Probabilistic Proximity Search: Fighting the Curse of Dimensionality in Metric Spaces

Edgar Chávez [a] Gonzalo Navarro [b]

[a] *Escuela de Ciencias Físico-Matemáticas, Univ. Michoacana, Morelia, Mich. México.* elchavez@zeus.ccu.umich.mx.

[b] *Dept. of Computer Science, Univ. of Chile, Santiago, Chile.* gnavarro@dcc.uchile.cl.

Abstract

Proximity searches become very difficult on "high dimensional" metric spaces, that is, those whose histogram of distances has a large mean and/or a small variance. This so-called "curse of dimensionality", well known in vector spaces, is also observed in metric spaces. The search complexity grows sharply with the dimension and with the search radius. We present a general probabilistic framework applicable to any search algorithm and whose net effect is to reduce the search radius. The higher the dimension, the more effective the technique. We illustrate empirically its practical performance on a particular class of algorithms, where large improvements in the search time are obtained at the cost of a very small error probability.

Key words: Metric spaces, proximity searching, nearest neighbor searching, distance based indexing, probabilistic algorithms.

1 Introduction

The concept of "proximity" searching has applications in a vast number of fields. Some examples are multimedia databases, data mining, machine learning and classification, image quantization and compression, text retrieval, computational biology and function prediction, just to name a few.

⋆ Supported by CYTED VII.19 RIBIDI Project (both authors), CONACyT grant R-28923A (first author) and FONDECYT Grant 1-000929 (second author).
⋆⋆An earlier version of this work appeared in *Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01), LNCS 2153, pp. 147-160 2001.*

All those applications have some common characteristics. There is a universe $\mathbb{X}$ of *objects*, and a nonnegative *distance function* $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$ defined among them. This distance satisfies the three axioms that make the set a *metric space*: strict positiveness ($d(x, y) = 0 \Leftrightarrow x = y$), symmetry ($d(x, y) = d(y, x)$) and triangle inequality ($d(x, z) \leq d(x, y) + d(y, z)$). This distance is assumed to be computationally expensive (e.g., comparing two fingerprints). We have a finite *database* $\mathbb{U} \subseteq \mathbb{X}$, of size n, which is a subset of $\mathbb{X}$. The goal is to preprocess $\mathbb{U}$ to answer (with as few distance computations as possible) *fixed radius queries* and *nearest neighbor* queries. We are interested in this work in the former, expressed as $(q, r)_d$ (a point q in $\mathbb{X}$ and a tolerance radius r), which should retrieve all the points at distance r or less from q, i.e. $\{u \in \mathbb{U},\ d(u, q) \leq r\}$. Nearest-neighbor queries retrieve the K elements of $\mathbb{U}$ that are closest to q.

A particular case of metric spaces are *k-dimensional vector spaces*, $\mathbb{X} = \mathbb{R}^k$ using Minkowski's L_s distance. In this case the objects have a geometric meaning and the coordinate information can be used to guide the search. Effective methods to search vector spaces are kd-trees [4] and R-trees [10], among many others. However, for random vectors on more than about 20 dimensions, all those structures cease to work well. Proven lower bounds exist [8] that show that the search complexity is exponential with the dimension.

It is interesting to point out that the concept of "dimensionality" can be translated into metric spaces as well. The histogram of distances of a high dimensional vector space has a large mean and normally a small relative variance. In [7] this is used to define the *intrinsic* dimension of a general metric space as $\rho = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and variance of the histogram of distances. Under this definition, a database of random k-dimensional vectors with uniformly distributed coordinates has intrinsic dimension $\rho = \Theta(k)$. Hence, the definition extends naturally that of vector spaces.

Analytical lower bounds and experiments in [7] show that all the search algorithms degrade as ρ increases. The problem has received the name of *curse of dimensionality*. In terms of the histogram, we see two reasons for it. First, if ρ increases because σ^2 is reduced, then most distances tend to give the same values and hence yield less information (think on the degenerate space $d(x, y) =$ if $x = y$ then 0 else 1). Second, if ρ increases because μ grows, then a larger search radius r is necessary to retrieve a fixed fraction of the database (and also to get a constant number of nearest neighbors). The search cost also grows sharply when the search radius increases.

