Information Systems 64 (2017) 206–218

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

An empirical evaluation of intrinsic dimension estimators[☆]

Gonzalo Navarro[a], Rodrigo Paredes[b,*], Nora Reyes[c,*], Cristian Bustos[c]

[a] Center of Biotechnology and Bioengineering, Department of Computer Science, University of Chile, Chile
[b] Departamento de Ciencias de la Computación, Universidad de Talca, Chile
[c] Departamento de Informática, Universidad Nacional de San Luis, Argentina

ARTICLE INFO

Article history:
Received 16 December 2015
Accepted 10 June 2016
Available online 16 June 2016

Keywords:
Intrinsic dimension
Complexity of searching
Metric spaces

ABSTRACT

We study the practical behavior of different algorithms and methods that aim to estimate the intrinsic dimension (IDim) in metric spaces. Some of them were specifically developed to evaluate the complexity of searching in metric spaces, based on different theories related to the distribution of distances between objects on such spaces. Others were originally designed for vector spaces only, and have been extended to general metric spaces. To empirically evaluate the fitness of various IDim estimations with the actual difficulty of searching in metric spaces, we compare two representatives of each of the broadest families of metric indices: those based on pivots and those based on compact partitions. Our conclusions are that the estimators Distance Exponent and Correlation fit best their purpose.

1. Introduction

Similarity search in metric spaces has received much attention due to its applications in many fields, ranging from multimedia information retrieval to machine learning, classification, and searching the Web. While a wealth of practical algorithms exist to handle this problem, it has been often noted that some datasets are intrinsically harder to search than others, no matter which search algorithms are used. An intuitive concept of "curse of dimensionality" has been coined to denote this intrinsic difficulty, but a clear method to measure it, and thus to predict the performance of similarity searching in a space, has been elusive.

The similarity between a set of objects $\mathbb{U}$ is modeled using a *distance function* (or *metric*) $d: \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}^+ \cup \{0\}$ that satisfies the properties of triangle inequality, strict positivity, reflexivity, and symmetry. In this case, the pair $(\mathbb{U}, d)$ is called a *metric space* [1–4].

In some applications, the metric spaces are of a particular kind called "vector spaces" of finite *explicit* or *representational* dimension, where the elements consist of D coordinates of real numbers. In this case, we can use some Minkowski metric or any other metric appropriate to the specific case (for instance, the cosine distance) as the dissimilarity measure between two objects. Many works exploit the geometric properties of vector spaces, but they usually cannot be extended to general metric spaces, where the only available information is the distance between objects. Since in most cases the distance is very expensive to compute, the main goal when searching in metric spaces is to reduce the number of distance evaluations. In contrast, vector space operations tend to be cheaper and the primary goal when searching them is to reduce the CPU cost or the number of I/O operations carried out.

Similarity queries are usually of two types. For a given database $S \subseteq \mathbb{U}$ with size $|S| = n$, $q \in \mathbb{U}$ and $r \in \mathbb{R}^+$, the *range query* $(q, r)_d$ returns all the objects of S at distance at most r

☆ Partially funded by basal funds FB0001, Conicyt, Chile and Fondecyt grant 1131044, Chile.

* Corresponding authors.

E-mail addresses: gnavarro@dcc.uchile.cl (G. Navarro), raparede@utalca.cl (R. Paredes), nreyes@unsl.edu.ar (N. Reyes), cjbustos@unsl.edu.ar (C. Bustos).

http://dx.doi.org/10.1016/j.is.2016.06.004
0306-4379/© 2016 Elsevier Ltd. All rights reserved.

from q, formally $(q,r)_d = \{x \in S, d(x,q) \le r\}$; whereas the *nearest neighbor query* $kNN_d(q)$ retrieves the k elements of S that are closest to q, that is, $kNN_d(q)$ is a set such that for all $x \in kNN_d(q)$ and $y \in S \setminus kNN_d(q)$, $d(q,x) \le d(q,y)$, and $|kNN_d(q)| = k$.

