Measuring the Dimensionality of General Metric Spaces *

Edgar Chávez† Gonzalo Navarro‡

Abstract

Searching in metric spaces is the problem of, given a set of elements and a distance function defined among them, find all the elements close enough to a given query element. For efficiency, they try to minimize the number of evaluations of the distance function. This problem has a large number of applications, and a well-known particular case is that of vector spaces, where the objects are ℓ-dimensional coordinates.

The search problem is known to be difficult as the dimension ℓ grows. However, in practice the "intrinsic" dimension of a real-world vector space is less than ℓ because the elements are not uniformly distributed. The behavior of all the search algorithms is known to be related to the intrinsic dimension, but defining it is difficult. Moreover, a similar phenomenon is observed in general metric spaces, where there are no coordinates.

In this paper we introduce a new definition of intrinsic dimensionality that is simple and efficient to estimate and which is shown analytically and experimentally to capture the essential feature of metric spaces that determine the behavior of all the search algorithms. We prove analytically that our definition extends naturally that of vector spaces and that lower bounds on the behavior of a large class of proximity search algorithms can be stated as a function of our intrinsic dimensionality measure.

1 Introduction

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 (where a new element must be classified according to its closest existing element); image quantization and compression (where only some vectors can be represented and those that cannot must be coded as their closest representable point); text retrieval (where we look for words in a text database allowing a small number of errors, or we look for documents which are similar to a given query or document); computational biology (where we want to find a DNA or protein sequence in a database allowing some errors due to typical variations); function prediction (where we want to search the most similar behavior of a function in the past so as to predict its probable future behavior); etc.

The above scenarios require more general models and search algorithms than those classically used for simple data. A unifying concept is that of "similarity searching" or "proximity searching",

*This project has been partially supported by CYTED VII.13 AMYRI Project.

†Escuela de Ciencias Físico-Matemáticas, Universidad Michoacana. Edificio "B", Ciudad Universitaria, Morelia, Mich. México 58000. elchavez@zeus.ccu.umich.mx. Partially supported by CONACyT under grant R-28923A

‡Depto. de Ciencias de la Computación, Universidad de Chile. Blanco Encalada 2120, Santiago, Chile. gnavarro@dcc.uchile.cl. Partially supported by Fondecyt Grant 1-000929.

i.e. searching for database elements which are similar or close to a given query element. Similarity is modeled with a *distance function* that satisfies the triangular inequality, and the set of objects is called a *metric space*. Any query can be solved by comparing it against every element of the database. However, as the distance function is costly to compute in most applications, the general goal is to preprocess the set so as to minimize the number of distance evaluations at query time.

In some applications the metric space turns out to be of a particular type called *vector space*, where the elements consist of ℓ real-valued coordinates. A lot of work has been done on vector spaces by exploiting their geometric properties, but normally these cannot be extended to general metric spaces where the only available information is the distance among objects.

A well known phenomenon on vector spaces is that all the proximity searching algorithms degrade systematically as the dimension ℓ grows. If the vectors and queries are random and uniformly distributed, then the problem becomes intractable for about $\ell \geq 20$. This has been called the *curse of dimensionality*. In real applications, however, higher dimensions can be dealt with because the distribution is not uniform. For example, if in a 50-dimensional vector space all the points lie in a plane, smart search algorithms can be have like if searching on a two-dimensional space. The same happens when searching clustered data. The value ℓ is called the *representational dimension* of the space, while a more fuzzy concept of *intrinsic dimension* tries to capture the "real" dimension of the space.

The general goal for proximity search algorithms is to make them behave according to the intrinsic and not the representational dimension of the space, and a number of dimensionality reduction techniques have been proposed to achieve this. Despite these efforts, just a few attempts to define the intrinsic dimension of a vector space have been made.

On the other hand, general metric spaces present similar problems to proximity search algorithms. Some metric spaces are easily dealt with by all the algorithms while others are intractable. Despite the absence of coordinates in general metric spaces, many authors speak in terms of "intrinsic dimensionality" of metric spaces as a measure of the difficulty of searching in them. Several authors have pointed out at the histogram of distances as a tool to measure the dimensionality, noting that in vector spaces the histogram of distances shrinks and shifts to the right as ℓ grows. However, no clear quantitative measure has been proposed up to now for the intrinsic dimensionality, nor it has been established its relationship with the difficulty of the search problem.

In this paper we propose a definition of *intrinsic dimensionality* of a set of points in a metric space (which includes vector spaces) which is simply defined in terms of statistical properties of the histogram of distances and is cheap to estimate. We show analytically that a random and uniformly distributed vector space of dimension ℓ has intrinsic dimension $\Theta(\ell)$ according to our definition, and show experimentally that the constant is between 1.00 and 1.43.

We also prove lower bounds on the performance of large classes of proximity searching algorithms in terms of our definition of intrinsic dimensionality.

This paper is organized as follows. In Section 2 we explain the basic concepts of proximity searching in metric spaces. In Section 3 we briefly cover the existing data structures and algorithms for proximity searching in vector spaces and general metric spaces. In Section 4 we discuss alternative notions of intrinsic dimensionality. In Section 5 we define our quantitative measure of intrinsic dimension and evaluate its consistency analytically and experimentally. In Section 6 we prove lower bounds on the behavior of proximity searching algorithms as a function of the intrinsic

dimension. Finally, in Section 8 we give our conclusions.

2 Basic Concepts

We present now the formal tools necessary to understand the rest of the paper.

2.1 Metric Spaces

The set $\mathbb{X}$ will denote the universe of *valid* objects. A finite subset of it, $\mathbb{U}$, of size $n = |\mathbb{U}|$, is the set of objects where we search. $\mathbb{U}$ will be called the *dictionary*, *database* or simply our set of *objects* or *elements*. The function

$$d \ : \ \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}$$

will denote a measure of "distance" between objects (i.e. the smaller the distance, the closer or more similar are the objects). Distance functions have the following properties:

Positiveness: $\forall x, y \in \mathbb{X},\ d(x, y) \geq 0$

Symmetry: $\forall x, y \in \mathbb{X},\ d(x, y) = d(y, x)$

Reflexivity $\forall x \in \mathbb{X},\ d(x, x) = 0$

and in most cases

Strict positiveness: $\forall x, y \in \mathbb{X},\ x \neq y \ \Rightarrow \ d(x, y) > 0$

The similarity properties enumerated above only ensure a consistent definition of the function, and cannot be used to save comparisons in a proximity query. If d is indeed a *metric*, i.e. if it satisfies

Triangle inequality: $\forall x, y, z \in \mathbb{X},\ d(x, y) \ \leq \ d(x, z) + d(z, y)$

then the pair $(\mathbb{X}, d)$ is called a *metric space*.

If the distance does not satisfy the strict positiveness property then the space is called a *pseudo-metric space*. Although for simplicity we do not consider pseudo-metric spaces in this work, all the presented techniques are easily adapted to them by simply identifying all the objects at distance zero as a single object. This works because, if the triangle inequality holds, one can easily prove that $d(x, y) = 0 \Rightarrow \forall z, d(x, z) = d(y, z)$.

2.2 Proximity Queries

There are basically two types of queries of interest in metric spaces:

Range query: Retrieve all elements which are within distance r to q.
This is, retrieve $(q, r)_d \ = \ \{u \in \mathbb{U} \ / \ d(q, u) \leq r\}$.

Nearest neighbor query: Retrieve the k closest elements to q in $\mathbb{U}$.

This is, retrieve $nn_d(q,k) \subseteq \mathbb{U}$ such that $|nn_d(q,k)| = k$ and $\forall u \in nn_d(q,k), v \in \mathbb{U} - nn_d(q,k),\ d(q,u) \leq d(q,v)$.

The most basic type of query is the range query, which is a pair $(q,r)_d$ with q an element in $\mathbb{X}$ and r a real number indicating the *radius* (or tolerance) of the query. The set $\{u \in \mathbb{U},\ d(q,u) \leq r\}$ will be called the *outcome* of the range query. The left part of Figure 1 illustrates a query on a set of points, using $\mathbb{R}^2$ as the metric space for clarity.

