Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Journal of Discrete Algorithms 2 (2004) 115–134
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

Probabilistic proximity searching algorithms based on compact partitions

Benjamin Bustos [a,*], Gonzalo Navarro [b]

[a] *Department of Computer and Information Science, University of Konstanz, Universitaetstr. 10, 78457 Konstanz, Germany*
[b] *Center for Web Research, Department of Computer Science, University of Chile, Blanco Encalada 2120, Santiago, Chile*

Abstract

The main bottleneck of the research in metric space searching is the so-called curse of dimensionality, which makes the task of searching some metric spaces intrinsically difficult, whatever algorithm is used. A recent trend to break this bottleneck resorts to probabilistic algorithms, where it has been shown that one can find 99% of the relevant objects at a fraction of the cost of the exact algorithm. These algorithms are welcome in most applications because resorting to metric space searching already involves a fuzziness in the retrieval requirements. In this paper, we push further in this direction by developing probabilistic algorithms on data structures whose exact versions are the best for high dimensions. As a result, we obtain probabilistic algorithms that are better than the previous ones. We give new insights on the problem and propose a novel view based on time-bounded searching. We also propose an experimental framework for probabilistic algorithms that permits comparing them in offline mode.

Keywords: Metric spaces; Range queries; Probabilistic algorithms; Approximate algorithms; Similarity searching

1. Introduction

The concept of proximity searching has applications in a vast number of fields, for example: Multimedia databases, machine learning and classification, image quantization and compression, text retrieval, computational biology, function prediction, etc. All those ap-

* Corresponding author.
E-mail addresses: bustos@informatik.uni-konstanz.de (B. Bustos), gnavarro@dcc.uchile.cl (G. Navarro).

1570-8667/$ – see front matter
doi:10.1016/S1570-8667(03)00067-4

plications have in common that the objects of the database form a *metric space* [8], that is, it is possible to define a positive real-valued function d among the objects, called *distance* or *metric*, that satisfies the properties of *strict positiveness* ($d(x, y) = 0 \Leftrightarrow x = y$), *symmetry* ($d(x, y) = d(y, x)$), and *triangle inequality* ($d(x, z) \leqslant d(x, y) + d(y, z)$). For example, a *vector space* is a particular case of metric space, where the objects are tuples of real numbers and the distance function belongs to the L_s family, defined as $L_s((x_1, \ldots, x_k), (y_1, \ldots, y_k)) = (\sum_{1 \leqslant i \leqslant k} |x_i - y_i|^s)^{1/s}$. For example, L_1 is called the *Manhattan distance*, L_2 is the *Euclidean distance* and $L_\infty = \max_{1 \leqslant i \leqslant k} |x_i - y_i|$ is called the *maximum distance*.

One of the typical queries that can be posed to retrieve similar objects from a database is a *range query*, which retrieves all the objects within distance r to a query object q. The naive algorithm to answer range queries is to perform an exhaustive search on the database. This turns out to be too expensive for real-world applications, because the distance d is considered expensive to compute (think, for example, of a biometric device that computes the distance between two fingerprints). In many practical applications, d is so costly that the extra CPU time or even I/O time costs can be neglected. For this reason, in this paper the complexity of the algorithms will be measured as the *number of distance computations performed* to answer a query.

Proximity searching algorithms build an *index* of the database and perform range queries using this index, avoiding the exhaustive search. Many of these algorithms are based on dividing the space in *partitions* or *zones* as compact as possible. Each zone stores a representative point, called the *center*, and data that permit discarding the entire zone at query time, without measuring the actual distance from the objects of the zone to the query object, hence saving distance computations. Other algorithms are based in the use of *pivots*, which are selected objects from the database that are used together with the triangle inequality to filter out objects of the database at query time. Usually, the index is built offline and has a construction cost also measured in distance computations. The construction cost is amortized over many queries, with the saved distance computations for these.

An inherent problem of proximity searching in metric spaces is that the search becomes more difficult when the "intrinsic" dimension of the metric space increases, which is known as the *curse of dimensionality*. The intrinsic dimension of a metric space is defined in [8] as $\mu^2/2\sigma^2$, where μ and σ^2 are the mean and the variance of the distance histogram of the metric space. This is coherent with the usual vector space definition. Analytical lower bounds and experiments [8] show that all proximity searching algorithms have their performance degraded as the dimension of the space grows. This problem is due to two possible reasons: High dimensional metric spaces have a very concentrated distance histogram, which gives less information for discarding objects at query time; on the other hand, it is necessary to use a larger search radius in order to retrieve a fixed fraction of the objects of the space, because in high dimensional spaces the objects are "far away" from each other.

Probabilistic algorithms are acceptable in most applications that need to search in metric spaces, because in general modeling the problem as a metric space already involves some kind of relaxation. In most cases, finding some close objects is as good as finding all of them. There exists a pivot-based probabilistic proximity searching algorithm which largely improves the search time at the cost of missing few relevant objects [7]. On the other hand,

it is known that compact partitioning algorithms perform better than pivot-based algorithms in high dimensional metric spaces [8] and they have lower memory requirements.

In this paper, we present several probabilistic algorithms for proximity searching based on compact partitions, which alleviate in some way the curse of the dimensionality. We also present experimental results that show that these algorithms perform better than probabilistic algorithms based on pivots, and that the latter need much more memory space to outperform the former when the dimension of the space is very high.

The paper is organized as follows: In Section 2, we survey the exact algorithms for proximity search in metric spaces. In Section 3, we give an overview of the actual probabilistic algorithms. Section 4 describes the data structures where we implement our probabilistic techniques. In Section 5, we describe the proposed probabilistic algorithms, and Section 6 presents the experimental results with synthetic and real-world data sets. Section 7 introduces the model for comparing ranking criteria. Finally, in Section 8, we conclude and discuss possible extensions of this work.

2. Basic concepts

Let $(\mathbb{X}, d)$ be a metric space and $\mathbb{U} \subseteq \mathbb{X}$ the set of objects or database, with $|\mathbb{U}| = n$. There are two typical proximity searching queries:

- *Range query*. A range query (q, r), $q \in \mathbb{X}$, $r \in \mathbb{R}^+$, reports all objects that are within distance r to q, that is $(q, r) = \{u \in \mathbb{U},\ d(u, q) \leqslant r\}$.
- *k nearest neighbors query* (k-*NN*). Reports the k objects from $\mathbb{U}$ closer to q, that is, returns the set $\mathbb{C} \subseteq \mathbb{U}$ such that $|\mathbb{C}| = k$ and $\forall x \in \mathbb{C}, y \in \mathbb{U} - \mathbb{C}, d(x, q) \leqslant d(y, q)$.

The volume defined by (q, r) is called the *query ball*, and all the objects from $\mathbb{U}$ inside it are reported. Nearest neighbors queries can be implemented using range queries. Our definition of range query for metric spaces preserves the same spirit of the "geometric range query", which is defined for vector spaces as a hypercube instead of a ball. The original definition has no meaning on a metric space scenario due to the lack of coordinates.

There exist two classes of techniques used to implement proximity searching algorithms: One based on pivots and one based on compact partitions.