A naïve way to answer similarity queries is to compare all the database elements with the query q and return those elements that are close enough to q. This *brute force* approach is too expensive for real applications. Research has then focused on ways to reduce the number of distance computations performed to answer similarity queries. There has been significant progress around the idea of building an *index*, that is, a data structure that allows discarding some database elements without explicitly comparing them to q. Moreover, there are some relatively recent works [5–10] that try to get jointly the goals of reducing the number of distance evaluations and the number of I/O operations performed.

In vector spaces with uniformly distributed data, the *curse of dimensionality* describes the well-known exponential increase of the cost of all existing search algorithms as the dimension grows. Non-uniformly distributed vector spaces may be easier to search than uniform ones, despite having the same explicit dimensionality. The phenomenon also extends to general metric spaces despite their absence of coordinates: some spaces are intrinsically harder to search than others. This has lead to the concept of *intrinsic dimensionality* (*IDim*) of a metric space, as a measure of the difficulty of searching it. A reliable measure of IDim has been elusive, despite the existence of several formulae.

Computing the IDim of a metric space is useful, for example, to determine whether it is amenable to indexing at all. If the IDim is too high, then we must just resort to brute-force solutions or to approximate search algorithms (which do not guarantee to find the exact answers). Even when exact indexing is possible, the IDim helps decide which kind of index to use and how to tune it. For example, in lower dimension spaces, a pivot-based method works fine using a small set of pivots; whereas in higher dimensions we need to use a large set of pivots [1], which also implies a large amount of memory for the index. Alternatively, if we do not have enough extra memory for the index, we can switch to the *List of Clusters* [11], which has reasonable performance in high dimension spending little space in the index.

In this work we aim to empirically study the fitness of various IDim measures to predict the search difficulty of metric space searching. Some measures were specifically developed for metric spaces, based on different theories related to the distribution of distances between objects. Others were originally designed for vector spaces and have then been adapted to general metric spaces. We chose various synthetic and real-life metric spaces and four indexing methods that are representatives of the major families of indices: two based on pivots and two based on compact partitions. Our comparison between real and estimated search difficulty yields that *Distance Exponent* [12,13] and *Correlation* [14] are currently the best predictors in practice, however all the estimators behave relatively well.

The rest of this paper is organized as follows. In Section 2, we review some relevant issues of IDim estimators for vector spaces. Next, in Section 3, we survey four methods for estimating IDim in vector spaces and show how to adapt them to the metric case. We also include three new IDim estimators for general metric spaces. The experimental evaluation for the seven methods is presented in Section 4. We finally draw our conclusions and future work directions in Section 5. An early version of this work appeared in [15].

2. Intrinsic dimension estimators for vector spaces

There are several interesting applications where the data are represented as D-dimensional vectors in $\mathbb{R}^D$. For instance, in pattern recognition applications, objects are usually represented as vectors [16]. Therefore, data are embedded in $\mathbb{R}^D$, even though this does not imply that its *intrinsic* dimension is D.

There are many definitions of IDim. For instance, the IDim of a given dataset is the minimum number of free variables needed to represent the data without loss of information [17]. In general terms, a dataset $\mathbb{X} \subseteq \mathbb{R}^D$ has IDim $M \le D$, if its elements fall completely within an M-dimensional manifold of $\mathbb{R}^D$ [18]. Another intuitive notion is the logarithm of the search cost, as in many cases this cost grows exponentially with the dimension.