In this work we concentrate on range queries for simplicity. Many of the results, however, can be applied to nearest neighbor searching as well, since the corresponding algorithms are normally built over those for range queries [15].

In most applications the distance $d()$ is very expensive to compute, and therefore the complexity of a search algorithm is measured in terms of number of evaluations of $d()$. It is clear that either type of query can be answered by examining the entire dictionary $\mathbb{U}$. In fact if we are not allowed to preprocess the data, i.e. to build an index data structure, then this exhaustive examination is the only way to proceed. An *indexing algorithm* is an off-line procedure to build beforehand a data structure (called *index*) designed to save distance computations when answering proximity queries later. This data structure can be expensive to build, but this will be amortized by saving distance evaluations over many queries to the database. The aim is therefore to design efficient indexing algorithms to reduce the number of distance evaluations. All these structures work on the basis of discarding elements using the triangular inequality (which is the only property that allows saving distance evaluations).

2.3 Vector Spaces

If the elements of the metric space $(\mathbb{X}, d)$ are indeed tuples of real numbers (actually tuples in any field) then the pair is called a *finite dimensional vector space,* or vector space for short.

An ℓ-dimensional vector space is a particular metric space where the objects are identified with ℓ real-valued coordinates $(x_1, ..., x_\ell)$. There are a number of options for the distance function to use, but the most widely used is the family of L_s distances, defined as

$$L_s((x_1, ..., x_\ell), (y_1, ..., y_\ell)) = \left(\sum_{i=1}^{\ell} |x_i - y_i|^s \right)^{1/s}$$

The right part of Figure 1 illustrates some of these distances. For instance, the L_1 distance accounts for the sum of the differences along the coordinates. It is also called "block" or "Manhattan" distance, since in two dimensions it corresponds to the distance to walk between two points in a city of rectangular blocks. The L_2 distance is better known as "Euclidean" distance, as it corresponds to our notion of spatial distance. The other most used member of the family is L_∞, which corresponds to taking the limit of the L_s formula when s goes to infinity. The result is that the distance between two points is the *maximum* difference along a coordinate:

$$L_\infty((x_1, ..., x_\ell), (y_1, ..., y_\ell)) = \max_{i=1}^{\ell} |x_i - y_i|$$

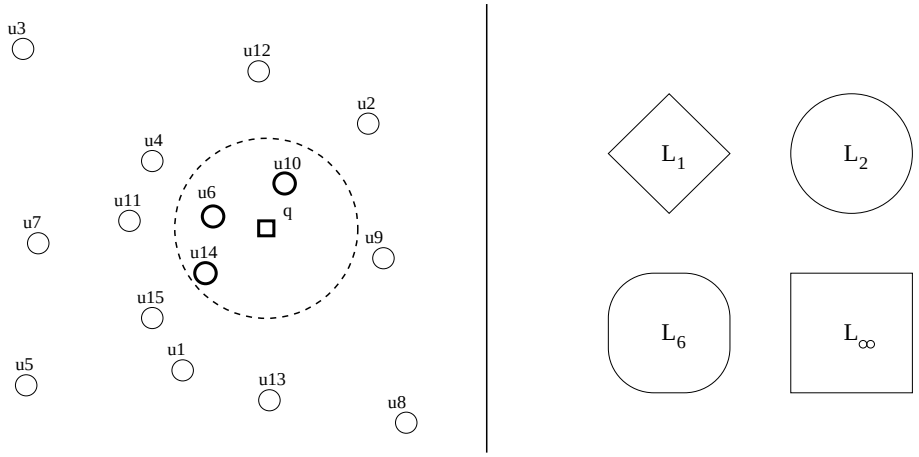


Figure 1: On the left, an example of a range query on a set of points. On the right, the set of points at the same distance to a center point, for different L_s distances.

3 Proximity Search Algorithms

Different data structures have been proposed to filter out elements based on the triangular inequality (see [15] for a complete survey). We first discuss briefly the case of vector spaces and then cover metric spaces, dividing the exposition according to the two main techniques used.

3.1 Vector Spaces

In many applications the metric space is indeed a vector space, i.e. the objects are ℓ-dimensional points and the similarity is interpreted geometrically. A vector space permits more freedom than a general metric space when designing search approaches, since it is possible to use geometric and coordinate information which is unavailable in a general metric space.

In this framework optimal algorithms (on the database size) exist in both the average and the worst case [7] for closest point search. Search structures for vector spaces are called *spatial access methods* (SAM). Among the most popular are *kd*-trees [5, 6], *R*-trees [25], quad-trees [35] and the more recent *X*-trees [8]. These techniques make extensive use of coordinate information to group and classify points in the space. For example *kd*-trees divide the space along different coordinates and *R*-trees groups points in hyper-rectangles. Unfortunately the existing techniques are very sensitive to the vector space dimension. Closest point search algorithms have an exponential dependency on the dimension of the space (this is called the *curse of dimensionality*).

Vector spaces may suffer from large differences between their *representational* dimension (ℓ) and their *intrinsic* dimension (i.e. the real number of dimensions in which the points can be embedded while keeping the distances among them). For example a plane embedded in a 50-dimensional space has intrinsic dimension 2 and representational dimension 50. This is in general the case of real applications, where the data is clustered, and it has lead to attempts to measure the intrinsic dimension such as the concept of "fractal dimension" [21]. Despite that no techniques can cope with intrinsic dimension higher than 20, much higher representational dimensions can be handled

by dimensionality reduction techniques [22, 18, 26].

Not all the applications can be modeled with vector spaces. For this reason several authors resort to general metric spaces, even knowing that the search problem is much more difficult. Of course it is also possible to treat a vector space as a general metric space, by using only the distances between points. One immediate advantage is that the intrinsic dimension of the space shows up, independent of any representational dimension (this requires extra care in vector spaces).

Specific techniques for vector spaces can be found in good surveys [35, 38, 24].

3.2 Pivot-based Algorithms

Pivot-based algorithms are built on a single general idea: select some elements from $\mathbb{U}$ (called *pivots*), and identify all the other elements with to their distances to (some of) the pivots. The methods differ in how they select the pivots, how much information they store about the distances among elements and pivots, etc.

Burkhard-Keller Trees (BKTs) [11] are designed for discrete distance functions: they select a pivot element p as the root of the tree, and put at child i the elements which are at distance i to the pivot. Each subtree is recursively built with the same technique until there are b elements or less, in which case the elements are simply stored in a "bucket" at the tree leaf. A range query q with tolerance radius r is searched by measuring $d(p,q)$, reporting p if appropriate, and entering only into subtrees numbered $d(p,q)-r$ to $d(p,q)+r$. The rest need not be considered because of the the triangle inequality. The buckets reached are exhaustively compared against q.

Fixed Queries Trees (FQTs) [3] are an evolution where the same pivot is used for all the nodes of the same level of the tree. In this case the pivot does not need to belong to the subtree. Many comparisons are saved in the backtracking process because only one different pivot per level exists. However, the tree is taller. A variant called Fixed Height FQT (FHQT) is also proposed where all the leaves are at the same depth h, regardless of the bucket size.

Vantage Point Trees (VPTs) [36, 39] are designed for continuous distance functions. The root has two equal-size subtrees that divide the elements in closer to and farther from the root. This can be extended to m-ary trees (MVPTs) [10, 9].

Finally, algorithms like AESA [37], LAESA [31, 30] and its variants [33, 13] and Fixed Queries Arrays (FQAs [14]) are based in a common idea: k pivots are selected and each object is mapped to k coordinates which are its distances to the pivots. Later, the query q is also mapped and if it differs from an object in more than r along some coordinate then the element is filtered out by the triangle inequality. That is, if for some pivot p_i and some element v of the set it holds $|d(q,p_i)-d(v,p_i)|>r$, then we know that $d(q,v)>r$ without need to evaluate $d(v,q)$. The elements that cannot be filtered out using this rule are directly compared.