2.1. Pivot-based algorithms

These algorithms select a number of "pivots", and classify all the other objects according to their distance to the pivots. The canonical pivot-based algorithm is as follows: Given a range query (q, r) and a set of k pivots $\{p_1, \ldots, p_k\}$, $p_i \in \mathbb{U}$, by the triangle inequality it follows for any $x \in \mathbb{X}$ that $d(p_i, x) \leqslant d(p_i, q) + d(q, x)$, and also that $d(p_i, q) \leqslant d(p_i, x) + d(x, q)$. From both inequalities, it follows that a lower bound on $d(q, x)$ is $d(q, x) \geqslant |d(p_i, x) - d(p_i, q)|$. The objects $u \in \mathbb{U}$ of interest are those that satisfy $d(q, u) \leqslant r$, so one can exclude all the objects that satisfy $|d(p_i, u) - d(p_i, q)| > r$ for some pivot p_i (exclusion condition), without actually evaluating $d(q, u)$. Defining the metric $D(x, y) = \max_{1 \leqslant i \leqslant k} |d(x, p_i) - d(y, p_i)|$, it follows that the pivot exclusion con-

dition can be expressed as $D(q,u) > r$. Note that D is a lower bound of the d distance between q and u.

The index consists of the kn distances $d(u, p_i)$ between every object and every pivot. Therefore, at query time it is necessary to compute the k distances between the pivots and the query q in order to apply the exclusion condition. Those distance calculations are known as the *internal complexity* of the algorithm, and this complexity is fixed if there is a fixed number of pivots. The list of objects $\{u_1, \ldots, u_m\} \subseteq \mathbb{U}$ that cannot be excluded by the exclusion condition, known as the *object candidate list*, must be checked directly against the query. Those distance calculations $d(u_i, q)$ are known as the *external complexity* of the algorithm. The total complexity of the search algorithm is the sum of the internal and external complexity, $k + m$. Since one increases and the other decreases with k, it follows that there is an optimum k^* that depends on the tolerance range r of the query. In practice, k^* is so large that one cannot store the k^*n distances, and the index uses as many pivots as space permits.

Examples of pivot-based algorithms [8] are *BK-Tree*, *Fixed Queries Tree* (*FQT*), *Fixed-Height FQT*, *Fixed Queries Array*, *Vantage Point Tree* (*VPT*), *Multi VPT*, *Excluded Middle Vantage Point Forest*, *Approximating Eliminating Search Algorithm* (*AESA*) and *Linear AESA*. With a few exceptions, pivot-based algorithms select them at random among the objects of the metric space. However, it is well known that the way in which the pivots are selected can affect the performance of the algorithms. One can select a "good set" of pivots maximizing the mean of the distribution of D [5]. In our experiments, we use random pivots as well as good pivots.

2.2. Algorithms based on compact partitions

These algorithms are based on dividing the space in *partitions* or *zones* as compact as possible. Each zone stores a representative point, called the *center*, and data that permit discarding the entire zone at query time, without measuring the actual distance from the objects of the zone to the query object. Each zone can be partitioned recursively into more zones, inducing a *search hierarchy*. There are two general criteria for partitioning the space: *Voronoi partition* and *covering radius*.

2.2.1. Voronoi partition criterion

The Voronoi diagram of a collection of objects is a partition of the space into cells, each of which consisting of the objects closer to one particular center than to any other. A set of m centers is selected and the rest of the objects are assigned to the zone of their closest center. Given a range query (q, r), the distances between q and the m centers are computed. Let c be the closest center to q. Every zone of center $c_i \neq c$ which satisfies $d(q, c_i) > d(q, c) + 2r$ can be discarded, because its Voronoi area cannot intersect with the query ball. Fig. 1 shows an example of the Voronoi partition criterion. For q_1 the zone of c_4 can be discarded, and for q_2 only the zone of c_3 must be visited.

2.2.2. Covering radius criterion

The covering radius $cr(c)$ is the maximum distance between a center c and an object that belongs to its zone. Given a range query (q, r), if $d(q, c_i) - r > cr(c_i)$ then zone i

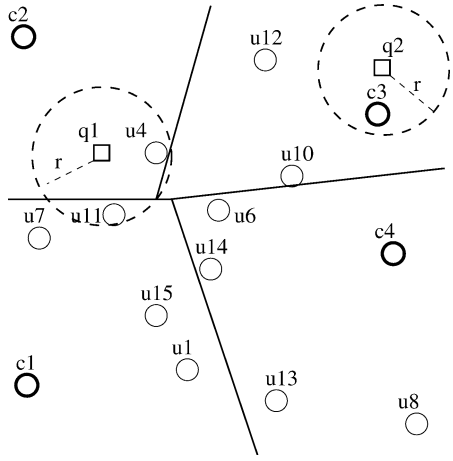

Fig. 1. Voronoi partition criterion.

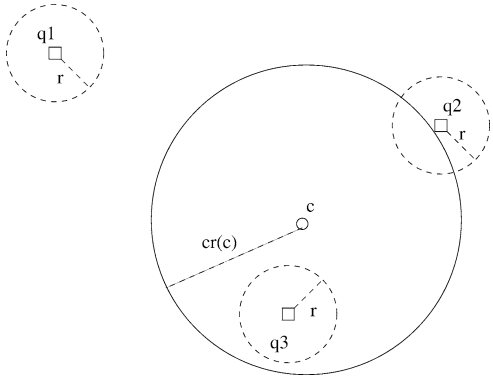

Fig. 2. Covering radius criterion.

cannot intersect with the query ball and all its objects can be discarded. In Fig. 2, the query ball of q_1 does not intersect with the zone of center c, thus it can be discarded. For the query balls of q_2 and q_3, the zone cannot be discarded, because it intersects these balls.

Generalized-Hyperplane Tree [20] is an example of an algorithm that uses the Voronoi partition criterion. Examples of algorithms that use the covering radius criterion are *Bisector Trees* (*BST*) [17], *Monotonous BST* [19], *Voronoi Tree* [13], *M-Tree* [11] and *List of Clusters* [6]. There exist algorithms that use both criteria, for example *Spatial Approximation Tree* [18] and *Geometric Near-neighbor Access Tree* [4].

3. Probabilistic algorithms for proximity searching

All the algorithms seen in the previous section are *exact algorithms*, which retrieve exactly the elements of $\mathbb{U}$ that are within the query ball of (q, r). In this work, we are

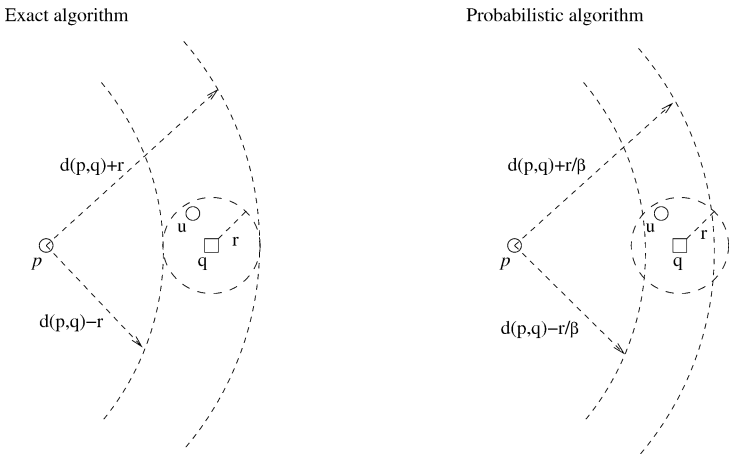

Fig. 3. How the probabilistic algorithm based on pivots works.

interested in *probabilistic algorithms*, which relax the condition of delivering the exact solution. As explained before, this is acceptable in most applications.

