A Probabilistic Spell for the Curse of Dimensionality [*]

Edgar Chávez[†] Gonzalo Navarro[‡]

Abstract

Range searches in metric spaces can be very difficult if the space is "high dimensional", i.e. when the histogram of distances has a large mean and a small variance. The so-called "curse of dimensionality" (well known in vector spaces) is also observed in metric spaces, and the term refers to the odd situation where using an index for proximity searching may be worse (in total elapsed time) than an exhaustive search. There are at least two reasons behind the curse of dimensionality: a large search radius and/or a high intrinsic dimension of the metric space. We present a general probabilistic framework that allows us to reduce the effective search radius of the search and which gets more effective as the dimension grows. The technique is based on probabilistically stretching the triangle inequality. We apply the technique to a particular class of indexing algorithms, even though it could be generalized to any indexing algorithm using the triangle inequality. We present an approximate analysis of the technique which helps understand the process, as well as empirical evidence showing dramatic improvements in the search time at the cost of a very small error probability.

1 Introduction and Related Work

The concept of "proximity" searching has applications in a vast number of fields. Some examples are non-traditional databases (where the concept of exact search is of no use and we search for similar objects, e.g. databases storing images, fingerprints or audio clips); machine learning and classification, image quantization and compression, text retrieval, computational biology and function prediction, just to name a few.

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 considered expensive to compute (think, for instance, in comparing two fingerprints). We have a finite *database* $\mathbb{U} \subseteq \mathbb{X}$, of size n, which is a subset of the universe of objects. The goal is to preprocess the database $\mathbb{U}$ to quickly answer (i.e. with as few distance computations as possible) *range queries* and *nearest neighbor* queries. We are interested in this work in range queries, expressed as (q,r) (a point in $\mathbb{X}$ and a tolerance radius), which should retrieve all the points at distance r or less from q, i.e. $\{u \in \mathbb{U} \;/\; d(u,q) \leq r\}$. The set of points of $\mathbb{X}$ that are at distance at most r to q is called the "query ball", so (q,r) is the intersection of the query ball and $\mathbb{U}$.

A particular case of this problem arises when the space is $\mathbb{R}^k$. There are effective methods for this case, such as kd-trees [2] or R-trees [9]. However, for more than roughly 20 dimensions those structures cease to work well. We focus in this paper in general metric spaces, although the solutions are well suited also for k-dimensional vector spaces.

In [7] a number of approaches to solve the problem of proximity searching in general metric spaces are considered, which are divided in two classes:

[*] This work has been partially supported by CYTED VII.13 AMYRI Project (both authors), CONACyT grant R-28923A (first author) and FONDECYT Project 1-000929 (second author).

[†] Escuela de Ciencias Físico-Matemáticas, Universidad Michoacana, Edificio "B", Ciudad Universitaria, Morelia, Mich. México 58000. elchavez@zeus.ccu.umich.mx

[‡] Department of Computer Science, University of Chile, Blanco Encalada 2120, Santiago, Chile. gnavarro@dcc.uchile.cl

- *Pivot-based algorithms:* which select a number of "pivots" from the database and classify all the other elements according to their distances to the pivots. The distances between elements and pivots and between the query q and the pivots are used together with the triangle inequality to filter out elements of the database without actually measuring their distance to q. These algorithms generally improve as more pivots are added, although the space requirements of the indexes increase as well.

- *Clustering algorithms:* which divide the set into spatial zones which are as compact as possible, and are able to discard complete zones by performing few distance evaluations (e.g. between the query q and a centroid of the zone). The partition into zones can be hierarchical, but the indexes use a fixed amount of memory and cannot be improved by giving them more space.

It is interesting to notice that the concept of "dimensionality" can be translated to metric spaces as well: the typical feature in high dimensional spaces is that the probability distribution of distances among elements has a very concentrated histogram (with larger mean as the dimension grows). We use in this paper a definition of the intrinsic dimensionality proposed in [5, 7]:

Definition: The *intrinsic dimensionality* of a database in a metric space is defined as $\rho = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and variance of its histogram of distances.

Under this definition, a database obtained as a uniformly distributed sample of a random k dimensional vector space has intrinsic dimension $\Theta(k)$, so the definition extends naturally that of vector spaces. As shown in [5, 7] by analytical lower bounds and experiments, all the algorithms degrade systematically as the intrinsic dimension ρ of the space increases. This is because a concentrated histogram of distances gives less information. In the extreme case we have a space where $d(x,x) = 0$ and $\forall y \neq x$, $d(x,y) = 1$, where it is impossible to avoid a single distance evaluation at search time.