An interesting feature of most of these algorithms is that they can reduce the number of distance evaluations by increasing the number of pivots. Define $D_k(x,y)=\max_{1\le j\le k}|d(x,p_j)-d(y,p_j)|$. Using the pivots $p_1,...,p_k$ is equivalent to discarding elements u such that $D_k(q,u)>r$. As more pivots are added we need to perform more distance evaluations (exactly k) to compute $D_k(q,*)$ (these are called *internal* evaluations), but on the other hand $D_k(q,*)$ increases its value and hence it has a higher chance of filtering out more elements (those comparisons against elements that cannot be filtered our are called *external*). It follows that there exists an optimum k. This optimum, however, cannot be normally reached because it is too high in terms of space requirements: kn

distances have to be precomputed and stored in order to use k pivots. Hence, in general these methods use as many pivots as they can, and they are normally well below their optimum.

3.3 Clustering Algorithms

Clustering algorithms try to divide the space in zones as compact as possible. They select a set of *centers*, which are elements from $\mathbb{U}$, and divide the space so that each center has its zone of influence. Each zone is normally divided recursively. The algorithms differ in how the centers are selected, how the zones are delimited, etc.

Generalized Hyperplane Trees (GHTs) [36] use two "centers" for each tree node and divide the space according to which of the two centers is closer to each object. This is like dividing the space with a hyperplane formed by the points at the same distance from both centers. At search time the query enters into the subtrees whose zone of influence has a nonempty intersection with the query ball.

Bisector Trees [27, 34] are similar but the zones are not defined according to which is the closest center but using the concept of "covering radius". The *covering radius* of a zone is the minimum radius of a sphere that is necessary to contain all the points in the zone, and the elements are inserted in the subtrees trying to minimize covering radii. Voronoi Trees (VTs) [19] are a modification that tries reduce the covering radii.

GHTs are generalized to an m-ary partition in the Geometric Near-neighbor Access Tree (GNATs) [10], which makes a Voronoi-like partition of the space [1] among the m pivots at each node of the tree. However, the GNAT uses also the covering radius criterion to prune the search even more.

The M-tree (MT) [16] also takes m elements and divides the space among its zones of influence, but it uses only the covering radius information to classify and search the elements. The MT is able of dynamic insertion and deletion of points and is optimized for secondary memory.

Spatial Approximation Trees (SATs) [32] are based on approaching the query spatially: the search starts at the root of the tree and moves to neighbors that are closer to the query. The ideal data structure to obtain this is a Voronoi graph, which in the paper is proven impossible to build on a general metric space. Therefore the SAT is a simplification which forces some backtracking in the tree.

4 Alternative Notions of Intrinsic Dimensionality

We cover in this section other existing attempts to define the intrinsic dimensionality. Unfortunately, they are not easy to apply to general metric spaces.

4.1 The Mapping Dimension

A natural definition of the intrinsic dimensionality of a metric space is: "the lowest k so that $\mathbb{U}$ can be mapped onto $\mathbb{R}^k$ and keep all the inter-object distances reasonably well". If we call Φ the mapping from $\mathbb{U}$ to $\mathbb{R}^k$, then the following definition of how "well" have the distances been

preserved has been proposed [29]

$$stress \;=\; \sqrt{\frac{\sum_{a,b\in\mathbb{U}}(D(\Phi(u),\Phi(v)) - d(u,v))^2}{\sum_{a,b\in\mathbb{U}} d(u,v)^2}}$$

where D is the distance in the target space. The better the distances are preserved, the smaller the *stress*.

All the existing work uses the L_2 (Euclidean) distance for the target space. If the original metric space is indeed a vector space of $\ell > k$ dimensions, then the optimal solution is given by the Karhunen-Loève (KL) transform [20, 23].

However, in the general case the KL transform cannot be applied. For general metric spaces there exist only heuristics that, given a k value, try to map the objects from $\mathbb{U}$ minimizing the distortion in distances [26]. We survey two techniques which aim at minimizing the *stress*.

A first technique is Multidimensional Scaling (MDS) [29, 18]. In its simplest version, the algorithm makes an initial assignment of objects to points (this can be even at random) and then tries to improve the assignment iteratively until a local minimum is found. At each iteration it examines each point, computes its distance to the other $n-1$ points, and relocates it so that the *stress* is minimized with respect to that point. MDS has the problem of requiring at least $O(n^2)$ time to obtain a reasonable guess (in fact it has only been used for applications where $n \leq 100$).

A faster technique is called Fastmap [22], which tries to minimize the *stress* as follows. The idea is to assume that $\mathbb{U}$ is indeed an ℓ-dimensional vector space. It first chooses two elements v_1, v_2 which are far apart from each other, and decides that the first coordinate of the target space will be the "line" joining v_1 and v_2. In that coordinate, v_1 will have value 0 and v_2 will have value 1. The rest of the points u are mapped to the line by considering a triangle v_1uv_2 and using the *cosine law*:

$$d(u,v_2)^2 \;=\; d(u,v_1)^2 + d(v_1,v_2)^2 - 2x_u d(v_1,v_2)$$

which is valid for Euclidean spaces, where x_u is the projection of u on the line v_1v_2. The formula permits obtaining x_u, which will be the coordinate of u along the selected axis.

In a second step, all the points have to be mapped onto the $(k-1)$-dimensional hyperplane which is perpendicular to the axis. In this projection, the original distances $d(u,v)$ are transformed using the equation

$$d'(u,v)^2 \;=\; d(u,v)^2 - (x_u - x_v)^2$$

The above process is repeated k times, and each point u is then mapped to the k-tuple of x_u values obtained for each of the k axis selected. A good feature of Fastmap is that it takes only $O(kn)$ time.

Finally, it is interesting to consider that a large subclass of the pivoting algorithms can be regarded as relying on a mapping of this kind. A widely used method is to select $\{p_1 \ldots p_k\} \subseteq \mathbb{U}$ and map each $u \in \mathbb{U}$ to $(\mathbb{R}^k, L_\infty)$ using $\Phi(u) = (d(u,p_1), \ldots, d(u,p_k))$. This is because

$$D_k(u,v) = \max_{1\leq i\leq k} |d(u,p_i) - d(v,p_i)| = L_\infty(\Phi(u),\Phi(v))$$

can be used to discard potential candidates (recall that $d(u,v) \geq D_k(u,v)$).

Despite that all these methods take k as a parameter and are not designed to define the intrinsic dimension, one can state that the intrinsic dimension is the minimum k which obtains low enough *stress*. It is clear that the necessary k to minimize the *stress* can be reduced by an intelligent selection of pivots, but little is known about which are good selection policies [15, 28]. Moreover, the resulting dimension would depend on the *stress* permitted, which is a disadvantage.

4.2 The Fractal Dimension

This concept is proposed in [21] to find the intrinsic dimension of non-uniformly distributed data in vector spaces. For a finite set of points $\mathbb{U}$, the idea is to split the space with an ℓ-dimensional grid of blocks of width r. Let $N(r)$ be the number of such blocks that contain some point of $\mathbb{U}$. Then the fractal dimension is defined as

$$-\frac{\delta \log(N(r))}{\delta \log(r)}$$

which should stabilize for small enough r. In practical terms this can be computed as the slope of a plot obtained by sampling.

The authors show that, for instance, a straight line from $(0,0)$ to $(1,1)$ will touch i blocks if the cell is partitioned in $i \times i$ squares, and therefore $N(r) = 1/r$ and hence the fractal dimension will be 1. A solid disk of radius $1/2$ will touch a fixed proportion of the blocks, tending to $\pi/4$ as r tends to zero. Therefore $N(r) = \pi/4(1/r)^2$ and $\log N(r) = \log(\pi/4) - 2\log r$, which gives a fractal dimension of 2. Clustered 2-dimensional data is shown to have fractal dimensions between 1 and 2.

