts Equation 22 it is obtained:

$$
\begin{aligned}
Err(d_1|d_2) &= \frac{1}{d_2^+} \int_0^{d_2^+} \left(F_{\mathbf{d_1}}(x) - F_{\mathbf{d_2}}(x)\right) dx = \\
&= \frac{1}{d_2^+} \left[\left(|x\ F_{\mathbf{d_1}}(x)|_0^{d_2^+} - \int_0^{d_2^+} x \cdot f_{\mathbf{d_1}}(x)\ dx\right) - \left(|x\ F_{\mathbf{d_2}}(x)|_0^{d_2^+} - \int_0^{d_2^+} x \cdot f_{\mathbf{d_2}}(x)\ dx\right)\right] = \\
&= \frac{1}{d_2^+} \left[\left(d_2^+ - E\left[\mathbf{d_1}\right]\right) - \left(d_2^+ - E\left[\mathbf{d_2}\right]\right)\right] = \\
&= \frac{E\left[\mathbf{d_2}\right] - E\left[\mathbf{d_1}\right]}{d_2^+}
\end{aligned}
$$

□

Proof of Corollary 3. Equation 30 can be proved as follows:

$$
\begin{aligned}
d_Q(\mathcal{Q}, O) &= CF(d_Q(Q_1, O), \ldots, d_Q(Q_p, O)) && \text{(by def. of } d_Q(\mathcal{Q}, O)) \\
&\geq CF\left(\frac{d_I(Q_1, O)}{S_{I\to Q}}, \ldots, \frac{d_I(Q_p, O)}{S_{I\to Q}}\right) && \text{(by hyp. } d_I \preceq S_{I\to Q}\ d_Q \text{ and } CF \text{ is monotonic)} \\
&\geq CF\left(\frac{d_I^{min}(Q_1, Reg(N))}{S_{I\to Q}}, \ldots, \frac{d_I^{min}(Q_p, Reg(N))}{S_{I\to Q}}\right) && \text{(by def. of } d_I^{min} \text{ and monotonicity of } CF) \\
&> \epsilon' && \text{(by hypothesis)}
\end{aligned}
$$

For the pruning of nodes using d_C it is:

$$
\begin{aligned}
d_Q(\mathcal{Q}, O) &\geq CF\left(\frac{d_I^{min}(Q_1, Reg(N))}{S_{I\to Q}}, \ldots, \frac{d_I^{min}(Q_p, Reg(N))}{S_{I\to Q}}\right) && \text{(from the previous proof)} \\
&\geq CF\left(\frac{d_C^{*}(Q_1, Reg(N))}{S_{C\to I}\ S_{I\to Q}}, \ldots, \frac{d_C^{*}(Q_p, Reg(N))}{S_{C\to I}\ S_{I\to Q}}\right) && \text{(by def. of } d_C^{*} \text{ and monotonicity of } CF) \\
&> \epsilon' && \text{(by hypothesis)}
\end{aligned}
$$

Similarly, when pruning objects:

$$
\begin{aligned}
d_Q(\mathcal{Q}, O) &= CF(d_Q(Q_1, O), \ldots, d_Q(Q_p, O)) && \text{(by def. of } d_Q(\mathcal{Q}, O)) \\
&\geq CF\left(\frac{d_C(Q_1, O)}{S_{C\to Q}}, \ldots, \frac{d_C(Q_p, O)}{S_{C\to Q}}\right) && \text{(by hyp. } d_C \preceq S_{I\to Q}\, d_Q \text{ and } CF \text{ is monotonic)} \\
&> \epsilon' && \text{(by hypothesis)}
\end{aligned}
$$

□