Even in vector spaces, there are many reasons to estimate the IDim of a dataset. Using more dimensions (more coordinates in the vectors) than necessary can bring several problems. For example, the space to store the data may be an issue. A dataset $\mathbb{X} \subseteq \mathbb{R}^D$ with $|\mathbb{X}| = n$ requires to store $n \times D$ real coordinates. Instead, if we know that the IDim of $\mathbb{X}$ is $M \le D$, we can map the points to $\mathbb{R}^M$ and just store $n \times M$ real coordinates. The CPU cost to compute a distance is also reduced. This can in addition help identify the important dimensions in the original data. Also, as the amount of available information increases, compressing the data storage becomes even more important. Secondly, as the asymptotic complexity of the algorithms is monotonically increasing with respect to the dataset dimensionality, a dimensionality reduction (to the actual dataset IDim) can produce an important CPU time reduction. For instance, in the case of data classification or pattern recognition, producing reliable classifiers is difficult when the dataset dimensionality is high (*curse of dimensionality* [19]); and according to the theoretical approximation of statistical learning [20], the classifier generalization capability depends on the IDim of the space.

There are two approximations to estimate the IDim of a vector space [16,17], namely, *local* and *global* methods. The local ones make the estimation by using the information contained in sample neighborhoods, avoiding the data projection over spaces of lower dimensionality. The global ones deploy the dataset over an M-dimensional space using all the dataset information. Unlike the local methods that only use the information contained in the neighborhood of each data sample, global methods use whole information of the dataset.

In this work we focus on global IDim estimators. That is, we consider all the dataset information to estimate the IDim as accurately as possible. Global methods can be split into three families: projection techniques, multidimensional scaling methods, and fractal based methods. The last two are more suitable to extend to metric spaces, so we have selected and adapted some representatives of these groups.

3. Intrinsic dimension estimators for metric spaces

In general metric spaces, since the curse of dimensionality severely affects the performance of the search process, knowing the IDim can help choose a metric index appropriate to the space dimension and also give some insight on the specific index tuning. For instance, in low IDim spaces, where searching is easier, pivot based indices usually perform better, even when using a small set of pivots. However, they can fail in high IDim spaces, or hard spaces, as a large set of pivot is needed to preserve the performance at the cost of an excessive amount of space for the index. Alternatively, if there is little amount of extra memory, we can use the *List of Clusters* (LC) [11], which is a very appropriate, RAM economical index.

Hence, a proper estimation of the operating dataset IDim is very important, as it helps improve the time and memory costs of the selected solution.

There are few IDim estimators that apply directly in general metric spaces. The IDim estimators that are proper to metric spaces can only consider the dataset objects and their distances between each other.

In this section we analyze various methods to estimate the IDim of vector spaces and others to general metric spaces. We discuss how to adapt the former to the case of general metric spaces. Note that, since multidimensional spaces are a particular case of metric spaces, our estimators can also be applied to obtain the IDim of D-dimensional vector spaces.

3.1. Fractal based methods

Unlike other families, fractal based methods can estimate non-integer IDim values. The most popular techniques of this family are *Box Counting* [21], which is a simplified version of the *Haussdorff dimension* [22,23], and *Correlation* [14]. These techniques have been successfully used to estimate the dimensionality of the underlying dynamic systems that generate time series [24].

The dimension estimation by Box Counting D_B of a set $\Omega \subseteq \mathbb{R}^D$ is defined as follows: if $v(r)$ is the number of boxes of size r needed to cover Ω, then

$$D_B = \lim_{r\to 0} \frac{\ln(v(r))}{\ln\left(\frac{1}{r}\right)}. \tag{1}$$

In this method, the boxes are multidimensional regions of side r on each dimension (that is, they are hypercubes of side r). Regrettably, even though efficient algorithms have been proposed, the Box Counting dimension can be computed only for low dimension datasets, because its algorithmic complexity grows exponentially with the dimension.

Estimating the dimension by Correlation is an alternative to Box Counting. It is defined as follows. Let $\Omega = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^D$ and the correlation integral

$$C_m(r) = \lim_{n\to\infty} \frac{2}{n(n-1)} \sum_{1\le i<j\le n} I(\|x_j - x_i\| < r), \tag{2}$$

where $I(\cdot)$ is the indicator function. Intuitively, $C_m(r)$ is the fraction of object pairs whose distance is lower than r. So, the dimension estimation by Correlation D_C is

$$D_C = \lim_{r\to 0} \frac{\ln(C_m(r))}{\ln r}. \tag{3}$$

3.1.1. Correlation

The most popular method to estimate the dimension by Correlation is the log–log plot. It consists in plotting $\ln(C_m(r))$ versus $\ln(r)$. The dimension by Correlation is the slope of the linear section of the curve.