The problem is so hard that it has received the name of "curse of dimensionality". An interesting question is whether a probabilistic algorithm could break the curse of dimensionality or at least alleviate it. Approximate and probabilistic algorithms are acceptable in many applications that search in metric spaces, because in general the modelization as a metric space carries already some kind of relaxation. For many applications, finding some close elements is as good as finding all them.

This relaxation uses additionally to the query a precision parameter ε to control how far away (in some sense) we want the outcome of the query from the correct result. A reasonable behavior for this type of algorithms is to asymptotically approach to the correct answer as ε goes to zero, and complementarily to speed up the algorithm, losing precision, as ε moves in the opposite direction. This definition encompasses approximate and probabilistic algorithms.

Approximation algorithms are considered in depth in [11]. An example is [1], which proposes a data structure for real vector spaces under any Minkowski metric L_s. The structure, called the BBD-tree, is inspired in *kd*-trees and can be used to find "$(1+\varepsilon)$ nearest neighbors": instead of finding u such that $d(u,q) \leq d(v,q)$ $\forall v \in \mathbb{U}$, they find an element u^*, an $(1+\varepsilon)$-nearest neighbor, differing from u by a factor of $(1+\varepsilon)$, i.e. u^* such that $d(u^*,q) \leq (1+\varepsilon)d(v,q)$ $\forall v \in \mathbb{U}$.

Probabilistic algorithms have been proposed for nearest neighbor searching only, for vector spaces in [1, 12, 11], and for general metric spaces in [8].

A good example of a probabilistic algorithm for vector spaces is [12]. The data structure is like a standard *kd*-tree using hyperplanes to recursively partition the vector space. The author uses "aggressive pruning" to improve the performance. The idea is to increase the number of branches pruned at the expense of losing some candidate points in the process. The data structure is useful for finding limited radius nearest neighbors, i.e. nearest neighbors within a fixed distance to the query. Finally, an example of a probabilistic nearest neighbor algorithm for general metric spaces is that of [8]. The author chooses a "training set" of queries and builds a data structure able to answer correctly only queries belonging to the training set. The idea is that this setup is enough to answer correctly, with high probability, an arbitrary query using the training set information. Under some probabilistic assumptions on the distribution of the queries, it is shown that the probability of not finding the nearest neighbor is $O((\log n)^2/K)$, where K can be made arbitrarily large at the expense of $O(Kn\rho)$ space and $O(K\rho \log n)$ expected search time. Here ρ is the logarithm of the ratio between the farthest and the nearest pairs of points in the union of the database $\mathbb{U}$ and the training set.

In this paper we present a probabilistic technique for range searching on general metric spaces. We exploit the intrinsic dimension of the metric space in our favor, specifically the fact that on high dimensions the difference between random distances is small. Every algorithm to search in metric spaces can make use of this property in one form or another in order to be converted into a much more efficient probabilistic algorithm. In particular, we choose the most basic pivot based algorithm (where it is easier to compute the general probability of making a mistake) and apply the technique to it. We show analytically that the net effect is a reduced search cost, corresponding to searching with a smaller radius. Reducing the search radius has an effect which is very similar to that of reducing the dimensionality of the metric space. We also present empirical results showing a dramatic increase in the efficiency of the algorithm at a very moderate error probability. An early version of this paper is a technical report [6].

2 Stretching the Triangle Inequality

Pivot based algorithms [10, 3, 4] are built on a single general idea. We select k random elements $\{p_1, \ldots, p_k\} \subset \mathbb{U}$, called *pivots*. The set is preprocessed so as to build a table of nk entries, where all the distances $d(u, p_i)$ are stored for every $u \in \mathbb{U}$ and every pivot p_i. When a query (q, r) is submitted, we compute $d(q, p_i)$ for every pivot p_i and then try to discard elements $u \in \mathbb{U}$ by using the triangle inequality on the information we have. Two facts can be used:

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

which can be reexpressed as $d(q,u) \;\geq\; |d(u,p_i) - d(q,p_i)|$, and therefore we can discard all those u where $|d(u,p_i) - d(q,p_i)| > r$. Note that by storing the table of kn distances and by computing the k distances from q to the pivots we have enough information to carry out this elimination. On the other hand, the elements u which cannot be eliminated with this rule have to be directly compared against q.