An interesting question is whether a probabilistic or approximate algorithm can break or at least alleviate the curse of dimensionality. These algorithms are acceptable in most applications, because in general the modelization as a metric space already carries some kind of relaxation, so finding some close elements is usually as good as finding all of them. This is our focus.

2 Related Work and Our Contribution

Most existing search algorithms for metric spaces are *exact*, that is, they retrieve the exact set $\{u \in \mathbb{U},\ d(q,u) \leq r\}$. A recent survey of these algorithms is [7]. In this work we focus on approximate and probabilistic algorithms, which relax the requisite of delivering the exact solution. A precision parameter ε measures how much may the outcome differ from the correct result.

Approximation algorithms are surveyed in depth in [14]. An example is [1], which proposes a general framework to search for an arbitrary region Q in real-valued vector spaces $(\mathbb{R}^k, L_2)$. The idea is to define areas Q^- and Q^+ such that $Q^- \subseteq Q \subseteq Q^+$. Points inside Q^- are guaranteed to be reported and points outside Q^+ not to be reported. In between the algorithm can err. The maximum distance between the real and the bounding areas is ε.

To illustrate the idea, one of the many trees used to decompose the space is 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 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.

Probabilistic algorithms have been proposed only for nearest neighbor searching, for vector spaces in [2,15,14], and for general metric spaces in [9].

In [15], 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.

In [9], the author chooses a "training set" $\mathbb{Q}$ of queries and builds a data structure to answer correctly only queries of the training set. The idea is that this setup is enough to answer correctly, with high probability, an arbitrary query. Under reasonable probabilistic assumptions it is shown that, paying $O(Kn\alpha)$ space and $O(K\alpha \log n)$ search time, the probability of not finding the nearest neighbor is $O((\log n)^2/K)$. Here α is the logarithm of the ratio between the farthest and the nearest pairs of points in $\mathbb{U} \cup \mathbb{Q}$.

In this paper we present a probabilistic technique for fixed radius searching on general metric spaces. We exploit the high dimension of the metric space, specifically the fact that the difference between random distances is small compared to a random distance. We show that this permits reducing the search radius and yet losing very few elements, and explain how any exact algorithm can make use of this property to become a much more efficient probabilistic algorithm. We exemplify the approach with a particular algorithm, which incidentally leads to a metric space version of [15]. We present empirical results showing a large increase in the search efficiency making very few errors.

3 Stretching the Triangle Inequality

A large class of algorithms to search metric spaces, called "pivot based" [7], are built on a single general idea. We select k random elements (*pivots*) $\{p_1, \ldots, p_k\} \subseteq \mathbb{U}$. The database is preprocessed to build a table of nk entries that stores the distances $d(u, p_i)$ for every $u \in \mathbb{U}$ and pivot p_i. When a query (q, r) is submitted, we compute $d(q, p_i)$ for every pivot p_i and then discard elements $u \in \mathbb{U}$ by using the triangle inequality. Two facts hold:

$$d(u,p_i) \;\leq\; d(u,q) \;+\; d(q,p_i) \quad \text{and} \quad d(q,p_i) \;\leq\; d(q,u) \;+\; d(u,p_i) \qquad (1)$$

which can be reexpressed as $d(q,u) \geq |d(u,p_i) - d(q,p_i)|$. Hence, we can discard all those u such that $|d(u,p_i) - d(q,p_i)| > r$ for *some* pivot p_i. The elements of $\mathbb{U}$ that cannot be discarded using this rule are directly compared against q.

Different pivot based algorithms share this principle and differ in the way they reduce the CPU cost incurred apart from that of computing distances. Trees, binary search and tries are some of the techniques used [7,13,6,3,5,12]. In this work we focus on reducing the number of distance computations and disregard extra CPU cost. Any known technique can be used to reduce the latter.