To illustrate the process of estimating IDim using the Correlation estimator, in Fig. 1 we show an example of its computation on the real world metric space Histograms (this dataset is described in Section 4.2). The line plotted by circles corresponds to the curve $\ln(C_m(r))$ versus $\ln(r)$, obtained from the experimental data. We estimate the IDim of this dataset by computing the slope of the linear section at the beginning of the plot (drawn with a line). Note that this is the section of the curve that shows the usual exponential growth of the fraction $C_m(r)$ with respect to the intrinsic dimensionality of the space. At the end of the linear section of the plot, $C_m(r)$ almost reaches its maximum value, so the growth beyond this linear section is very mild. Hence, we need to neglect this section of the curve, otherwise we can underestimate the space intrinsic dimensionality. To compute the slope we use linear regression with least squares over the first linear section of the curve. Note that this procedure allows us to estimate IDim with a dataset of modest size, because we are only focused on the section of the curve that does reveal exponential growth.

3.1.2. Ball counting

Analogously with Correlation, to estimate the dimension by Box Counting, we can compute the slope of the linear section of the curve $\ln(v(r))$ versus $\ln(1/r)$. However, in the general case of metric spaces, we do not have coordinates. Thus, to adapt the Box Counting method, we consider *balls* of radius r, that is, the set of objects within a distance r from a reference object o. We randomly pick the reference objects from the dataset, and count the number $B(r)$ of balls of radius r needed to cover the dataset. To do so, we use the *List of Clusters* (*LC*) index [11], whose code is available from SISAP [25], with the variant of fixed radius and centers chosen at random. Then, $B(r)$ is just the length of the LC.

To estimate the dimension by Box Counting, which in this case is Ball Counting, we replace $\ln(v(r))$ by $\ln(B(r))$, plot $\ln(B(r))$ versus $\ln(\frac{1}{r})$ in log–log and obtain the IDim as the slope of the linear section of the curve by using linear

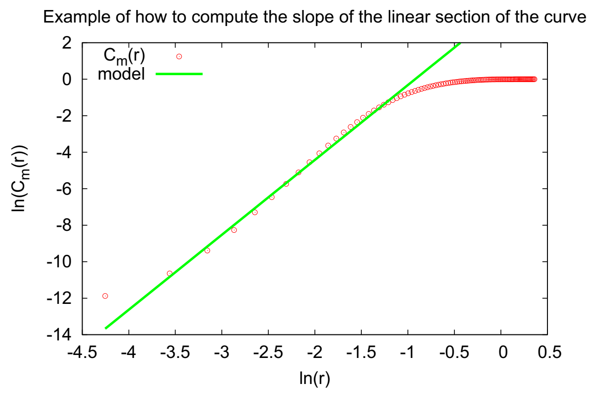


Fig. 1. IDim estimation with Correlation by computing the slope of the linear section of the curve.

regression with least squares over the experimental data $(\ln(B(r)), \ln(\frac{1}{r}))$, using a procedure analogous to Correlation.

3.2. Distance exponent

Traina et al. [12,13] discuss the problem of the selectivity estimation for range queries in real-world metric spaces, including spatial or multidimensional datasets as special cases. It plays an important role when analyzing real metric spaces. Their main finding is that several datasets follow the so-called *Power Law*. They call *Distance Exponent* the exponent of the power law, and show how to use it to derive formulae for estimating the selectivity of range queries. For instance, the number of objects relevant to the query, the number of I/Os to answer the query when the data is stored on disk, the amount of time needed to answer the query, and so on.