This is designed for vector spaces, and it seems difficult to extend to general metric spaces. We could try to generalize the technique by replacing the grid by the random choice of many balls centered around selected elements of $\mathbb{U}$. That is, we make m attempts of choosing $x \in \mathbb{X}$ randomly and measure $N(r)$ as the number of balls that contain some element of $\mathbb{U}$. This, however, would produce a *positive* derivative of $\log N(r)$ with respect to r. A key factor that makes it negative on vector spaces is that the grid covers all the space, and therefore the number of cells increase as r is reduced. The amount of the increase is exponential on the representational dimension, and this is a key fact for the result. It seems not possible to obtain something similar on a general metric space.

5 A New Definition of Intrinsic Dimension

The previous definitions of intrinsic dimension try to capture geometrical properties of the space, they are complex and expensive to evaluate and some are difficult to extend to a general metric space. Our goal is to have a simple and easy to compute measure of intrinsic dimension, which additionaly explains the curse of dimensionality. So we start by the other end: we try to capture the essence of the difficulty of searching in a metric space and later show that this matches with the classical definition of dimension.

Many authors [10, 12, 17] have proposed to use the histogram of distances to characterize the difficulty of searching in an arbitrary metric space, but no quantitative definition has been attempted. We present in this section a quantitative measure in this line and study its suitability.

Let us start with a well-known example. Consider a distance such that $d(x,x) = 0$ and $d(x,y) = 1$ for all $x \neq y$. Under this distance (in fact an equality test), we do not obtain any information from a comparison except that the element considered is or is not our query. It is clear that it is not possible to avoid a sequential search in this case, no matter how smart is our indexing technique.

Let us consider the *histogram* of distances between points in the metric space $\mathbb{X}$. This can be approximated by using the dictionary $\mathbb{U}$ as a random sample of $\mathbb{X}$. The idea is that, as the space has higher intrinsic dimension, the mean μ of the histogram grows and/or its variance σ^2 is reduced (at least this is the case on random vector spaces). Our previous example is an extreme case.

Figure 2 gives an intuitive explanation of why the search problem is harder when the histogram is concentrated. If we consider a random query q and an indexing scheme based on random pivots, then the possible distances between q and a pivot p are distributed according to the histogram of the figure. The elimination rule says that we can discard any point u such that $d(p,u) \notin [d(p,q) - r, d(p,q) + r]$. The grayed areas in the figure show the points that we *cannot* discard. As the histogram is more and more concentrated around its mean, less and less points can be discarded using the information given by $d(p,q)$. Moreover, in many cases the search radius r must grow as the mean distance μ grows, which makes the problem even harder.

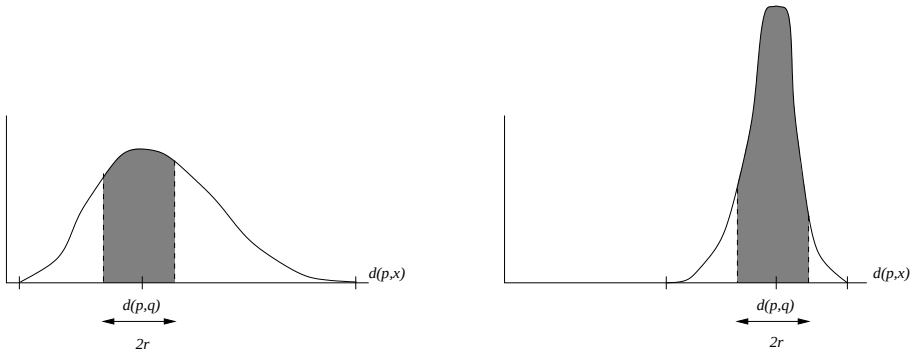


Figure 2: A low-dimensional (left) and high-dimensional (right) histogram of distances, showing that on high dimensions virtually all the elements become candidates for exhaustive evaluation.

This phenomenon is independent on the nature of the metric space (vectorial or not, in particular) and gives us a way to quantify how hard is to search on an arbitrary metric space.

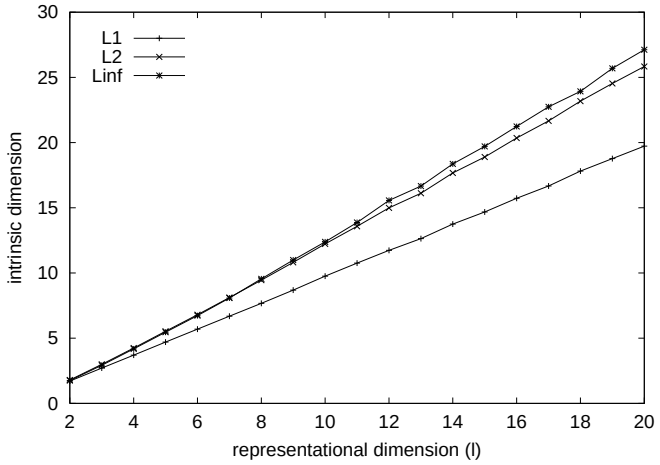
Definition. *The* intrinsic dimensionality *of a metric space is defined as* $\rho = \frac{\mu^2}{2\sigma^2}$, *where* μ *and* σ^2 *are the mean and variance of its histogram of distances.*

The technical convenience of the exact definition is made clear shortly. Observe that the intrinsic dimensionality grows with the mean and with the inverse of the variance of the histogram. Moreover, measuring this intrinsic dimension on an arbitrary and unknown metric space can be accomplished by simple statistical means via a reasonable number of distance evaluations among random points of the set. This is much simpler and cheaper than all previous approaches.

Let us check our definition on vector spaces. As shown in [40], a uniformly distributed ℓ-dimensional vector space under the L_s distance has mean $\Theta(\ell^{1/s})$ and standard deviation $\Theta(\ell^{1/s-1/2})$. Therefore its intrinsic dimensionality is $\Theta(\ell)$ (although the constant is not necessarily 1). So the

intuitive concept of dimensionality in vector spaces matches our general concept of intrinsic dimensionality.

Figure 3 shows an experimental verification of the above analytical result. We have generated random uniformly distributed points in the real ℓ-dimensional vector space $[0,1)^\ell$ for ℓ between 2 and 20, using three different distances L_1, L_2 and L_∞. We have selected one million pairs of points for each combination of dimension ℓ and distance L_s and have computed the intrinsic dimension of the resulting histogram of distances (by computing μ and σ using the classical statistical formulas). As can be seen, the intrinsic dimension grows linearly with the representational dimension when the points are chosen at random. We have included a table with the least squares estimations, which shows that a vector space of dimension ℓ has intrinsic dimension from $1.00 \times \ell$ to $1.43 \times \ell$, depending on the L_s distance used (any L_s with $s \geq 3$ generates lines between those of L_2 and L_∞).



Distance	Intrinsic Dimension	Percentual Error
L_1	$-0.314 + 1.003\ell$	0.34%
L_2	$-1.182 + 1.346\ell$	1.66%
L_∞	$-1.631 + 1.425\ell$	3.27%

Figure 3: Representational (ℓ) versus intrinsic dimension for random uniformly distributed vectors.

6 Lower Bounds in Terms of the Intrinsic Dimension

The main goal of the definition of an intrinsic dimensionality measure is that the search cost should increase as the dimension increases. Therefore an essential part of the test of the suitability of our new measure is to establish the relationship between intrinsic dimension and complexity of the search task. We do that analytically in this section and experimentally in the next one.

6.1 A Lower Bound for Pivoting Algorithms

Our main result in this section relates the intrinsic dimensionality with the difficulty of searching with a given search radius r using a pivoting equivalence relation that chooses the pivots at random. As we show next, the difficulty of the problem is related to r and the intrinsic dimensionality ρ.

We are considering independent identically distributed (i.i.d.) random variables for the distribution of distances between points. Although not accurate, this simplification is optimistic and

hence can be used to lower bound the performance of the indexing algorithms. We come back to this shortly.