More precisely, let us define

$$D_k(q,u) \quad = \quad \max_{i \in 1 \ldots k} \; |d(u,p_i) - d(q,p_i)|$$

and hence we discard any u such that $D_k(q,u) > r$. The k distances $d(q,p_i)$ computed are called *internal evaluations*, while the $d(q,u)$ computed against those u that cannot be ruled out ($D_k(q,u) \leq r$) are called *external evaluations*. As the latter decrease (or at least do not increase) with k, it follows that there is an optimum k. In most cases, however, kn reaches the space limit well before k reaches its optimum, so one uses as many pivots as space permits.

Figure 1 illustrates a useful cost model. Let X be a random variable for the distance $d(x,y)$ in $\mathbb{X}$ and Z be a random variable for the distance $D_k(x,y)$.

Their distributions f_X and f_Z are illustrated, and let us call F_X and F_Z their cumulative distributions. To retrieve a fraction τ of the database we need to use a search radius large enough to make $F_X(r) \geq \tau$. On the other hand, since we discard elements u such that $D_k(q, u) > r$, our external complexity is $n \times F_Z(r)$. That is, in order to retrieve the double grayed area in Figure 1 we have to examine all the single grayed area. Clearly $D_k(x, y) \leq d(x, y)$, and the ideal situation is that both distributions are as close as possible. When this happens, the external complexity is the size of the result.

Note that the mean of X is μ, while the mean of Z is related to σ and independent of μ, as it is a maximum over differences of distances. In high dimensions (large $\rho = \frac{\mu^2}{2\sigma^2}$) the ratio between both means increases, which means that the distribution of Z falls behind that of X. Hence in order to retrieve the same τ it is necessary to examine a larger fraction of the database. To avoid this problem we can increase k to shift f_Z to the right, but this is limited by the amount of memory available and by the increase in internal complexity.

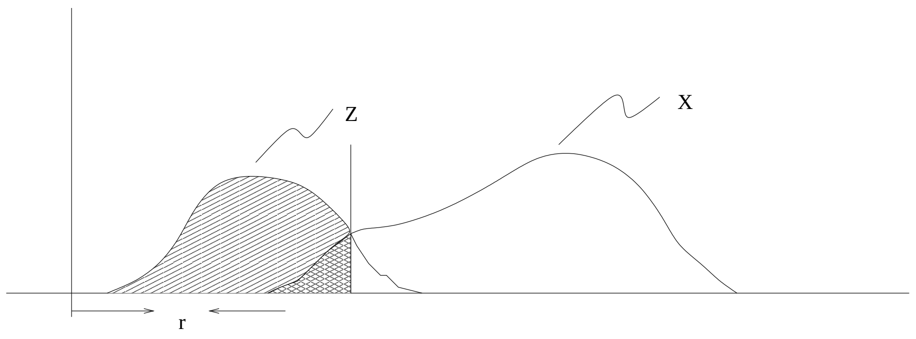


Fig. 1. The search complexity of pivoting algorithms.

The histogram comparison sketched above accurately predicts the search cost: To retrieve $nF_X(r)$ database elements we must pay $k + nF_Z(r)$ distance computations. The fact that we examine all the area where $D_k(q, u) \leq r$ is justified because $D_k(x, y)$ is upper bounded by $d(x, y)$, so we cannot miss any element. A natural question towards a probabilistic search algorithm is: How frequently does $D_k(x, y)$ really reach $d(x, y)$? Or, alternatively, how many elements would we lose if we searched with radius αr, for $0 < \alpha < 1$?

If we consider that the mean of X is μ and that of Z is related to σ, then $D_k(x, y)$ becomes much smaller than $d(x, y)$ as the dimension ρ grows. This means that, for higher dimension (or for a poorer index) we can "safely" use a smaller α. Hence the method promises to work better as the dimension grows or for worse indexes. However, for a finer analysis we cannot rely on whole histograms but on individual measures.

Consider the random variable $0 \leq Q \leq 1$ taking values in the quotient $D_k(x, y)/d(x, y)$. Figure 2 depicts the behavior of $F_Q(\alpha)$ by estimating its histogram for different k values on the metric space $([0,1)^{16}, L_2)$, with elements and pivots chosen at random.