A survey on approximate similarity queries is presented in [10]. It proposes a classification schema for existing approaches, considering as relevant characteristics of them: Type of data (metric or vector spaces), error metrics (changing space or reducing comparisons), quality guarantees (none, deterministic or probabilistic parametric/non-parametric), and user interaction (static or interactive).

A probabilistic algorithm based on "stretching" the triangle inequality is presented in [7]. The idea is general, but it is applied to pivot based algorithms. Their analysis shows that the net effect of the technique is to reduce the search radius by a factor β, and that reduction is larger when the search problem becomes harder, i.e., the intrinsic dimension of the space becomes high. Even with very little stretching, large improvements in the search time are obtained with low error probability. The factor β can be chosen at search time, so the index can be built beforehand and later one can choose the desired level of accurateness and speed of the algorithm. As the factor is used only to discard elements, no element closer to q than r/β can be missed during the search. In practice, all the elements that satisfy $|d(p_i, u) - d(p_i, q)| > r/\beta$ for some p_i are discarded. Fig. 3 illustrates how the idea operates. The exact algorithm guarantees that no relevant element is missed, while the probabilistic one stretches both sides of the ring and can miss some elements.

A data structure called $M(\mathbb{U}, Q)$ to answer nearest neighbor queries is proposed in [12]. It requires a training data set Q of m objects, taken to be representative of typical query objects. This data structure may fail to return a correct answer, but the failure probability can be made arbitrarily small at the cost of increasing the query time and space requirements for the index. When the metric space obeys a certain sphere-packing bound [12], it is shown that $M(\mathbb{U}, Q)$ answers range queries in $O(K \ln(n) \log(\Upsilon(\mathbb{U} \cup Q)))$ time, with failure probability $O(\log^2(n)/K)$ and requires $O(Kn \log(\Upsilon(\mathbb{U} \cup Q)))$ space, where K is a parameter that allows one to control the failure probability and $\Upsilon(T)$ is the ratio of the distance between the farthest and closest pair of points of T.

An approach to approximate nearest neighbor similarity search called *probabilistic approximately correct NN* (PAC-NN) is presented in [9]. The algorithm retrieves an $(1+\varepsilon)$ nearest neighbor with probability greater or equal than $1-\delta$, where ε and δ are parameters that can be tuned at query time. The algorithm can be implemented in an arbitrary index, and in [9] both sequential and index-based PAC-NN algorithms are described. Given a query object q, r_δ^q is defined as the maximum distance from q so that the probability of finding an object closer to q than r_δ^q is lower or equal than δ. An estimation of r_δ^q can be obtained from the distance distribution of the query points. Then, the database is scanned until an object u such that $d(q,o) \leqslant (1+\varepsilon)r_\delta^q$ is found, reporting u as the probably approximately correct nearest neighbor of q. On the other hand, an $(1+\varepsilon)$ approximation is guaranteed by pruning from the search every element whose lower bound distance to q (proved by the index structure) exceeds $r^*/(1+\varepsilon)$, where r^* is the current distance to the kth nearest neighbor.

An index structure called *P-Sphere tree* for nearest neighbor queries is proposed in [14]. The tree has a two-level structure, a root level and a leaf level. The root contains a list of "sphere descriptions" and pointers to all leaf levels. Each leaf contains a *center point* and all data points that lie within the sphere described in the corresponding sphere descriptor from the root level. Three parameters must be set before constructing the tree: The fanout of the root, the center points in the sphere descriptors, and the leaf size. The search algorithm consists in determining the leaf whose center point is closest to the query object, and then a linear scan is performed on that leaf, reporting the closest object to the query. Selecting the appropriate parameters at construction time [14], which also depend in the desired accuracy level, the index will yield a probably correct answer.

Approximate k-NN queries with the M-tree are presented in [23]. Three different approximation techniques are proposed, which trade query precision for improved efficiency: Approximation through relative distance errors, approximation through distance distributions, and approximation through the slowdown of distance improvements. Experimental results suggest that the best method is the one based on distance distributions. Given the distance distribution F_q of a query object q, the stopping criterion $F_q(d(q,o_A^k)) \leqslant \rho$ can be defined, where o_A^k is the kth approximated nearest neighbor of q (as found by the search algorithm) and ρ is the fraction of best cases to which this current approximate result belongs. This criterion is used to stop the search before the exact k-NN are found. No search improvements are obtained when $\rho \leqslant F_q(d(q,o_N^k))$, where o_N^k is the actual kth nearest neighbor of q. If the distribution F_q is unknown, in [23] it is proposed to use a "representative distance function", e.g., the average distribution function defined as $F_{\text{avg}}(x) = E[F_o(x)]$.

Approximation algorithms for vector spaces are surveyed in depth in [10,21]. An example is [1], which proposes a general framework to search for an arbitrary region Q in $(\mathbb{R}^k, L_2)$. The idea is to define areas Q^- and Q^+ such that $Q^- \subset Q \subset Q^+$. Points inside Q^- are guaranteed to be reported and points outside Q^+ are guaranteed not to be reported. In between, the algorithm can err. The maximum distance between the real and the bounding areas is ε. The vector space is partitioned using trees, which are used to guide the search by including or excluding whole areas. Every decision about including (excluding) a whole area can be done using Q^+ (Q^-) to increase the probability of pruning the search in either way. Those areas that cannot be fully included or excluded are analyzed

in more detail by going down to the appropriate subtree. The complexity is shown to be $O(2^k \log(n) + (3\sqrt{k}/\varepsilon)^k)$ and a very close lower bound is proven for the problem.

In [2] is proposed a data structure called *BBD-tree* for searching in a vector space $\mathbb{R}^k$ under any metric L_s. This structure is inspired in the *kd-tree* and it can be used to find the "$(1+\varepsilon)$ nearest neighbor", that is, to find an object u^* such that $\forall u \in \mathbb{U},\ d(u^*, q) \leqslant (1+\varepsilon)d(u, q)$. The essential idea of the algorithm is to locate the query q in a cell (each leaf in the tree is associated with a cell in the space decomposition). Every point inside the cell is processed so as to obtain its nearest neighbor p. The search stops when no promising cells are found, i.e., when the radius of any ball centered at q and intersecting a nonempty cell exceeds the radius $d(q, p)/(1+\varepsilon)$. The search time for this algorithm is $O(\lceil 1 + 6k/\varepsilon \rceil^k \log(n))$.

In [22], a proposal called "aggressive pruning" for "limited radius nearest neighbors" is presented. This query seeks for nearest neighbors that are inside a given radius. The idea can be seen as a particular case of [1], where the search area is a ball and the data structure is a *kd-tree*. Relevant elements may be lost but irrelevant ones cannot be reported, i.e., $Q^+ = Q$. The ball Q, of radius r and centered at $q = (q_1, \ldots, q_k)$, is pruned by intersecting it with the area between hyperplanes $q_i - r + \varepsilon$ and $q_i + r - \varepsilon$. The authors give a probabilistic analysis assuming normally distributed distances, which almost holds if the points are uniformly distributed in the space. The search time is $O(n^\lambda)$, where λ decreases as the permitted failure probability ε increases.