Let $(q,r)_d$ be a range query over a metric space indexed by means of k random pivots, and let u be an element of $\mathbb{U}$. The probability that u cannot be excluded from direct verification after considering the k pivots is exactly

$$Pr(|d(q,p_1) - d(u,p_1)| \leq r, \ldots, |d(q,p_k) - d(u,p_k)| \leq r) \tag{1}$$

Since all the pivots are assumed to be random and their distance distributions i.i.d. random variables, this expression is the product of probabilities

$$Pr(|d(q,p_1) - d(u,p_1)| \leq r) \ \times \ \ldots \ \times \ Pr(|d(q,p_k) - d(u,p_k)| \leq r) \tag{2}$$

which for the same reason can be simplified to

$$Pr(\text{not discarding } u) \quad = \quad Pr(|d(q,p) - d(u,p)| \leq r)^k \tag{3}$$

for a random pivot p.

If X and Y are two i.i.d. random variables with mean μ and variance σ^2, then the mean of $X-Y$ is 0 and its variance is $2\sigma^2$. Using Chebyschev's inequality[1] we have that $Pr(|X-Y| > \varepsilon) < 2\sigma^2/\varepsilon^2$. Therefore,

$$Pr(|d(q,p) - d(u,p)| \leq r) \quad \geq \quad 1 - \frac{2\sigma^2}{r^2}$$

where σ^2 is precisely the variance of the distance distribution in our metric space. The argument that follows is valid for $2\sigma^2/r^2 < 1$, or $r > \sqrt{2}\sigma$ (large enough radii), otherwise the lower bound is zero. Then, we have

$$Pr(\text{not discarding } u) \quad \geq \quad \left(1 - \frac{2\sigma^2}{r^2}\right)^k$$

We have now that the total search cost is the number of internal distance evaluations (k) plus the external evaluations (those to check the remaining candidates), whose number is on average $n \times Pr(\text{not discarding } u)$. Therefore

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

is a *lower bound* to the average search cost by using pivots. Optimizing we obtain that the best k is

$$k^* \quad = \quad \frac{\ln n \ + \ \ln\ln(1/t)}{\ln(1/t)}$$

where $t = 1 - 2\sigma^2/r^2$. Using this optimal k^*, we obtain an absolute (i.e. independent on k) lower bound for the average cost of any random pivot-based algorithm:

$$Cost \quad \geq \quad \frac{\ln n \ + \ \ln\ln(1/t) \ + \ 1}{\ln(1/t)} \quad \geq \quad \frac{\ln n}{\ln(1/t)} \quad \geq \quad \frac{r^2}{2\sigma^2} \ \ln n$$

[1] For an arbitrary distribution Z with mean μ_z and variance σ_z^2, $Pr(|Z - \mu_z| > \varepsilon) < \sigma_z^2/\varepsilon^2$.

which shows that the cost depends strongly on σ/r. As r increases t tends to 1 and the scheme requires more and more pivots and it is anyway more and more costly.

A nice way to represent this result is to assume that we are interested in retrieving at most a fixed fraction f of the elements, in which case r can be written as $r = \mu - \sigma/\sqrt{f}$ by Chebyschev's inequality. In this case the lower bound becomes

$$\frac{r^2}{2\sigma^2} \ln n \;=\; \frac{(\mu - \sigma/\sqrt{f})^2}{2\sigma^2} \ln n \;=\; \left(\sqrt{\rho} - \frac{1}{\sqrt{2f}}\right)^2 \ln n$$

which is valid for $f \geq 1/(2\rho)$. We have just proved

Theorem 1 *Any pivot based algorithm using random pivots has a lower bound* $(\sqrt{\rho} - 1/\sqrt{2f})^2 \ln n$ *in the average number of distance evaluations performed for a random range query retrieving at most a fraction* f *of the set, where* ρ *is the intrinsic dimension of the metric space.*

This result matches that of [2, 4] on FHQTs, about obtaining $\Theta(\log n)$ search time using $\Theta(\log n)$ pivots, but here we are more interested in the "constant" term, which depends on the dimension.

The theorem shows clearly that the parameters governing the performance of range searching algorithms are ρ and f. As ρ grows and f stays fixed, this tends to $\rho \ln n$.

We have considered i.i.d. random variables for each pivot and the query. This is a reasonable approximation, as we do not expect much difference between the "view points" from the general distribution of distances to the individual distributions, a subject discussed in depth in [17]. The expression given in Eq. (3) cannot be obtained without this simplification.

A stronger assumption comes from considering all the variables as independent. This is an optimistic consideration equivalent to assuming that in order to discard each element u of the set we take k new pivots at random. The real algorithm fixes k random pivots and uses them to try to discard all the elements u of the set. The latter alternative can suffer from dependencies from a point u to another, which cannot happen in the former case (for example, if u is close to a pivot p and u' is close to u then the distance from u' to p carries less information). Since the assumption is optimistic, using it to reduce the joint distribution in Eq. (1) to the expression given in Eq. (2) keeps the lower bound valid.

Figure 4 shows an experiment on the search cost in $(\mathbb{R}^\ell, L_2)$ using different number of pivots k and dimensions ℓ. The $n = 100,000$ elements are generated at random and the pivots are randomly chosen from the set. We average over 1,000 random queries whose radius is set to retrieve 10 elements of the set. We count the number of distance evaluations. The left plot shows the existence of an optimum $k^* = 110$ in 8 dimensions, while the right plot shows the predicted $O(n(1-1/\Theta(\ell))^k)$ behavior for fixed k.

Figure 5 shows the growth of the optimum k^* and of the search cost using the optimal k^* as the dimension grows. Note that despite that our lower bound is linear on ρ, in practice the search cost grows faster.

6.2 The Target Space

It is interesting to study which is the behavior of the mapped space after selecting k pivots. Let us start with the mean of the distance D_k in the target space. As already seen, $D_k(\Phi(u), \Phi(v))$ is

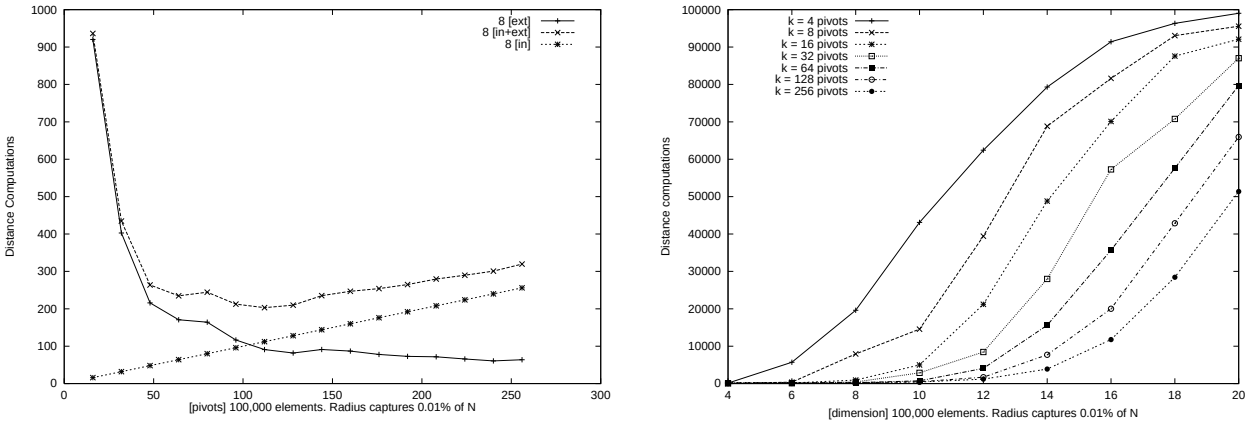


Figure 4: On the left, internal, external and overall distance evaluations in 8 dimensions, using different number of pivots k. On the right, overall distance evaluations as the dimension grows for fixed k.