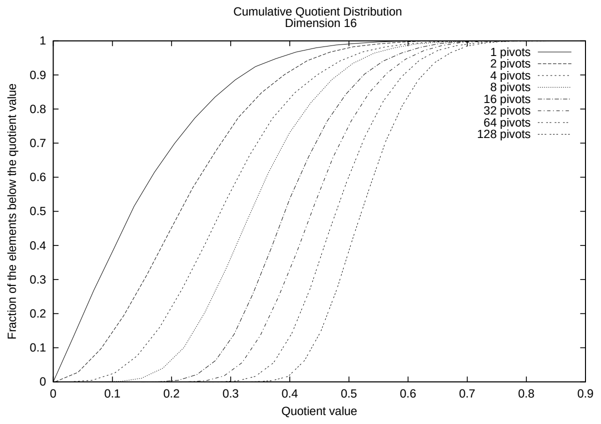


Fig. 2. The cumulative empirical distribution of Q for different instances of the mapping

Let us interpret the meaning of Q with the help of Figure 2. With a confidence level γ (determined solely by $D_k(x,y)$ and $d(x,y)$) we can assume a smaller upper bound for $D_k(x,y)$ in terms of the original distance $d(x,y)$. For example, in Figure 2, for 16 pivots we can fix a factor of $\alpha = 0.5$ with a confidence level $\gamma > 0.9$. In other words, for this particular selection, in 9 out of each 10 cases, if we measure a distance $D_k(x,y)$, the actual distance $d(x,y)$ for the pair (x,y) will be at least twice $D_k(x,y)$. We may take advantage of this observation by discarding elements using a smaller radius, knowing that the probability of missing a relevant element is at most 0.1.

In standard searching we perform the query $(q,r)_{D_k}$ to obtain a candidate list. Since $D_k(x,y) \leq d(x,y)$ we are sure that $(q,r)_d \subseteq (q,r)_{D_k}$ for any r. Then, we examine the elements in the candidate list using d. Using the statistics of Q we design a probabilistic generalization of the above procedure, which finds a subset of the correct answer $(q,r)_d$. Its confidence level (or probability of success) γ is defined as the probability that a given element in $(q,r)_d$ is actually found. The method simply obtains $\alpha = F_Q^{-1}(\gamma)$ and builds the candidate list using the stricter query $(q, \alpha r)_{D_k}$. The rest is unchanged.

It is clear that the probability of losing a relevant element u is that of $D_k(q,u) > \alpha r$ given that $d(q,u) \leq r$. Since this implies $D_k(q,u)/d(q,u) > \alpha$, we have that this occurs with probability at most γ. Note, however, that for accurateness we should compute $F_Q(\alpha)$ over pairs (x,y) such that $d(x,y) \leq r$. Some interesting properties of the algorithm are: (i) it is one-sided error, since it never report elements that should not be reported; (ii) elements at distance at most αr from q are guaranteed to be reported; (iii) we can choose γ and α at query time; and (iv) basically no modifications to the indexing and searching algorithms are necessary. The main remaining question is which tradeoff we obtain between speed and accuracy.

4 Efficiency and the Statistical Model

Using the model described in the previous section we can accurately predict the behavior of the probabilistic algorithm for a given pivot based index. The cost of satisfying $(q,r)_d$ exactly using an index with k pivots is $k + nF_Z(r)$. To satisfy a query with probability $\gamma = F_Q(\alpha)$ we have to pay $k + nF_Z(\alpha r) = k + nF_Z(F_Q^{-1}(\gamma)\ r)$.

We show experimentally how our prediction works. We use $([0,1)^{dim}, L_2)$ as our metric spaces, with uniformly distributed coordinates for the vectors. The advantage of this space is that we can know its exact dimensionality. Our search radius retrieves 1% of the database, $r = F_X^{-1}(0.01)$. Figure 3 (top) shows the experimental probability of retrieving a relevant element as α increases, using $k = 1$ and $k = 64$ pivots. Note that we lose less elements when the dimension is higher or when we have less pivots, as expected.