4. The indexes we build on

Of all the exact algorithms presented in Section 2, two of the most efficient in high dimensions are SAT and List of Clusters. We use these indexes to implement our probabilistic algorithms, so now we briefly explain how these algorithms work.

4.1. Spatial approximation tree

The *SAT* [18] is based on approaching the query spatially rather than dividing the search space, that is, start at some point in the space and get closer to the query, which is done only via "neighbors". The SAT uses both compact partition criteria for discarding zones, it needs $O(n)$ space, reasonable construction time $O(n \log^2(n)/\log(\log(n)))$ and sublinear search time $O(n^{1-\Theta(1/\log(\log(n)))})$ in high dimensional spaces.

Construction of SAT is as follows: An arbitrary object $a \in \mathbb{U}$ is chosen as the root node of the tree (note that since there exists only one object per node, we use both terms interchangeably in this section). Then, we select a suitable set of neighbors $N(a)$, such that $\forall u \in \mathbb{U}, u \in N(a) \Leftrightarrow \forall v \in N(a) - \{u\}, d(u, v) > d(u, a)$. Note that $N(a)$ is defined in terms of itself in a non-trivial way, and that multiple solutions fit the definition. In fact, finding the minimal set of neighbors seems to be a hard combinatorial optimization problem [18]. A simple heuristic that works well in most cases considers the objects in $\mathbb{U} - \{a\}$ in increasing order of their distance from a, and adds an object x to $N(a)$ if x is closer to a than to any object already in $N(a)$. Next, we put each node in $\mathbb{U} - N(a)$ into the bag

of it closest object of $N(a)$. Also, for each subtree $u \in N(a)$, we store its covering radius $cr(u)$. The process is repeated recursively in each subtree using the objects of its bag.

This construction process ensures that if we search an object $q \in \mathbb{U}$ by spatial approximation, we will find that object in the tree because we are repeating exactly what happened during the construction process, i.e., we enter into the subtree of the neighbor closest to q, until we reach q (in fact, in this case we are doing an exact search because q is present in the tree). For general range queries (q, r), instead of simply going to the closest neighbor, we first determine the closest neighbor c of q among $\{a\} \cup N(a)$. Then, we enter into all neighbors $b \in N(a)$ such that $d(q, b) \leqslant d(q, c) + 2r$. During the search process, all the nodes x such that $d(q, x) \leqslant r$ are reported. The search algorithm can be improved a bit more: When we search for an object $q \in \mathbb{U}$ (exact search), we follow a single path from the root to q. At any node a' in this path, we choose the closest to q among $\{a'\} \cup N(a')$. Therefore, if the search is currently at tree node a, we have that q is closer to a than to any ancestor a' of a and also any neighbor of a'. Hence, if we call $A(a)$ the set of ancestors of a (including a), we have that, at search time, we can avoid entering any object $x \in N(a)$ such that $d(q, x) > 2r + \min\{d(q, c),\ c \in \{a'\} \cup N(a'),\ a' \in A(a)\}$. This condition is a stricter version of the original Voronoi partition criterion. The covering radius stored for all nodes during the construction process can be used to prune the search further, by not entering into subtrees such that $d(q, b) - r > cr(b)$.

4.2. List of Clusters

The *List of Clusters* [6] is a list of "zones". Each zone has a center and stores its covering radius. A center $c \in \mathbb{U}$ is chosen at random, as well as a radius rp, whose value depends on whether the number of objects per compact partition is fixed or not. The *center ball* of (c, rp) is defined as $(c, rp) = \{x \in \mathbb{X},\ d(c, x) \leqslant rp\}$. We then define $I = \mathbb{U} \cap (c, rp)$ as the bucket of "internal" objects lying inside (c, rp), and $E = \mathbb{U} - I$ as the rest of the objects (the "external" ones). The process is repeated recursively inside E. The construction process returns a list of triples (c_i, rp_i, I_i) (center, radius, internal bucket).

This data structure is asymmetric, because the first center chosen has preference over the next centers in case of overlapping balls. With respect to the value of the radius rp of each compact partition and the selection of the next center in the list, there exist many alternatives. In [6] it is shown experimentally that the best performance is achieved when the compact partition has a fixed number of objects, so rp becomes simply $cr(c)$, and the next center is selected as the object which maximizes the distance sum to the centers previously chosen. The brute force algorithm for constructing the list takes $O(n^2/m)$, where m is the size of the compact partition, but it can be improved using auxiliary data structures to build the partitions. For high dimensional metric spaces, the optimal m is very low (we used $m = 5$ in our experiments).

Given a range query (q, r), $d(q, c)$ is computed, reporting c if it is within the query ball. Then, we search exhaustively inside I only if $d(q, c) - cr(c) \leqslant r$. E is processed only if $cr(c) - d(q, c) < r$, because of the asymmetry of the data structure. The search cost has a form close to $O(n^\alpha)$ for some $0.5 < \alpha < 1.0$ [6].

5. Our approach

We focus on probabilistic algorithms for high dimensional metric spaces, where for exact searching it is very difficult to avoid the exhaustive search regardless of the index and search algorithm used.

It is well known that compact partition algorithms perform better than pivot-based algorithms in high dimensional metric spaces [8], and that the latter need more space requirements, i.e., many pivots, to reach the performance of the former. For this reason, it is interesting to develop probabilistic algorithms based on compact partitions, with the hope that these algorithms could have at least the same performance as pivot-based probabilistic algorithms but with less memory requirements.

We propose two probabilistic techniques, the first based on incremental searching and the second based on ranking zones.

5.1. Probabilistic incremental search

This technique is an adaptation of the *incremental nearest neighbor search* algorithm [16]. The incremental search traverses the search hierarchy defined by the index (whatever it be) in a "best-first" manner. At any step of the algorithm, it visits the "element" (zone or object) with the smallest distance from the query object among all unvisited elements in the search hierarchy. This can be done by maintaining a priority queue of elements organized by their maximum lower bound distance known to the query object at any time.

In [16] is proved that this search is *range-optimal*, that is, it obtains the kth nearest neighbor, o_k, after visiting the same search hierarchy elements as would a range query with radius $d(q, o_k)$ implemented with a top-down traversal of the search hierarchy.

The incremental nearest neighbor search can be adapted to answer range queries. We report all objects u that satisfy $d(q, u) \leqslant r$, but we stop when an element with lower bound $l > r$ is taken out of the queue (*global stopping criterion*). It is not possible to find another object within the query ball among the unexplored elements, because we have retrieved them ordered by their lower bounded distances to q. An equivalent method is to enqueue elements only if they have a lower bound $l \leqslant r$, in which case the queue must be processed until it gets empty.

The idea of the probabilistic technique based on the incremental search is to fix in advance the number of distance computations allowed to answer a range query. Using the adapted incremental search for range queries, if the search is pruned after we make the maximum number of distance computations allowed, then we obtain a probabilistic algorithm in the sense that some relevant objects can be missed. However, as the search is performed range-optimally, one can presume that the allotted distance computations are used in an efficient way.