The k distance computations $d(q, p_i)$ are called *internal evaluations*, while those computations $d(q, u)$ against elements u that cannot be ruled out with the pivot information are called *external evaluations*. It is clear that, as k grows, the internal evaluations grow and the external ones decrease (or at least do not increase). It follows that there is an optimum k. However, it is well known [7] that even for moderately high dimensions the optimal k is so large that the index requires impractical amounts of memory. Therefore, in practice one uses the largest k that the available space permits. Figure 1 illustrates the curse of dimensionality for pivot based algorithms.

An interesting fact is that the difference between two random distances $d(u, p_i)$ and $d(q, p_i)$ can be seen as a random variable which distributes more or less like the histogram (indeed, with mean zero and variance twice that of the histogram). This means that, as the intrinsic dimension grows, the difference $|d(u,p_i) - d(q,p_i)|$ is normally very small.

We take advantage of this fact by "stretching" the difference between both distances. That is, we pick a $\beta \geq 1$ and multiply the difference between the distances by β before using it. In practice,

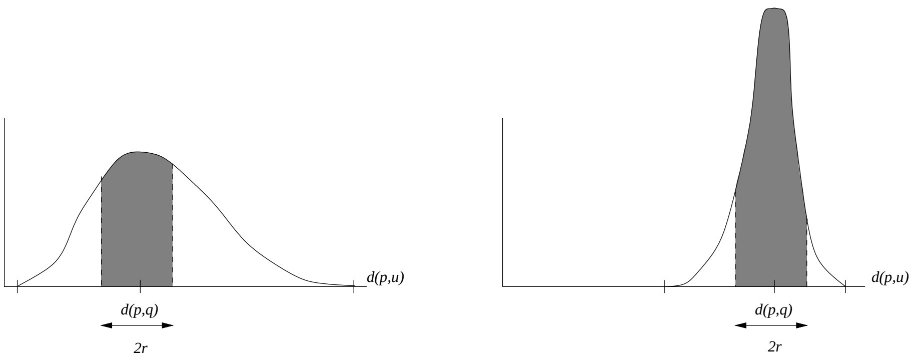

Figure 1: The histograms of distances for a low and a high dimensional metric space. The grayed parts are those that a pivot based algorithm cannot discard.

we discard all the elements which satisfy

$$|d(u,p_i) - d(q,p_i)| \quad > \quad r/\beta$$

Figure 2 illustrates the normal and the probabilistic algorithm. The normal one selects a ring which guarantees that no relevant element is left out, while the probabilistic one stretches both sides of the ring and can miss some elements.

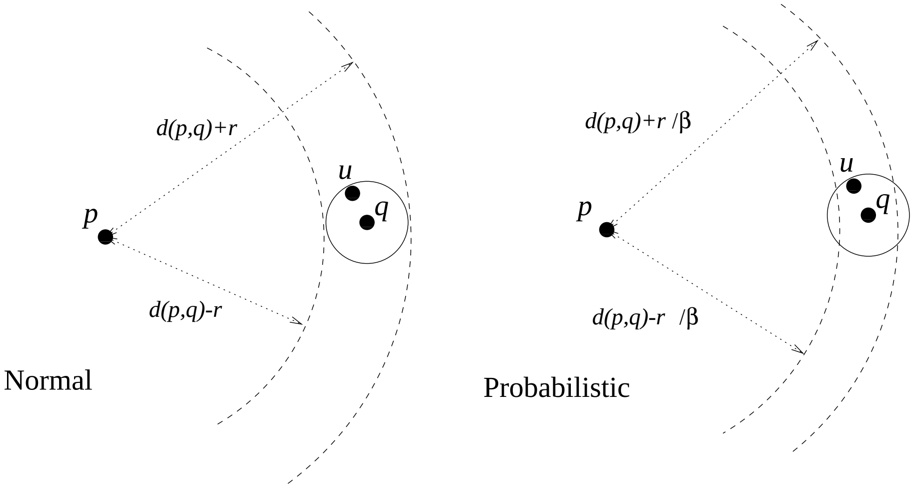

Figure 2: The rings of the elements that are not discarded under the normal and the probabilistic algorithm.

All the rest of the algorithm stays unchanged. The aim of the next sections is to analyze and test the resulting efficiency and probability of mistakes. In the final part of this paper we will discuss an alternative approach using directly the triangle inequality to derive a more general discarding rule.