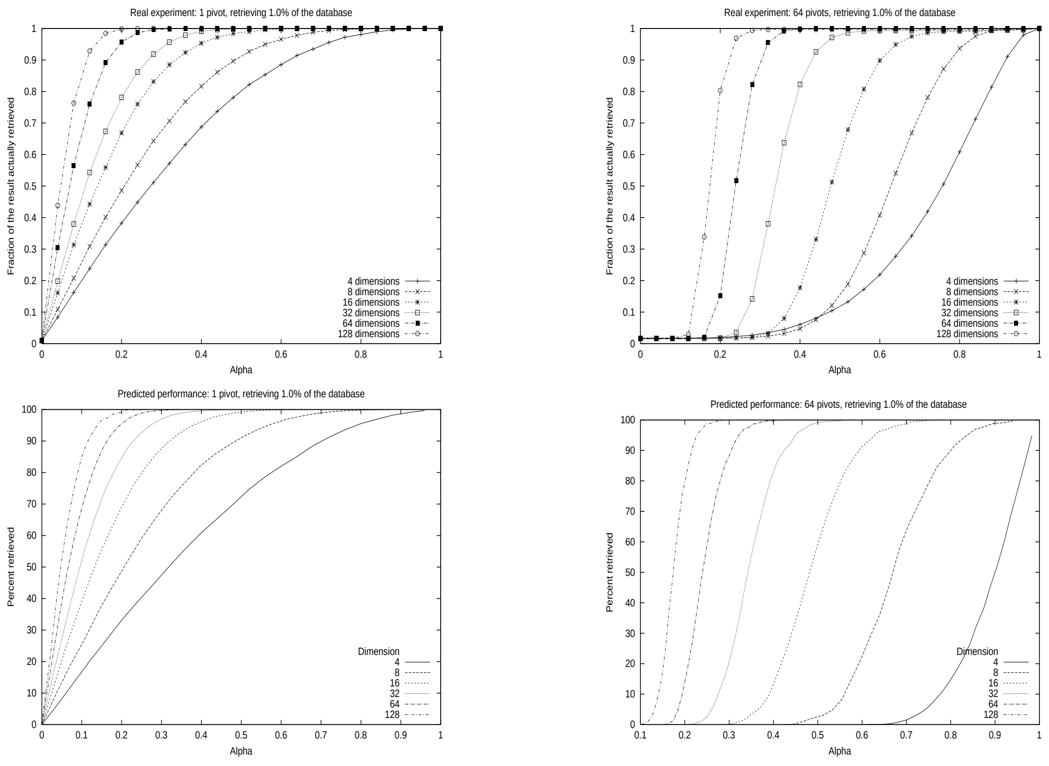


Fig. 3. Fraction of relevant elements retrieved as a function of α. On top the experimental results, on the bottom the predicted behavior.

Figure 3 (bottom) shows the prediction obtained using our model, just by plotting the cumulative distribution F_Q. The prediction is not perfect because we have estimated F_Q on arbitrary pairs (x,y) instead of over pairs at distance $\leq r$. However, the prediction is very accurate in the region of interest $\gamma \geq 0.9$, so we favor this simpler model.

Let us now compare the search cost against the probability of success. Figure 4 shows the number of comparisons as a function of the fraction of relevant elements retrieved, for different combinations of k and r. As can be seen, we retrieve even 90% or 95% of the relevant elements paying much less than the exact algorithm ($\alpha = 1$ in the plots). In many cases there is a large difference between the costs to retrieve 99% and 100% of the set. These differences are more notorious when k is too low to get good results with the exact algorithm. We can obtain the same efficiency of the exact algorithm using much less pivots. For example, 16 dimensions is almost intractable for the exact algorithm with less than 256 pivots, while with the probabilistic algorithm we can get acceptable results with 16 pivots.