Fig. 4 depicts the general form of the probabilistic incremental search. *Index* is the data structure that indexes $\mathbb{U}$, q is the query object, e is an element of the index and $d_{LB}(q, e)$ is a lower bound of the real distance between q and all the elements rooted in the search hierarchy of e, where $d_{LB}(q, e) = d(q, e)$ if e is an object of $\mathbb{U}$, and $d_{LB}(q, e) \geqslant d_{LB}(q, e')$ if e' is an ancestor of e in the hierarchy. For example, in the List of Clusters, if e is a child of a and belongs to the zone of center c then $d_{LB}(q, e) = d(q, c) - cr(c)$; in SAT

```
ProbabilisticIncrementalSearch(q, Index, quota)
1.  e ← root of Index
2.  counter ← 0 // Number of distances computed
3.  Q ← {(e,0)} // Priority queue
4.  while Q is not empty do
5.    (e, d_LB(q,e)) ← element in Q with lower d_LB(q,e)
6.    Q ← Q − {(e, d_LB(q,e))}
7.    if e is a zone then
8.      for each child element e' of e do
9.        cost ← cost to compute d_LB(q,e')
10.       if counter + cost ⩽ quota
11.         Compute d_LB(q,e')
12.         if d_LB(q,e') ⩽ r then
13.           Q ← Q ∪ {(e', max(d_LB(q,e), d_LB(q,e')))}
14.         counter ← counter + cost
15.       endif
16.     enddo
17.   endif
18.   else report e // object within the query ball
19. enddo
```

Fig. 4. Probabilistic incremental search algorithm.

if e is a child of a then $d_{LB}(q,e) = \max\{d(q,e) - cr(e),\ (d(q,e) - \min\{d(q,c),\ c \in \{a'\} \cup N(a'), a' \in A(a)\})/2\}$. The maximum number of distance computations allowed to perform the search is denoted by *quota*. Once *quota* has been reached, no more elements are inserted in the queue. Note that the only stopping criterion of the algorithm is that the queue gets empty, even if the work quota has been reached, because for all the objects in the queue their distances to q are already known. Variable *cost* indicates the number of distance computations needed to process a child e' of an element e in the search hierarchy. In SAT, the cost of processing all the children of e is equal to $N(e)$; in List of Clusters, this cost is equal to the size of the compact partition, m.

5.2. Ranking of zones

The probabilistic incremental search aims at quickly finding objects within the query ball, before the work quota gets exhausted. As the maximum number of distance computations is fixed, the total search time is also bounded. This technique can be generalized to what we call *ranking of zones*, where the idea is to sort the zones in order to favor the most promising and then to traverse the list until we use up the quota. The probabilistic incremental search can be seen as a ranking method, where we first rank all the zones using $d_{LB}(q,e)$ and then work until we use up the quota. However, this ranking does not have to be the best zone ranking criterion.

The sorting criterion must aim at quickly finding objects that are close to the query object. As the space is partitioned into zones, we must sort these zones in a promising search order using the information given by the index data structure. For example, in List

of Clusters the only information we have is the distances from q to each center ($d(q,c)$) and the covering radius of each zone ($cr(c)$), which is precomputed, so we estimate how promising a zone is using only $d(q,c)$ and $cr(c)$. One not only would like to search first the zones closer to the query, but also to search first the zones that are more compact, that is, the zones which have "higher object density". In spite of the fact that it is very difficult to define the volume of a zone in a general metric space, we assume that if the zones have the same number of objects, as in the best implementation of List of Clusters, then the zones with smaller covering radii have higher object density than those with larger covering radii.

We have tested several zone ranking criteria, all in ascending order:

- $d(q,c)$: The distance from q to each zone center.
- $cr(c)$: The covering radius of each zone, $cr(c)$.
- $d(q,c) + cr(c)$: An upper bound of the distance from q to the farthest object in the zone of center c.
- $d(q,c) - cr(c)$: A lower bound of the distance from q to the closest object in the zone of center c.
- $\beta(d(q,c) - cr(c))$: What we call *dynamic beta*.

The first two criteria are the simplest ones. The third criterion aims to search first in those zones that are closer to q and also are compact. The fourth criterion is similar to the probabilistic incremental search. The last technique is equivalent to reducing the search radius by a factor β as in [6], where $1/\beta \in [0..1]$. If β is fixed, then this criterion is equivalent to $d(q,c) - cr(c)$, because the ordering is the same in both cases. However, instead of using a constant factor β, we define a *dynamic factor* of the form $\beta = 1/(1.0 - \frac{cr(c)}{mcr})$, where *mcr* is the maximum size of the covering radius of all zones. This implies that we reduce the search radii more in zones of larger covering radii. A special case is when $cr(c') = mcr$ for a zone c'. In this case, we define $d_{LB}(q,e) = \infty$ for all objects in that zone. Note that $d(q,c) - cr(c)$ is the only criterion that can be used with the probabilistic incremental search, because with this criterion it is guaranteed that $d_{LB}(q,e) \geqslant d_{LB}(q,e')$ holds for any e' ancestor of e.

Each ranking criterion implements a different *node scheduling* policy. It is not clear a priori which of these schedules will have the best performance. Therefore, it is relevant to experimentally test different schedules and to compare their effectiveness.

6. Performance of the new techniques

6.1. Experimental results

We use the SAT and List of Clusters to implement the probabilistic techniques described in Section 5, but with SAT we only implement the probabilistic incremental search because in this data structure every node is a center, so it takes $O(n)$ time to compute the distances between the query and every center. We have tested the probabilistic techniques on a synthetic set of random points in a k-dimensional vector space treated as a metric space, that is, we have not used the fact that the space has coordinates, but treated the points as abstract

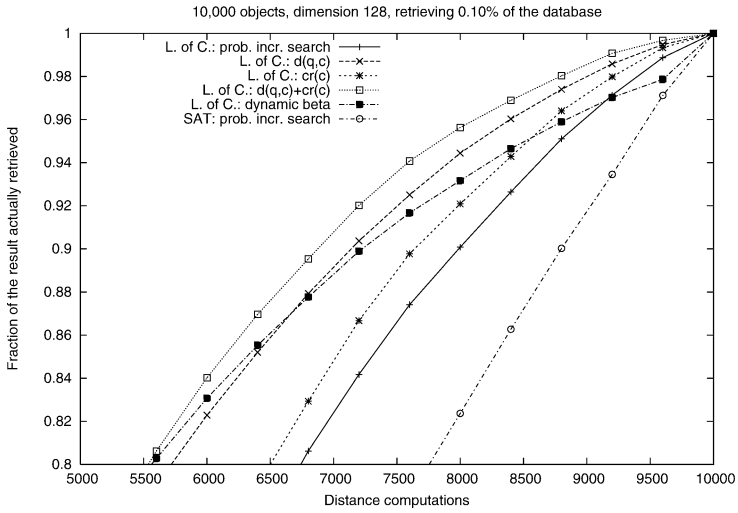

Fig. 5. Probabilistic List of Clusters and SAT in a vector space of dimension 128.

objects in an unknown metric space. The advantage of this choice is that it allows us to control the exact dimensionality we are working with, which is very difficult to do in general metric spaces. The points are uniformly distributed in the unitary cube, our tests use the L_2 (Euclidean) distance, the database size is $n = 10{,}000$ and we perform range queries returning 0.10% of the total database size, taking an average from 1,000 queries. The techniques were tested using a space of dimension 128, where no known exact algorithm can avoid an exhaustive search to answer useful range queries.

Fig. 5 shows the results of the probabilistic List of Clusters and SAT. The curves represent the fraction of the result actually retrieved (that is, the fraction of relevant objects retrieved) as a function of the number of distances computations allowed to perform the search. The best technique, in this experiment, is the ranking zone method with criterion $d(q,c) + cr(c)$.