3 Probability of Error

The algorithm we have presented is probabilistic with one-sided error, in the sense that it can fail to report some elements which are indeed at distance r or less to q, but it cannot report elements that are outside the query ball. In this section we derive the maximum β that can be used in order to have a probability not larger than ε of missing a relevant answer.

We are firstly interested in computing an approximation of the probability of incorrectly discarding an element u which should be returned by the query (q,r). Since $d(q,u) \leq r$, then by the triangle inequality we have $|d(q,p) - d(u,p)| \leq r$. Let us define two random variables $X = d(q,p)$ and $Y = d(u,p)$. It is easy to see that both variables are distributed according to the histogram of the metric space, since they correspond to distances among random elements. However, there is a positive covariance between X and Y.

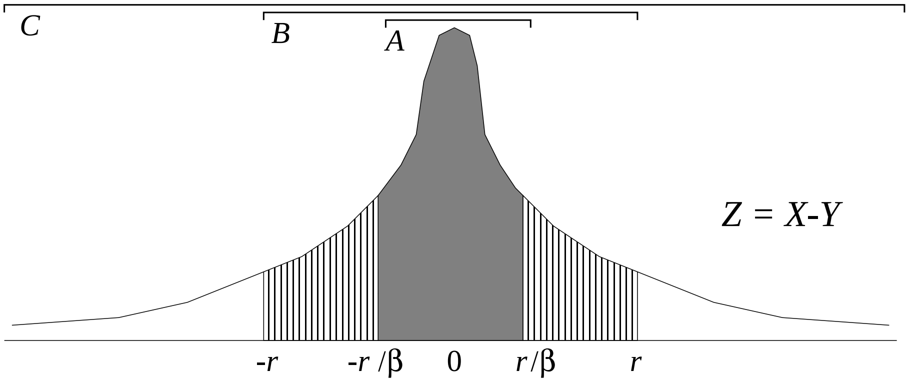

Figure 3: The histogram of the differences between two random distances and our model to adapt it when they have the constraint of differing in at most r.

Our question is therefore which is the probability that $|X - Y| > r/\beta$. We use a simplified model of the situation in order to obtain a reasonable approximation to the probability of error. Let us consider the random variable $Z = X - Y$, which has mean zero and whose variance would be $2\sigma^2$ if X and Y were independent. We assume that the distribution of Z is like the corresponding to independent X and Y, but chopping out the areas $|Z| > r$.

Figure 3 shows the model we use to derive an approximation to the error probability. We assume that the distribution of Z is like that corresponding to independent X and Y, but chopping out the areas $|Z| > r$. The remaining part is called B in the figure. The probability of *not* making a mistake corresponds to the area marked A in the figure, whose relative mass with respect to the total is A/B.

We can compute upper bounds for the relative mass of A and B with respect to the total C by using Chebyschev inequality (for a random variable V, $Pr(|V - \mu_v| > x) < \sigma_v^2/x^2$). That is, we know

$$1 - \frac{A}{C} \;=\; Pr(|Z| > r/\beta) \;<\; \frac{2\sigma^2}{(r/\beta)^2} \quad \text{and} \quad 1 - \frac{B}{C} \;=\; Pr(|Z| > r) \;<\; \frac{2\sigma^2}{r^2}$$

and therefore

$$Pr(error) \;=\; 1 - \frac{A}{B} \;=\; 1 - \frac{A}{C}\,\frac{C}{B} \;\approx\; 1 - \frac{1 - \frac{2\sigma^2}{(r/\beta)^2}}{1 - \frac{2\sigma^2}{r^2}} \;=\; \frac{2\sigma^2(\beta^2 - 1)}{r^2 - 2\sigma^2} \qquad (2)$$

which is useful for $r > \sqrt{2}\sigma$. Note that the probability goes to zero as β goes to 1.