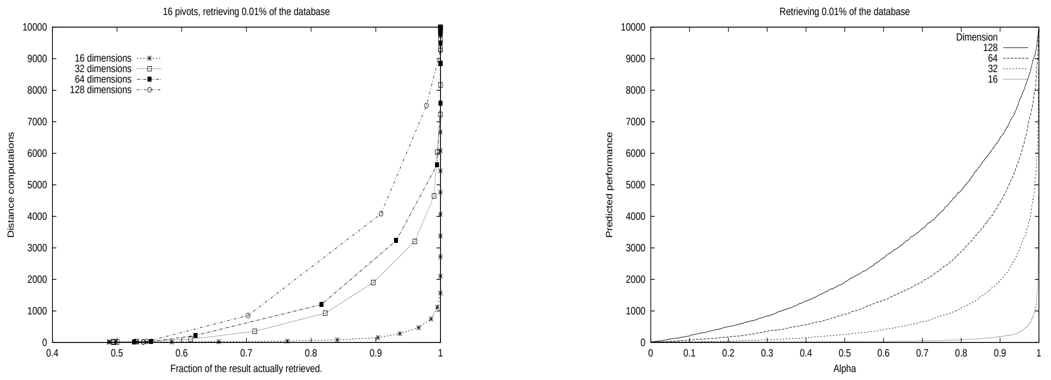


Fig. 4. Number of distance evaluations as a function of the fraction of elements retrieved, for different dimensions. On the left, the actual experiment, on the right, the predicted behavior.

Figure 5 (left) shows the effect of different search radii, retrieving from 0.01% to 5% of the database. Note that increasing the search radius has an effect similar to that of increasing the dimension.

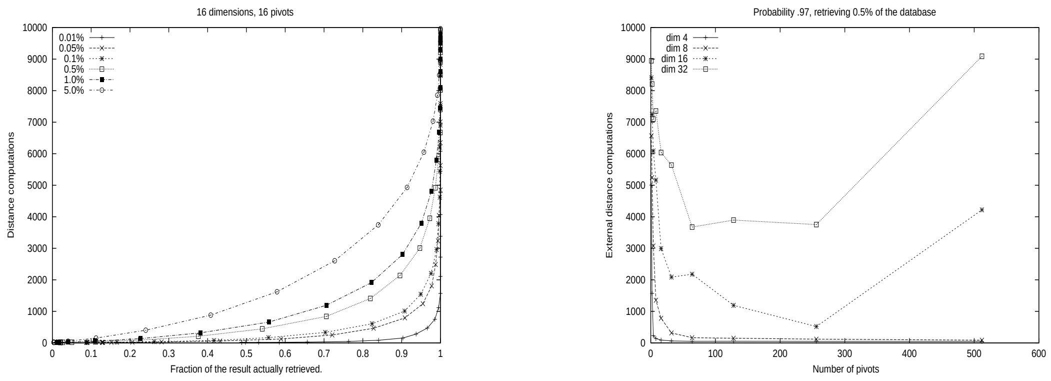


Fig. 5. On the left, number of distance evaluations as a function of the fraction of elements retrieved, for different radii. On the right, external complexity as a function of the number of pivots used, for $\gamma = 0.97$ and a radius that should retrieve 0.5% of the dataset.

Figure 5 (right) shows that the external complexity is not monotonously decreasing with k, as it is for the exact algorithm. This is because, as k increases, the probability of missing a relevant answer grows, and therefore we need a larger α to keep the same error probability. This, in turn, increases the search time. This fact worsens in higher dimensions: if we use enough pivots so as to fight the high dimension, then the error probability goes up. Therefore, the scheme does also get worse as the dimension grows. However, it worsens much slower than the exact algorithm.

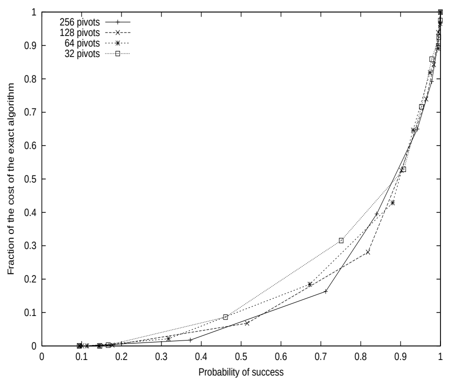
5 Real-life Examples

We show the performance of our method on two real applications. The first one is a database of text lines from the Federal Register collection of TREC-3 [11]. We used *edit distance*: the minimum number of character insertions, deletions and substitutions to make the two strings equal. This model is commonly used in text retrieval, signal processing and computational biology applications.