Fig. 6 shows a comparison of the probabilistic List of Clusters and the probabilistic pivot-based algorithm, implemented in its canonical form (see Sections 2.1 and 3). In this experiment, the performance of the probabilistic List of Clusters is almost equal to the pivot-based algorithm with 256 pivots when more than 97% of the result is actually retrieved. The pivot-based techniques are slightly better when the pivots are selected using the "good pivots" criterion [5]. However, the size of the List of Clusters index (0.12 Mb) is about 82 times less than the size of the pivot-based index with 256 pivots (9.78 Mb) and about 5 times less than the size of the pivot-based index with 16 pivots (0.62 Mb). Experiments with different search radii and database sizes obtained similar results to those presented here.

One of the most clear applications of metric space techniques to Information Retrieval is the task of finding documents relevant to a query (which can be a set of terms or a whole document itself) [3]. Documents (and queries) are seen as vectors, where every term is a coordinate whose value is the weight of the term in that document. The distance between

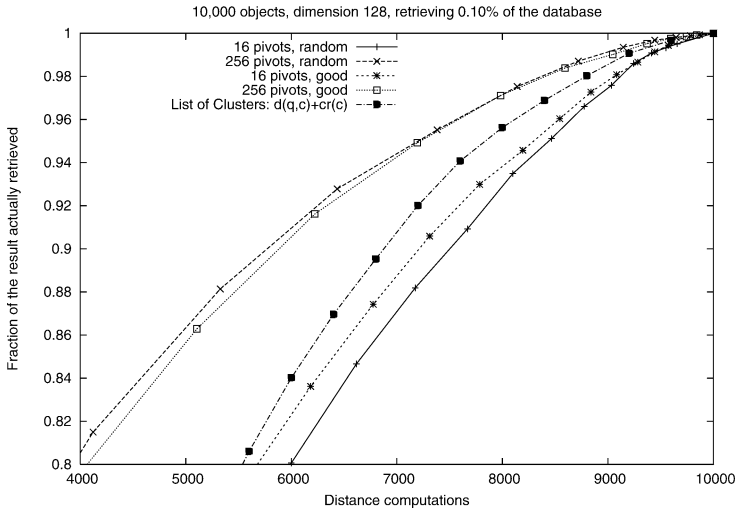

Fig. 6. Comparison among probabilistic algorithms in a vector space of dimension 128.

two documents is the angle between their vectors, so documents sharing important terms are seen as more similar. Documents closer to a query are considered to be more relevant to the query. Hence the task is to find the elements of this metric space of documents which are closest to a given query.

Despite of this clear link, metric space techniques have seldom been used for this purpose. One reason is that the metric space of documents has a very high dimension, which makes impractical any exact search approach. This is a case where probabilistic algorithms would be of great value, since the definition of relevance is fuzzy and it is customary to permit approximations. Fig. 7 shows the result of an experiment testing the zone ranking criteria on a subset of the TREC-3 collection [15]. The database consisted on 24,960 documents, and we average over 1,000 query documents chosen at random from the original subset ($m = 10$ for the List of Clusters, retrieving on average 0.035% of the database per query). The results show that, for this experiment, the best criteria for ranking zones is the dynamic beta and $d(q, c)$.

Fig. 8 shows a result comparing the pivot-based algorithm with the ranking zone method using the dynamic beta criterion. The results show that our probabilistic algorithms can handle better this space, retrieving more than 99% of the relevant objects and traversing merely a 17% of the database, using much less memory, approximately 16 times less than the index with 64 pivots, hence becoming for the first time a feasible metric space approach to this long standing problem.

6.2. Ranking of zones versus ranking of objects

The sorting criterion $d(q, c) - cr(c)$ can be modified to take advantage of the information provided by the List of Clusters data structure. If for each zone, in addition to the covering radius, we store the distances from its center c to all the objects u_i that belongs to this zone, then we can obtain an improved lower bound of the distance from q to u_i, which

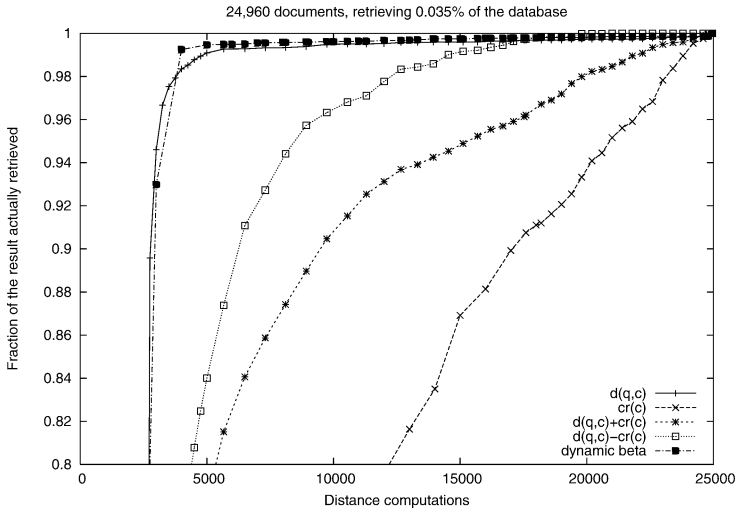

Fig. 7. Comparison among ranking criteria in a document database.

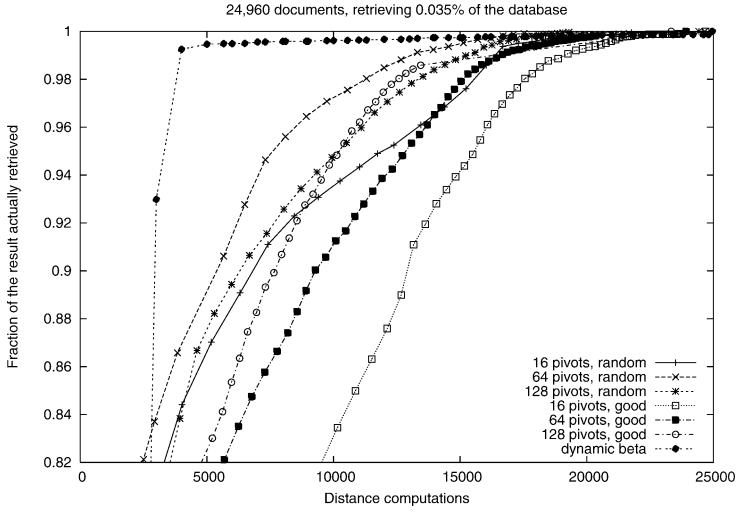

Fig. 8. Comparison among probabilistic algorithms in the document database.

is $d(q,c) - d(c,u_i)$. Therefore, a variant of the original criterion is to sort the objects according to the values given by the improved lower bound. Note that in this variant we are not ranking zones, but each object of the database.