We want that the probability of missing a relevant element after using the k pivots is at most ε. Since any pivot can discard an element u, the probability of incorrectly discarding u has to consider the union of all the probabilities. What we want is

$$1 - (1 - Pr(error))^k \;\leq\; \varepsilon$$

which, by using our approximation of Eq. (2) yields a lower bound on β:

$$\beta \;\leq\; \sqrt{1+\varepsilon_k\left(\frac{r^2}{2\sigma^2}-1\right)} \tag{3}$$

where we have defined $\varepsilon_k = 1-(1-\varepsilon)^{1/k}$, which goes from 0 to 1 together with ε, but at a faster pace that depends on k. The limit on β goes to 1 as ε goes to zero. The bound improves as the search radius grows. To show that the bound improves as the intrinsic dimension of the space grows, we reexpress r as $\mu-\sigma/\sqrt{f}$, which by Chebyschev is the minimum necessary value to return a fraction f of the set $\mathbb{U}$. This yields

$$\beta \;\leq\; \sqrt{1+\varepsilon_k\left(\left(\sqrt{\rho}-\frac{1}{\sqrt{2f}}\right)^2-1\right)} \tag{4}$$

Hence, for a fixed error probability ϵ, the bound gets more permissive as the problem becomes harder (larger ρ or f).

Figure 4 shows the probability of retrieving a relevant element as $1/\beta$ grows. We show the effect with 1 and 64 pivots. The metric space is formed by 10,000 random vectors in the cube $[0,1)^{dim}$ with the Euclidean distance. Despite that this is in particular a vector space, we treat it as a general metric space. The advantage of using a vector space is that we can precisely control its dimensionality.

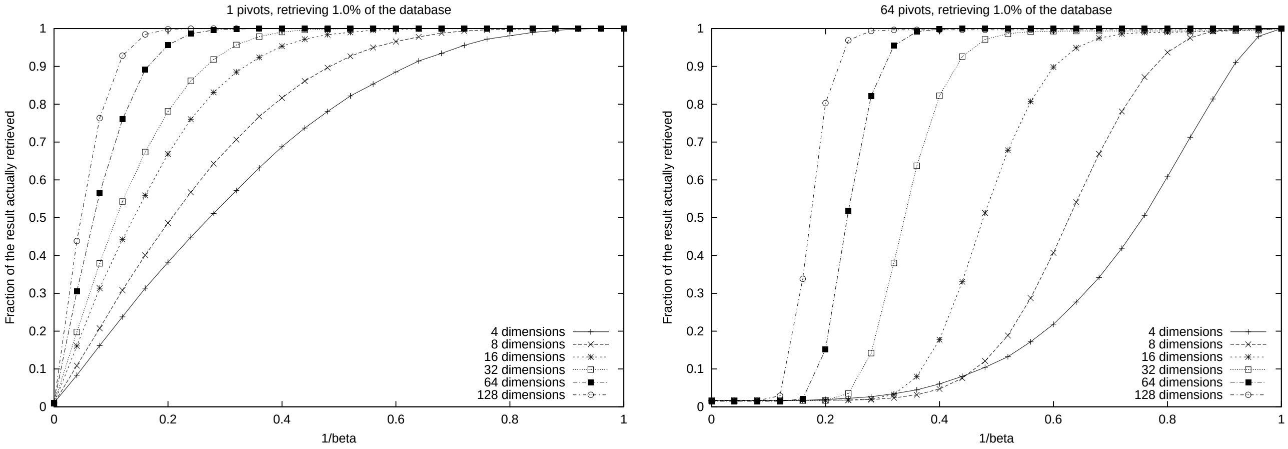

Figure 4: The probability of retrieving relevant elements as a function of $1/\beta$

The plots show clearly that the probability of missing an element for a fixed β increases as the number of pivots grows. This is a negative effect of increasing k which is not present on the exact algorithm, and which can make it preferably to use a smaller k. A second observation is that, with the same β, the probability of error is reduced as the dimension grows. Hence we are permitted to relax more the algorithm on higher dimensions. In the experiments that follow, we do not use β directly anymore, but plot the results as a function of the probability of retrieving a relevant element.

4 Efficiency

Let us now consider the expected complexity when we use $\beta \geq 1$. We will establish in fact a lower bound, i.e. we present an optimistic analysis. This is because Chebyschev inequality does not permit us bounding in the other direction. Still the analysis has interest since it can be compared against the lower bound obtained in [5, 7] in order to check the improvement obtained in the lower bound.