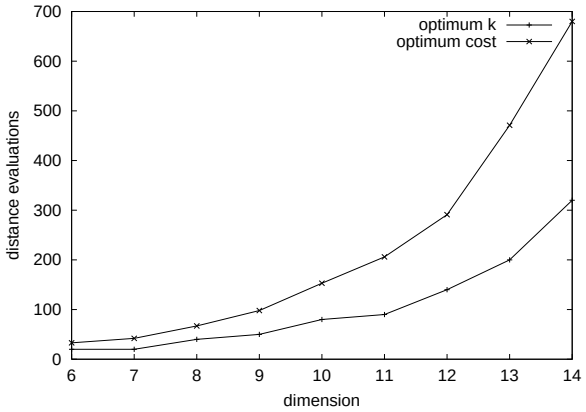


Figure 5: Optimum k^* and cost using that k^* as the dimension grows.

the maximum of k random variables $|d(u,p_i) - d(v,p_i)|$, and therefore

$$Pr(D_k(\Phi(u),\Phi(v)) \leq \varepsilon) \quad = \quad Pr(|d(u,p) - d(v,p)| \leq \varepsilon)^k \quad \geq \quad \left(1 - \frac{2\sigma^2}{\varepsilon^2}\right)^k$$

where the last inequality comes from applying Chebyschev as in the previous section. Calling Z the random variable associated to D_k, we have that its cummulative probability function is

$$F_z(x) \quad = \quad Pr(D_k(\Phi(u),\Phi(v)) \leq x) \quad \geq \quad \left(1 - \frac{2\sigma^2}{x^2}\right)^k$$

if $x \geq \sqrt{2}\sigma$. $F - z(x)$ can be optimistically assumed to be zero otherwise ("optimistically" means that we will obtain an upper bound on $E(Z)$). To obtain the density function $f_z(x)$ we derive the

cumulative distribution and get

$$f_z(x) \;=\; k(1-2\sigma^2/x^2)^{k-1}4\sigma^2/x^3$$

if $x \geq \sqrt{2}\sigma$ and zero otherwise. Now to obtain the mean we compute

$$E(Z) \;\leq\; \int_{\sqrt{2}\sigma}^{\infty} x\, f_z(x)\, dx \;=\; 4k\sigma^2 \int_{\sqrt{2}\sigma}^{\infty} (1-2\sigma^2/x^2)^{k-1}/x^2\, dx$$

Now doing the change $y = 2\sigma^2/x^2$ (and hence $dy = -4\sigma^2/x^3\, dx$) we have

$$E(Z) \;\leq\; \sqrt{2}k\sigma \int_0^1 (1-y)^{k-1} y^{-1/2}\, dy \;=\; \sqrt{2}k\sigma \binom{k+\frac{1}{2}}{\frac{1}{2}}$$

This is the exact solution (recall that $\binom{k+1/2}{1/2} = (k+1/2)(k-1/2)(k-3/2)\cdots(5/2)(3/2)$). For large k this converges to

$$E(Z) \;\leq\; \sigma\sqrt{2\pi k}$$

This means that the mean value of D_k is independent on the mean μ of the original metric space. Rather, it is proportional to the standard deviation (this is reasonable because it is a maximum of differences between distances). On the other hand, this mean grows with the square root of the number of pivots. Recall that, since this comes from using Chebyschev's inequality, the result is just an upper bound on the mean of D_k.

It is clear that D_k, as a lower bound for d, should be as close to d as possible in order to filter out most of the irrelevant elements. Therefore, we could like that both means be equal. Solving $\mu = \sigma\sqrt{2\pi k}$ yields $k = \rho/\pi$, which means that we must have at least a number of pivots proportional to the intrinsic dimensionality of the space (as shown in Theorem 1, the optimum is indeed larger). This is an optimistic bound and in practice k has to be much larger.

We have tried to obtain an upper bound on the variance of D_k, but the integral $\int x^2 f_z(x)\, dx$ does not converge. This does not mean that D_k does not have variance, because ours is just an upper bound.

Figure 6 shows an experimental result related to this analysis. As the dimension grows, the histogram of L_2 moves to the right ($\mu = \Theta(\sqrt{\ell})$) and its variance σ^2 remains constant. Yet the pivot distance D_k (in the projected space $(\mathbb{R}^k, L_\infty)$) remains about the same for fixed k. Increasing k from 32 to 512 moves the histogram slightly to the right. This shift is effective in low dimensional metric spaces, but in high dimensions both histograms are still far away. The plots of these two histograms can measure how good are the pivoting algorithms. As the overlap increases the search algorithms become more effective.

This also shows that we could search in the mapped space with radius

$$r' \;=\; \frac{r\sqrt{2}k\sigma \binom{k+1/2}{1/2}}{\mu} \;=\; \frac{rk\binom{k+1/2}{1/2}}{\sqrt{\rho}} \;\approx\; \frac{\sqrt{\pi k}}{\sqrt{\rho}}\, r$$

and still get most of the results. This opens the door to probabilistic algorithms which, with the penalty of a small error probability, are much faster than the exact versions.

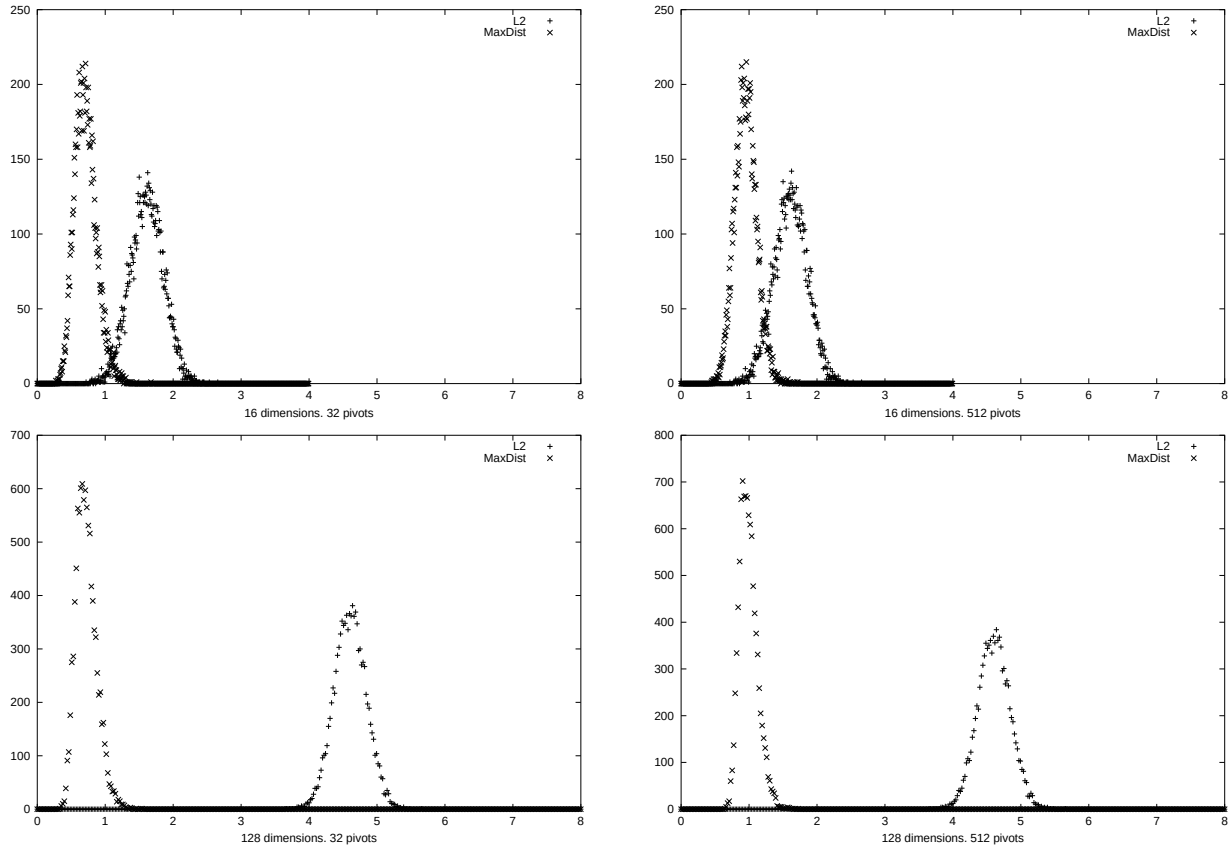


Figure 6: Histograms comparing the L_2 distance in different ℓ-dimensional Euclidean spaces and the pivot distance (MaxDist) obtained using different numbers k of pivots. In the top row $\ell = 16$ and in the bottom row $\ell = 128$. On the left $k = 32$ and on the right $k = 512$.