However, in practice this variant results in no improvements over the original technique, but the opposite. The comparison between both techniques and the dynamic beta criterion is shown in Fig. 9. The dynamic beta criterion has still a far superior performance than the other criteria. This is an unexpected result. We conjecture that the reason of the bad

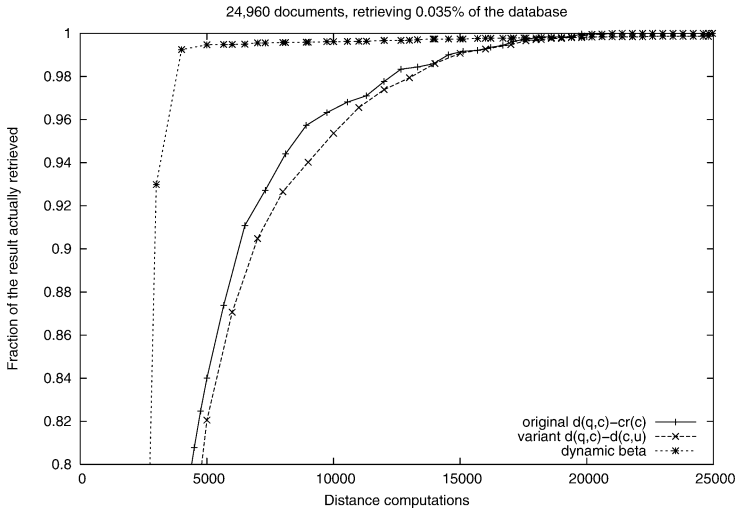

Fig. 9. Comparison between ranking of objects and ranking of zones.

performance of the ranking of objects is that we lose valuable clustering information when we rank each object separately.

Another possibility for ranking objects instead of zones is using a pivot-based index. The ranking in this case consists of sorting the objects by increasing D distance (see Section 2.1) to the query, and then search in that order, stopping when the work quota is over or when the distance D is greater than r. Fig. 10 shows the results of an experiment in the document database, using different number of pivots. The results show that this method is quite competitive, but it is outperformed by the dynamic beta criterion when retrieving more than 99% of the relevant documents. We also compared the difference between random and good pivots index. The result shows that the use of good pivots increases the performance of this sorting criterion.

7. A model for comparing ranking criteria

Now we describe a model for ranking criteria comparison, which allows us to compare different ranking criteria in an offline mode, without having to repeat each experiment for each different pair of parameters.

Let $\mathbb{U}$ be a database with $|\mathbb{U}| = n$. For a given set Q of k queries, each query is performed using some criterion without work limit. We save the order in which elements were retrieved and their distance to the query object. With this information, we generate a *cloud of points* which is represented in a graph *distance to the query as a function of the number of distances computations*. The X axis range is $[0, n]$ and the Y axis range is $\mathbb{R}^+$. If object u was retrieved after performing i distance computations, then the point $(i, d(q, u))$ is added to the cloud. This procedure is repeated for all objects retrieved in all the queries, totalizing kn points. Fig. 11 shows an example of a cloud of points.

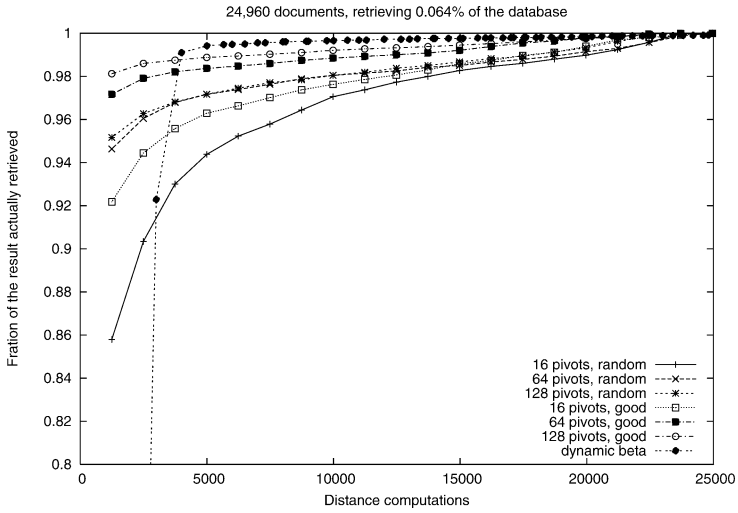

Fig. 10. Ranking of objects using a pivot-based index.

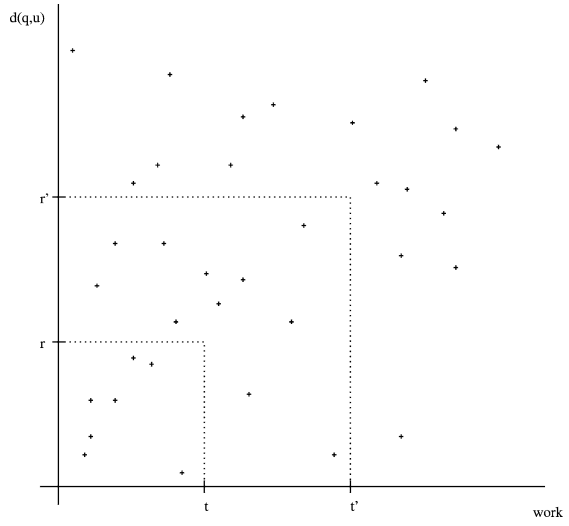

Fig. 11. Example of a cloud of points for a given criterion.

This cloud of points allows us to simulate any experiment on the preprocessed query set, varying the allowed amount of work or the search radius. For example, if one wants to know how many relevant objects the algorithm would retrieve on average with a search radius r and a work quota t, then one just has to count the points (x, y) of the cloud which satisfy $x \leqslant t$ and $y \leqslant r$, and then divide this quantity by the total number of queries, k. Let $A(t, r)$ be the resulting value. Since that all distances between objects and queries are known, it is easy to know how many objects are within a query ball for a fixed search

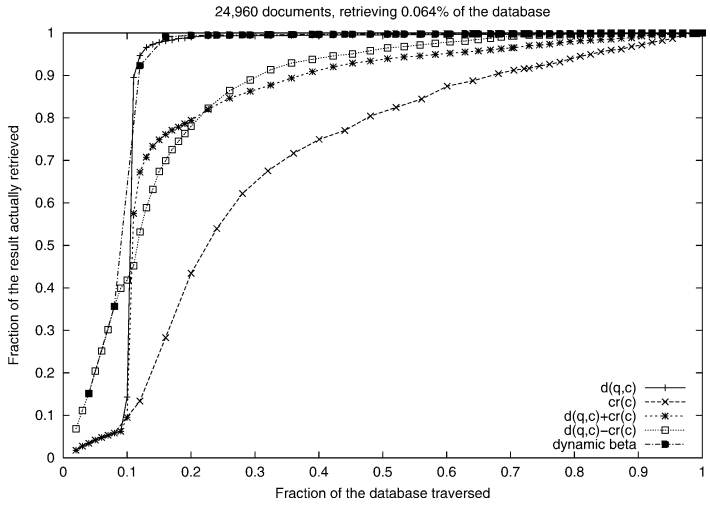

Fig. 12. Fraction of the retrieved objects as a function of the traversed fraction of the database, 0.064% retrieved.

radius, which turns out to be $A(\infty, r)$. Then, the fraction f of retrieved relevant objects using a work quota t is $f = A(t, r)/A(\infty, r)$.

The procedure described can be repeated for different r' and t' values. If the search radius is fixed and one computes f for different amounts of work quota, then we can obtain several points of the cost function for a specific criterion. Fig. 12 shows the results obtained with a traditional experiment, and Fig. 13 shows the results obtained with 100 queries, using the comparison model. There are just minor differences between both figures.