We discard an arbitrary point u with a pivot p_i whenever $|X-Y| > r/\beta$, where X and Y are as in the previous section but this time they are truly independent, as u is a random element (not one in the outcome of the query). Hence $Z = X - Y$ has mean zero and variance $2\sigma^2$ and we can use directly Chebyschev to lower bound the probability of discarding u with a random pivot p_i:

$$Pr(|X-Y| > r/\beta) \quad < \quad \frac{2\sigma^2}{(r/\beta)^2}$$

and the probability of discarding an element u with any of k random pivots is $1-(1-Pr(|X-Y| > r/\beta)^k)$. The average search cost is the internal evaluations k plus the external ones, which are on average n times the probability of not discarding a random element u. This gives a lower bound

$$Cost \quad \geq \quad k \quad + \quad n\left(1-\frac{2\sigma^2}{(r/\beta)^2}\right)^k$$

where, if we compare against the lower bounds obtained in [5, 7], we can see that the net effect is that of reducing the search radius. That is, our search cost is the same as if we searched with radius r/β (this reduction is used just to discard elements, of course when we compare q against the remaining candidates we use the original r). The search cost grows very fast as the search radius increases, so we expect a large improvement in the search time with a very moderate β, and hence with a moderate error probability. To see the effect in terms of the intrinsic dimension, we convert again $r = \mu - \sigma/\sqrt{f}$ to get

$$Cost \quad \geq \quad k \quad + \quad n\left(1-\frac{\beta^2}{(\sqrt{\rho}-1/\sqrt{2f})^2}\right)^k$$

Now we relate β with its maximum allowed value obtained in the previous section (Eq. 4). The result is

$$Cost \quad \geq \quad k \quad + \quad (1-\varepsilon)n\left(1-\frac{1}{(\sqrt{\rho}-1/\sqrt{2f})^2}\right)^k$$

which says that, if we obtain $(1-\varepsilon)$ of the result, then we basically pay $(1-\varepsilon)$ of the search cost.

Of course this last result is not good. This is not a consequence of the application of Chebyschev (which is a very loose bound), but of the simplified model we have used in Figure 3. Using A, B and C as defined there, the probability of not making a mistake with k pivots is $(A/B)^k$, and this has to be $1-\varepsilon$. The probability of not discarding an element when comparing it against the pivots is $(A/C)^k$ on our probabilistic method and $(B/C)^k$ with the classical algorithm, so our new search cost is $k+n(A/C)^k = k+n(A/B)^k(B/C)^k = k+n(1-\varepsilon)(B/C)^k$, which is in essence $1-\varepsilon$ times the cost of the classical algorithm.

As we show next, in practice the results are much better. The real reason for the pessimistic analytical results is the simplification of assuming that the histogram of $Z = X - Y$ when X and Y are distances from p to two very close elements q and u is the same as for two random X and Y chopping out the tails. In practice, the histogram of $X - Y$ is much more concentrated and hence the error probability is much lower. Despite this looseness, the analysis permits understanding the role played by the different variables.

Figure 5 shows the number of comparisons as a function of the fraction of relevant elements retrieved, for different combinations of numbers of pivots and search radii. We are using the same database as for the previous experiments.

As can be seen, we can retrieve even 90% or 95% of the relevant elements paying much less than the time necessary for the exact algorithm (which corresponds to $1/\beta = 1$ in the plots). In many cases there is a large difference in the cost to retrieve 99% and 100% of the set. These differences are most notorious when the number of pivots is not enough to get good results with the exact algorithm. In practice, we can obtain the same result as the exact algorithm with 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.