Figure 6 (left) shows the results for different k values. As can be seen, with a moderately high probability (more than 0.8) we improve the exact algorithm by a factor of 3. The exact algorithm examines around 90% of the database, and our probabilistic approach around 26%. Again, there is an optimum k that depends on the desired γ.

The second experiment takes the documents of the same collection and builds a metric space using the cosine distance, which is heavily used in information retrieval to determine documents relevant to a query. In this distance documents are seen as vectors in a space where the terms are the coordinates and the value of document i along the coordinate of term j is defined as $f_{i,j}\log(N/n_j)$, where $f_{i,j}$ is the number of occurrences of j in i, N is the total number of documents, and n_j is the number of documents where term j appears. The distance between two vectors is the cosine of the angle between their vectors.

Figure 6 (right) shows the result for $k = 64$ pivots. The problem is totally intractable with 64 pivots using the exact algorithm, i.e., the index examines 99.99% of the set. However, our probabilistic algorithm performs much better. With success probability 0.8 we pay a small fraction of the exact algorithm: 2% to 12% depending on the search radius. If we require the probability of success to be 0.9, the cost reaches 40%-50% of the exact algorithm, which is still very good.



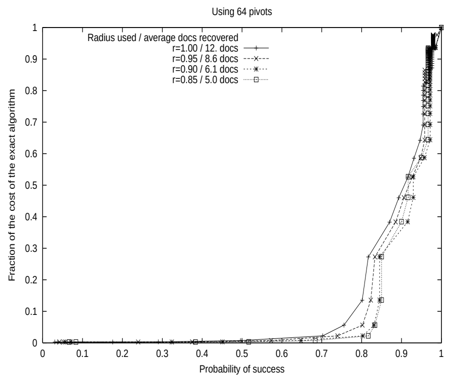


Fig. 6. Performance of our approach versus the exact algorithm. On the left, for the edit distance on text lines, on the right, for the cosine distance.

6 Conclusions

We have presented a probabilistic algorithm to search metric spaces and a predictive model able to describe both the performance of the algorithm and the expected probability of success. The algorithm can be used on any index. For the particular case of pivot based indexes we have shown that the model accurately predicts the behavior of the algorithm and permits fine tuning of the parameters by using simple statistics.

Even with very low error probability we obtain large improvements in the search time. The idea can also be used to search semimetric spaces, where the triangle inequality "almost" holds: We can relax the condition to eliminate candidates instead of tightening it as in the present work.

We are currently working on applying the idea to other data structures, in particular to clustering algorithms [7]. As explained in Section 3, this involves defining D appropriately.

References

[1] S. Arya and D. Mount. Approximate range searching. In *Proc. 11th Annual ACM Symposium on Computational Geometry*, pages 172–181, 1995.

[2] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. In *Proc. SODA'94*, pages 573–583, 1994.

[3] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. CPM'94*, LNCS 807, pages 198–212, 1994.

[4] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering*, 5(4):333–340, 1979.

[5] E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *Proc. SPIRE'99*, pages 38–46. IEEE CS Press, 1999.

[6] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: a fast and economical data structure for proximity searching. *Multimedia Tools and Applications*, 14(2):113–135, 2001.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 2001. To appear.

[8] B. Chazelle. Computational geometry: a retrospective. In *Proc. ACM STOC'94*, pages 75–94, 1994.

[9] K. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Computational Geometry*, 22(1):63–93, 1999.

[10] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD'84*, pages 47–57, 1984.

[11] D. Harman. Overview of the Third Text REtrieval Conference. In *Proc. TREC-3*, pages 1–19, 1995. NIST Special Publication 500-207.

[12] L. Micó, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Patt. Recog. Lett.*, 17:731–739, 1996.

[13] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear preprocessing-time and memory requirements. *Patt. Recog. Lett.*, 15:9–17, 1994.

[14] D. White and R. Jain. Algorithms and strategies for similarity retrieval. Tech. Rep. VCL-96-101, Visual Comp. Lab., Univ. of California, 1996.

[15] Peter N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. Tech. Rep., NEC Research Institute, Princeton, NJ, 1999.