6.3 A Lower Bound for Clustering Algorithms

We now try to obtain lower bounds to the search cost of clustering algorithms. The result is surprisingly similar to that of pivoting algorithms. One lower bound considers only the hyperplane criterion to delimit Voronoi regions, and the other considers covering radii. Both results are similar. We assume just the same facts about the distribution of distances as in the previous section: all of them are i.i.d. random variables.

Let $(q, r)_d$ be a range query over a metric space indexed by means of m random centers $\{c_1 \ldots c_m\}$. The m distances $d(q, c_i)$ can be considered random variables $X_1 \ldots X_m$ whose distribution is that of the histogram of the metric space. The distribution of the distance from q to its closest center c is that of $Y = \min\{X_1 \ldots X_m\}$. The hyperplane criterion specifies that a class $[c_i]$ *cannot* be excluded if $d(q, c) + r \geq d(q, c_i) - r$. The probability that this happens is $Pr(Y \geq X_i - 2r)$. But since Y is the minimum over m variables with the same distribution, the probability is $Pr(Z \geq X - 2r)^m$, where X and Z are two random variables distributed according to the histogram. Using Chebyschev's inequality and noticing that if $Z < X - 2r$ then X or Z are

at distance at least r from their mean, we can say that

$$Pr(\text{not discarding } [c_i]) \quad = \quad Pr(Z \geq X - 2r)^m \quad \geq \quad \left(1 - \frac{\sigma^2}{r^2}\right)^m$$

On average each class has n/m elements, so that the external complexity is $n \times Pr($ not discarding $[c_i])$. The internal cost to find the intersected classes deserves some discussion. In all the hierarchical schemes that exist, we consider that the real partition is that induced by the leaves of the trees, i.e. the most refined ones. We see all the rest of the hierarchy as a mechanism to reduce the internal complexity of finding the small classes (hence the m we use here is not, say, the m of GNATs, but the total number of final classes). It is difficult to determine this internal complexity (an upper bound is m), so we call it $\mathcal{C}_I(m)$, knowing that it is between $\Omega(\log m)$ and $O(m)$. Then a lower bound to the search complexity is

$$Cost \quad \geq \quad \mathcal{C}_I(m) \; + \; n \; \left(1 - \frac{\sigma^2}{r^2}\right)^m$$

which indeed is very similar to the lower bound on pivot based algorithms. Optimizing on m yields

$$m^* \quad = \quad \frac{\ln n \; + \; \ln \ln(1/t') - \ln \mathcal{C}'_I(m^*)}{\ln(1/t')}$$

where $t' = 1 - \sigma^2/r^2$. Using the optimal m^* the search cost is lower bounded by

$$Cost \quad = \quad \Omega\left(\mathcal{C}_I(\log_{1/t'} n)\right) \quad = \quad \Omega\left(\mathcal{C}_I\left(\frac{r^2}{\sigma^2} \; \ln n\right)\right)$$

which also shows an increase in the cost as the dimensionality grows. As before we can write $r = \mu - \sigma/\sqrt{f}$. We have just proved

Theorem 2 *Any Voronoi based algorithm using on random centers has a lower bound* $\mathcal{C}_I(2(\sqrt{\rho} - 1/\sqrt{2f})^2)$ *in the average number of distance evaluations performed for a random range query retrieving a fraction of at most* f *of the database, where* ρ *is the intrinsic dimension of the space and* $\mathcal{C}_I()$ *is the internal complexity to find the relevant classes, satisfying* $\Omega(\log m) = \mathcal{C}_I(m) = O(m)$.

This result is weaker than Theorem 1 because of our inability to give a good lower bound on $\mathcal{C}_I$, so we cannot ensure more than a logarithmic increase with respect to ρ. However, even assuming $\mathcal{C}_I(m) = \Theta(m)$ (i.e. exhaustive search of the classes), when the theorem becomes very similar to Theorem 1, there is an important reason that explains why the Voronoi based algorithms can in practice be better than pivot based ones. We *can* in this case achieve the optimal number of centers m^*, which is impossible in practice for pivot-based algorithms. The reason is that it is much more economical to represent the Voronoi partition using m centers than the pivot partition using k pivots.

Let us now consider the other property, namely the covering radius. We show a similar lower bound. Assume that the set has been partitioned into $m = n/M$ clusters of M elements each, where we optimistically assume that the cluster of each center contains its M closest elements. Now, a

query q will need to be compared against the elements of the cluster if the query ball intersects it. The discarding rule of the covering radius says that if we call $cr(c_i)$ the distance between a center c_i and the farthest element of its ball, then we cannot discard $[c_i]$ if $d(c_i,q) - r \leq cr(c_i)$.

Since we have assumed that the clusters are the best possible, i.e. they contain the M closest neighbors of c_i, the probability of not discarding $[c_i]$ is that of $d(c_i,q) - r$ being among the M smallest values in the set $\{d(u,c_i),\ u \in \mathbb{U}\}$. Let $Y_u = d(u,c_i)$ be random variables that are distributed according to the histogram, and let $X = d(q,c_i)$, which has the same distribution. Now, let us define $p_> = Pr(X - r > Y_u)$ and $p_\leq = Pr(X - r \leq Y_u)$. The probability of $X - r$ being exactly the $(i+1)$-th value in the set of Y_u's is

$$\binom{n}{i} p_>^i p_\leq^{n-i}$$

and therefore the probability that $X - r$ is among the first M positions is

$$\sum_{i=0}^{M-1} \binom{n}{i} p_>^i p_\leq^{n-i} \quad \geq \quad \sum_{i=0}^{M-1} \binom{n}{i} p_\leq^{n-i} \quad \geq \quad \binom{n}{M-1} p_\leq^{n-M+1} \quad \geq \quad \binom{n}{M-1} \left(1 - \frac{2\sigma^2}{r^2}\right)^{n-M+1}$$

where we have used Chebyschev at the end. Each cluster visited costs M comparisons and this can happen for each of the $n/M = m$ clusters. On the other hand, we have as before the $\mathcal{C}_I(m)$ internal complexity to find the relevant clusters. Hence the cost is

$$Cost \quad \geq \quad \mathcal{C}_I(n/M) \ + \ n \binom{n}{M-1} \left(1 - \frac{2\sigma^2}{r^2}\right)^{n-M+1}$$

which we can derive to obtain the optimum M:

$$M^* \quad = \quad \frac{n}{1 + \frac{r^2}{2\sigma^2} \ln n}$$

and hence the optimal cost is

$$Cost \quad \geq \quad \Omega\left(\mathcal{C}_I\left(1 + \frac{r^2}{2\sigma^2} \ \ln n\right)\right)$$

which is basically the same as for the case of Voronoi partitions. This leads to

Theorem 3 *Any covering radius based algorithm using random centers has a lower bound* $\mathcal{C}_I(1 + (\sqrt{\rho} - 1/\sqrt{2f})^2)$ *in the average number of distance evaluations performed for a random range query retrieving a fraction of at most* f *of the database, where* ρ *is the intrinsic dimension of the space and* $\mathcal{C}_I()$ *is the internal complexity to find the relevant classes, satisfying* $\Omega(\log m) = \mathcal{C}_I(m) = O(m)$.