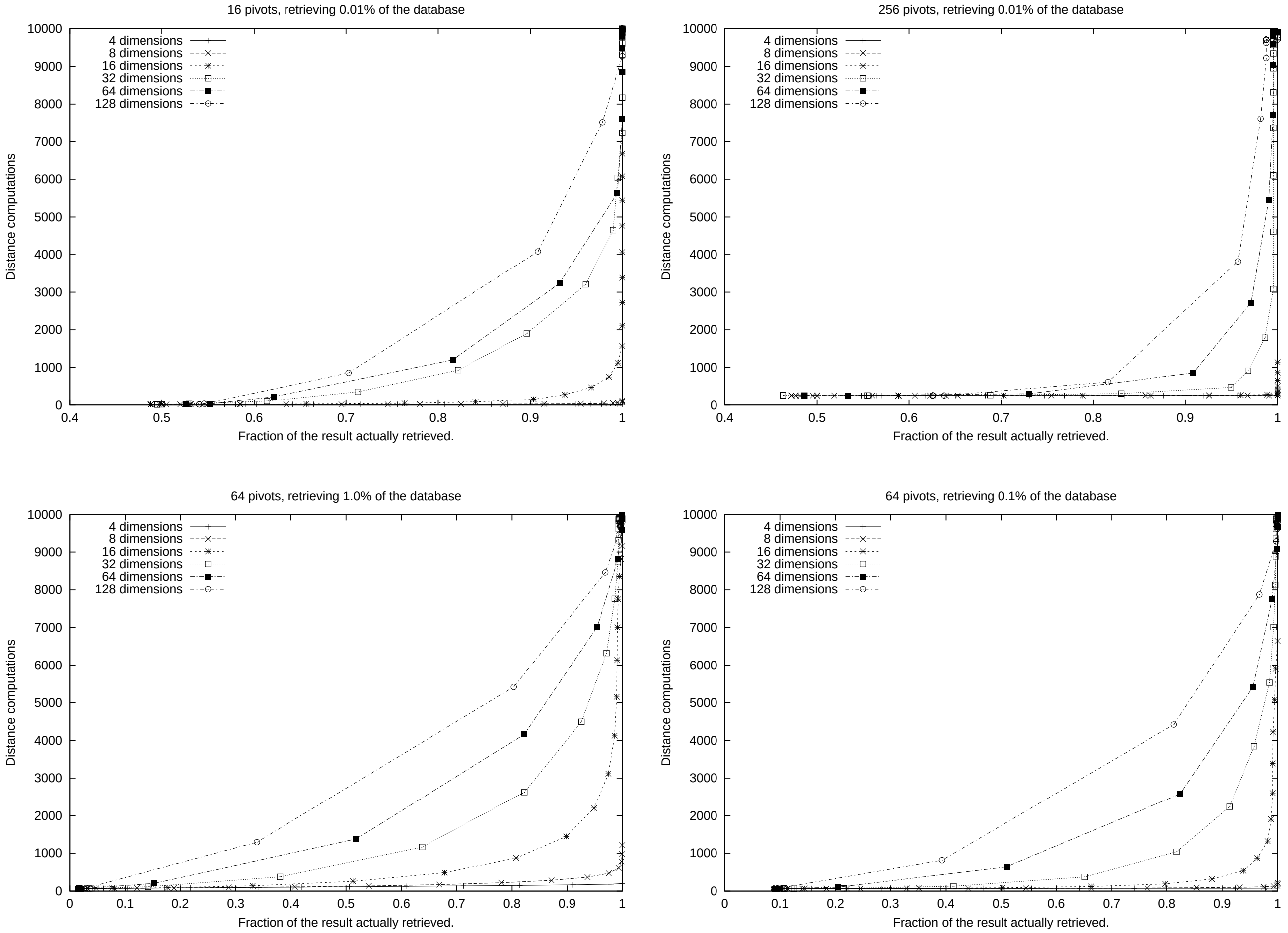

Figure 5: Number of distance evaluations as a function of the fraction of elements retrieved, for different dimensions.

Figure 6 shows the effect of different search radii (retrieving from 0.01an effect similar to that of increasing the dimension.

Finally, Figure 7 shows that, as we mentioned, the external complexity is not monotonously decreasing with the number of pivots k as for the exact algorithm. This is because, as k increases, the probability of missing a relevant answer grows, and therefore we need a larger β 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, as shown analytically, the scheme does also get worse as the dimension grows. However, it worsens much slower and obtain similar results as the exact algorithm with much less pivots. Even reaching the optimal number of pivots in medium dimensions, which is unthinkable in the exact algorithm, becomes feasible now.

5 Conclusions

We have presented a probabilistic algorithm to search in metric spaces. It is based on taking advantage of high dimensionalities by "stretching" the triangle inequality. The idea is general, but we have applied it to pivot based algorithms in this work. We have analytically found the improvement that can be expected as a function of the error probability permitted, the fraction of the set that is to be retrieved and the number of pivots used. The analysis shows that the net effect of the technique is to reduce the search radius, and that the reduction is larger when the search problem becomes harder (i.e. ρ or f grow). Finally, we have experimentally shown that even with very little stretching (and hence with very low error probability) we obtain dramatic improvements in the search time.