The disadvantage of this comparison model is that it needs to save huge amounts of information, because each query contributes with an amount of data proportional to $|\mathbb{U}|$. This can be solved using s discrete values for $d(q, u)$ and defining a matrix of $s \times n$ storage cells for the discrete values of $(i, d(q, u))$. With this approach, the space cost is st, but some precision will be lost when computing $A(t, r)$.

8. Conclusions

We have defined a general probabilistic technique based on the incremental nearest neighbor search, that allows us to perform time-bounded range search queries in metric spaces with a high probability of finding all the relevant objects. We also defined a probabilistic technique based on ranking zones, which is a generalization of the former technique. Our experimental results show in both synthetic and real-world examples that the best criteria for ranking zones perform better than the pivot-based probabilistic algorithm in high dimensional metric spaces, as the latter needs much more memory space to be competitive. Also, we studied variants of this technique which rank objects instead of zones, but our experimental results show that these variants make no improvement over the ranking of zones technique.

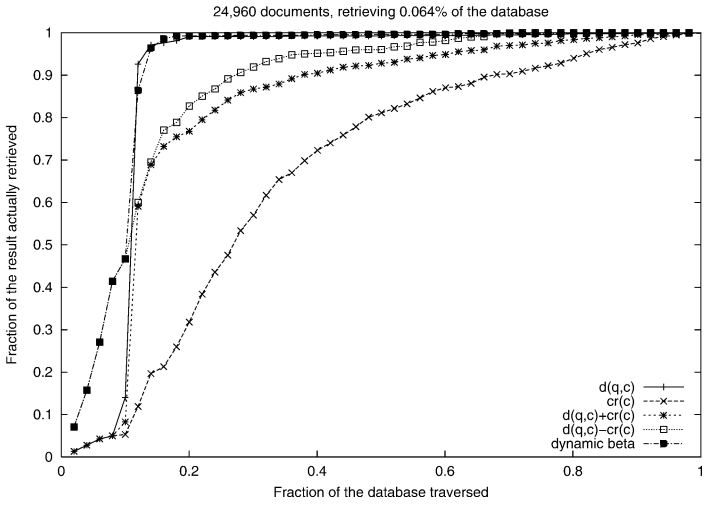

Fig. 13. Result using the comparison model, 0.064% retrieved.

According to the schema proposed in [10], the probabilistic techniques proposed in this paper can be classified as methods that are applicable on metric spaces, that reduce the number of comparisons performed during the search (in fact, this value is fixed in the algorithms), that give no guarantee on the error introduced by the approximation, and that allow the user to interactively set the parameters (amount of work to perform during the search) to tune the quality of the answer set. Our techniques can be seen as a practical realization of the theoretical framework introduced with the PAC approach [9]. Our contributions in this respect have been to empirically compare specific index structures and specific schedules, which was not done previously. Moreover, we have proposed a new way to regard the problem, as a time-bounded computation, and have devised a technique to simplify experimentation in this area.

Future work involves testing more zone ranking criteria and to use more advanced clustering techniques for testing our probabilistic search algorithms. Based on the results obtained with the document database, the ranking of zones seems to be a promising alternative as a ranking method for effective and efficient similarity searching for Information Retrieval applications. It would be interesting to compare the effectiveness of our ranking technique against the traditional approaches in terms of precision versus recall figures.

Acknowledgements

This work was partially supported by the German Science Foundation (DFG), project no. KE 740/6-1 of the strategic research initiative SPP 1041 (first author), and by the Millennium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile (second author). The first author is on leave from the Department of Computer Science, University of Chile.

References

[1] S. Arya, D. Mount, Approximate range searching, in: Proc. 11th Annual ACM Symposium on Computational Geometry, 1995, pp. 172–181.
[2] S. Arya, D. Mount, N. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimension, in: Proc. 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94), 1994, pp. 573–583.
[3] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley, Reading, MA, 1999.
[4] S. Brin, Near neighbor search in large metric spaces, in: Proc. 21st Conference on Very Large Databases (VLDB'95), Morgan Kaufmann, 1995, pp. 574–584.
[5] B. Bustos, G. Navarro, E. Chávez, Pivot selection techniques for proximity searching in metric spaces, Pattern Recognition Letters 24 (14) (2003) 2357–2366.
[6] E. Chávez, G. Navarro, An effective clustering algorithm to index high dimensional metric spaces, in: Proc. 7th Symposium on String Processing and Information Retrieval (SPIRE'00), IEEE CS Press, 2000, pp. 75–86.
[7] E. Chávez, G. Navarro, Probabilistic proximity search: fighting the curse of dimensionality in metric spaces, Inform. Process. Lett. 85 (2003) 39–46.
[8] E. Chávez, G. Navarro, R. Baeza-Yates, J. Marroquín, Searching in metric spaces, ACM Comput. Surv. 33 (3) (2001) 273–321.
[9] P. Ciaccia, M. Patella, PAC nearest neighbor queries: approximate and controlled search in high-dimensional and metric spaces, in: Proc. 16th International Conference on Data Engineering (ICDE'00), 2000, pp. 244–255.
[10] P. Ciaccia, M. Patella, Approximate similarity queries: a survey, Technical Report CSITE-08-01, Department of Electronics, Computer Science and Systems, University of Bologna, May 2001.
[11] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: Proc. 23rd Conference on Very Large Databases (VLDB'97), Morgan Kaufmann, 1997, pp. 426–435.
[12] K. Clarkson, Nearest neighbor queries in metric spaces, Discrete Comput. Geom. 22 (1) (1999) 63–93.
[13] F. Dehne, H. Noltemeier, Voronoi trees and clustering problems, Inform. Syst. 12 (2) (1987) 171–175.
[14] J. Goldstein, R. Ramakrishnan, Contrast plots and P-sphere trees: space vs. time in nearest neighbor searches, in: Proc. 26th International Conference on Very Large Databases (VLDB'00), Morgan Kaufmann, 2000, pp. 429–440.
[15] D. Harman, Overview of the third text REtrieval conference, in: Proc. Third Text REtrieval Conference (TREC-3), 1995, pp. 1–19, NIST Special Publication 500-207.
[16] G. Hjaltason, H. Samet, Incremental similarity search in multimedia databases, Technical Report TR 4199, Department of Computer Science, University of Maryland, November 2000.
[17] I. Kalantari, G. McDonald, A data structure and an algorithm for the nearest point problem, IEEE Trans. Software Engrg. 9 (5) (1983) 631–634.
[18] G. Navarro, Searching in metric spaces by spatial approximation, VLDB J. 11 (1) (2002) 28–46.
[19] H. Noltemeier, K. Verbarg, C. Zirkelbach, Monotonous Bisector* Trees—a tool for efficient partitioning of complex schemes of geometric objects, in: Data Structures and Efficient Algorithms, in: Lecture Notes in Comput. Sci., vol. 594, Springer, Berlin, 1992, pp. 186–203.
[20] J. Uhlmann, Satisfying general proximity/similarity queries with metric trees, Inform. Process. Lett. 40 (1991) 175–179.
[21] D. White, R. Jain, Algorithms and strategies for similarity retrieval, Technical Report VCL-96-101, Visual Computing Laboratory, University of California, La Jolla, California, July 1996.
[22] P. Yianilos, Locally lifting the curse of dimensionality for nearest neighbor search, in: Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA'00), 2000, pp. 361–370.
[23] P. Zezula, P. Savino, G. Amato, F. Rabitti, Approximate similarity retrieval with M-trees, VLDB J. 7 (4) (1998) 275–293.