Figure 7 shows an experiment on the same dataset, where we have used different m values and a hierarchical clustering partitioning based on them. We have used the hyperplane and the covering radius criteria to prune the search. As can be seen, the dependency on the dimension of the space is not so sharp as for pivot based algorithms, and is closer to a dependency of the form $\Theta(\ell)$.

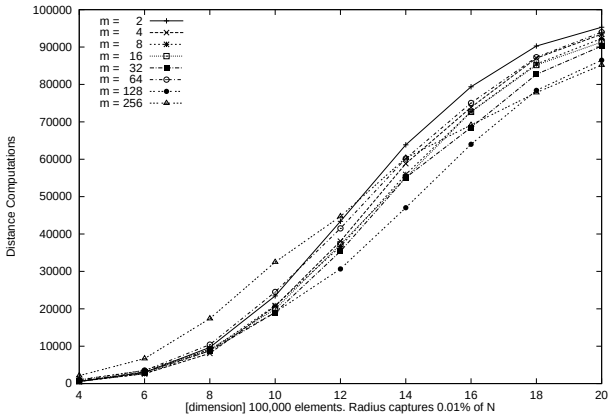


Figure 7: Overall distance evaluations using a hierarchical clustering partitioning with different arities.

The general conclusion is that, even if the lower bounds using pivot based or clustering based algorithms look similar, the first ones need much more space to store the index resulting from k pivots than the last ones using the same number of pivots. Hence, the latter can reallistically use the optimal number of centers, while the former cannot. If pivot based algorithms are given all the necessary memory, then using the optimal number of pivots they can improve over clustering based algorithms, because t is better than t', but this is more and more difficult as the dimension grows.

Figure 8 compares both types of partitioning. As can be seen, the pivoting algorithm improves over the clustering algorithms if we give it enough pivots. However, "enough" is a number that increases with the dimension and with the fraction retrieved (i.e. ρ and f). For ρ and f large enough, the required number of pivots will be unacceptably high in terms of memory requirements.

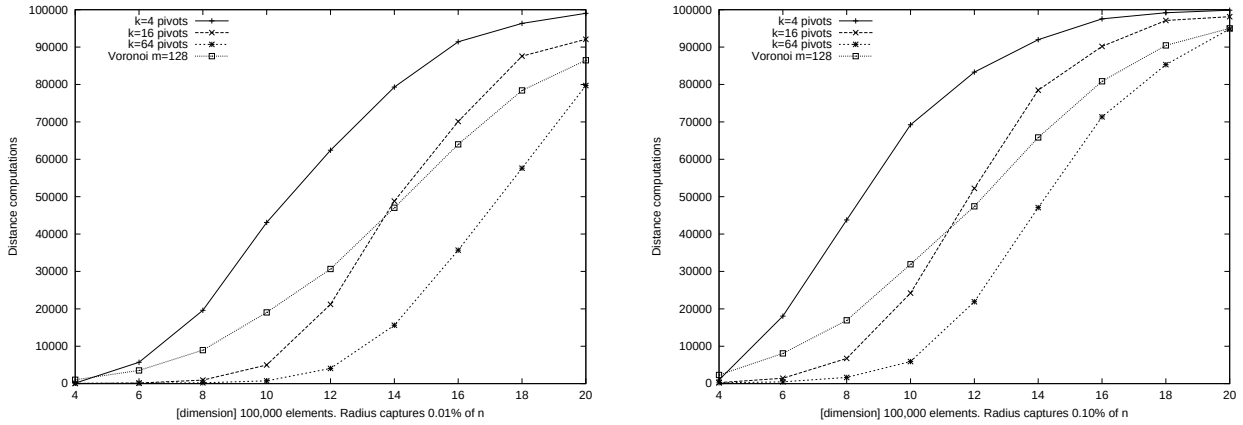


Figure 8: Distance evaluations for increasing dimension. We compare the clustering algorithm of Figure 7 using 128 centers per level against the filtration using k pivots for $k = 4, 16, 64$. On the left plot the search radius captures 0.01% of the set, on the right 0.1%.

7 Conclusions

We have proposed a new quantitative measure of the intrinsic dimensionality of a general metric space. Unlike previous attempts, this one is simple and cheap to compute for any metric space. We have shown that our measure is consistent with the classical concept of dimension in vector spaces. We have also proved lower bounds on the search times of large classes of proximity search algorithms in terms of this intrinsic dimension, therefore giving a formal justification for the so-called curse of dimensionality.

Future work involves showing that this measure of intrinsic dimension reasonably characterizes the performance of the proximity search algorithms on a given metric space, therefore giving a practical and extremely useful tool to predict the difficulty of searching on an unknown metric space.

Acknowledgements

We thank Benjamín Bustos for his help in the experiments of Figures 3 and 5.

References

[1] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), 1991.

[2] R. Baeza-Yates. Searching: an algorithmic tour. In A. Kent and J. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 37, pages 331–359. Marcel Dekker Inc., 1997.

[3] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[4] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. In *Proc. 5th South American Symposium on String Processing and Information Retrieval (SPIRE'98)*, pages 14–22. IEEE CS Press, 1998.

[5] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9):509–517, 1975.

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

[7] J. Bentley, B. Weide, and A. Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. on Mathematical Software*, 6(4):563–580, 1980.

[8] S. Berchtold, D. Keim, and H. Kriegel. The X-tree: an index structure for high-dimensional data. In *Proc. 22nd Conference on Very Large Databases (VLDB'96)*, pages 28–39, 1996.

[9] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 357–368, 1997. Sigmod Record 26(2).

[10] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)*, pages 574–584, 1995.

[11] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Comm. of the ACM*, 16(4):230–236, 1973.

[12] E. Chávez and J. Marroquín. Proximity queries in metric spaces. In R. Baeza-Yates, editor, *Proc. 4th South American Workshop on String Processing (WSP'97)*, pages 21–36. Carleton University Press, 1997.

[13] 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.

[14] E. Chávez, J. Marroquín, and G. Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-Based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999. `ftp://-garota.fismat.umich.mx/pub/users/elchavez/fqa.ps.gz`.

[15] 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`.

[16] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. of the 23rd Conference on Very Large Databases (VLDB'97)*, pages 426–435, 1997.

[17] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*, 1998.

[18] T. Cox and M. Cox. *Multidimensional Scaling.* Chapman and Hall, 1994.

[19] F. Dehne and H. Nolteimer. Voronoi trees and clustering problems. *Information Systems*, 12(2):171–175, 1987.

[20] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[21] C. Faloutsos and I. Kamel. Beyond uniformity and independence: analysis of R-trees using the concept of fractal dimension. In *Proc. 13th ACM Symposium on Principles of Database Principles (PODS'94)*, pages 4–13, 1994.

[22] C. Faloutsos and K. Lin. Fastmap: a fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. *ACM SIGMOD Record*, 24(2):163–174, 1995.

[23] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd edition, 1990.

[24] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[25] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[26] J. Hair, R. Anderson, R. Tatham, and W. Black. *Multivariate Data Analysis with Readings.* Prentice-Hall, 4th edition, 1995.

[27] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5), 1983.

[28] D. Kao, R. Bergeron, and T. Sparr. Mapping metric data to multidimensional spaces. Technical Report TR 97-13, Dept. of Computer Science, Univ. of New Hampshire, 1997.

[29] J. Kruskal and M. Wish. *Multidimensional Scaling.* SAGE Publications, Beverly Hills, 1978.

[30] L. Micó, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.

[31] 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. *Pattern Recognition Letters*, 15:9–17, 1994.

[32] G. Navarro. Searching in metric spaces by spatial approximation. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, pages 141–148. IEEE CS Press, 1999.

[33] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, 1997.

[34] H. Nolteimer, K. Verbarg, and C. Zirkelbach. Monotonous Bisector* Trees – a tool for efficient partitioning of complex schenes of geometric objects. In *Data Structures and Efficient Algorithms*, LNCS 594, pages 186–203. Springer-Verlag, 1992.

[35] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[36] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[37] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

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

[39] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.

[40] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.