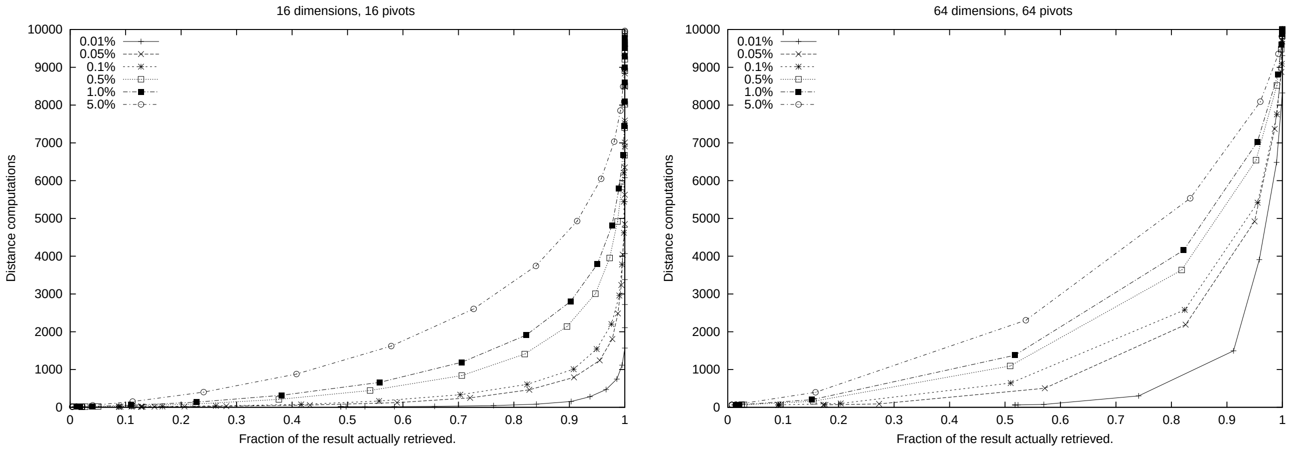

Figure 6: Number of distance evaluations as a function of the fraction of elements retrieved, for different radii.

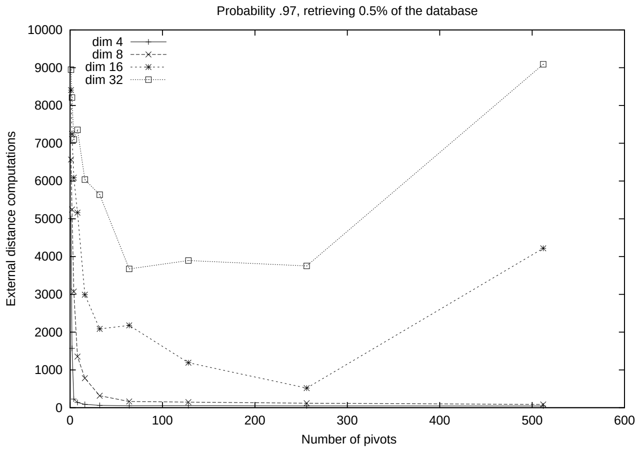

Figure 7: Number of external distance evaluations as a function of the number of pivots used, when retrieving a fixed fraction 0.97 of the relevant results with a search radius that should retrieve 0.5% of the dataset.

It is worth noting that β can be chosen at search time, so the indexing can be done beforehand and later we can choose the desired combination of speed and accurateness. Moreover, β represents a deterministic guarantee by itself: no element closer than r/β can be missed.

The technique can be successfully applied to other data structures, which opens a number of possibilities for future work. In general, all the algorithms can be modeled as performing some internal evaluations to discard elements with some rule related to the triangle inequality and later comparing the not discarded elements (external evaluations) [7]. If the internal evaluations do not depend on each other, then using this technique the internal complexity remains the same and the externals correspond to searching with radius r/β.

References

[1] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 573–583, 1994.

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

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

[4] E. Chávez, J. Marroquín, and G. Navarro. Overcoming the curse of dimensionality. In *Proc. European Workshop on Content-Based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999.

[5] E. Chávez and G. Navarro. Measuring the dimensionality of general metric spaces. Technical Report TR/DCC-00-1, Dept. of Computer Science, Univ. of Chile, 2000. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/metricmodel.ps.gz`.

[6] E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. Technical Report TR/DCC-00-2, Dept. of Computer Science, Univ. of Chile, 2000. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/probmetric.ps.gz`.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. Technical Report TR/DCC-99-3, Dept. of Computer Science, Univ. of Chile, 1999. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/survmetric.ps.gz`.

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

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

[10] 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.

[11] D. White and R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, Visual Computing Laboratory, University of California, La Jolla, CA, July 1996.

[12] Peter N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. Technical report, NEC Research Institute, Princeton, NJ, June 1999.