To find a formula that estimates the number of neighbors of objects within a distance r in a dataset of n-objects, they introduce the following notions: (i) the *Distance Plot* of a metric set is the number of object pairs at distance at most r versus the distance r, and both axes are drawn in logarithmic scale; and (ii) the *Distance Exponent* is the slope of the line that better fits the distance plot in case it is linear for a range of scales. Using these two notions, they define the *Distance Law*.

Definition 1 (*Distance Law*). Given a dataset of n objects from a metric space with distance function $d(x,y)$, the average number of distances lower than a radius r follows a power law; that is, the average number of neighbors $\overline{nb}(r)$ within a distance r is proportional to $r^{\mathcal{D}}$. Formally,

$$n \cdot \Phi(r) = \overline{nb}(r) \propto r^{\mathcal{D}}, \tag{4}$$

where n is the number of objects in the dataset and $\Phi(r)$ is the accumulated distribution function of the probability of a pair of objects to be within a distance r.

If a dataset has a metric to evaluate the distance between every object pair, then this plot can always be drawn. They show that the distance plot has an almost linear behavior for many databases, both real and synthetic. Building the distance plot requires $O(n^2)$ distance computations. To reduce this cost, $\overline{nb}(r)$ is estimated using an index [12], in particular the *M-tree* [5]. That is, a way to estimate the distance exponent $\mathcal{D}$ of a dataset stored in a metric index is by means of the very same index.

Since in this work we are only interested in comparing the different IDim measures, indexing the space is not necessary and we compute $\overline{nb}(r)$ directly, considering a reference object chosen at random from the dataset. We only determine the number of elements at distance r from that object. The result is averaged over various choices for the object.

3.3. Fastmap

This method arises from the proposal [26] of a fast algorithm to map objects of any metric space onto points of a k-dimensional space (k being defined by the user), so that the dissimilarities are preserved. Its goal is to speed up searches in traditional or multimedia databases.

To do so, the objects are mapped onto the k-dimensional space using k feature extraction functions, provided by domain experts [26]. The main issue is how to define such feature extraction functions. For example, in the metric case of strings with the edit distance [27], it is not clear which features can be considered.

For a domain expert, it is generally easier to provide a distance function to compare objects than to provide feature extraction functions. *Fastmap* [28] is a generalization of the original method [26], where the objects are mapped using only a distance function.

Fastmap finds, given a dataset of n objects from a metric space $(\mathbb{U}, d)$, n image points in a k-dimensional target space, such that the distances between the objects in the original space are preserved as much as possible in the target space.

For evaluating the dissimilarity preservation in the target space, a *stress* function is defined as follows,

$$stress^2 = \frac{\left(\sum_{i,j}(\hat{d}_{ij} - d_{ij})^2\right)}{\left(\sum_{i,j} d_{ij}^2\right)}, \tag{5}$$

where d_{ij} is the dissimilarity measure (the distance of the original space) between objects o_i and o_j, and $\hat{d}_{ij}$ is the Euclidean distance between their respective images p_i and p_j. The stress function gives the relative error that the distances in the target space suffer on average after the transformation. Fastmap begins with an estimation that is iteratively improved, until no additional improvement is possible.

In the metric case, we can assume that we have the $n \times n$ matrix of distances between all the dataset objects, and Fastmap must find n points in the k-dimensional space whose Euclidean distances are close to the original matrix of $n \times n$ distances. The crux is to assume that objects are points in some m-dimensional space, with unknown m, and to project these points over k mutually orthogonal directions. The challenge is to compute all these projections using only the distance matrix. Fastmap projects the objects over carefully selected lines. It chooses two objects

o_a and o_b, and considers the "line" passing through them in the original space. The projections x_i' of the objects over this line are obtained using the *cosine law*:

Theorem 1 (*Cosine Law*). *Any triangle* $o_a \overset{\triangle}{o_i} o_b$ *satisfies*:

$$d(o_b, o_i)^2 = d(o_a, o_i)^2 + d(o_a, o_b)^2 - 2x_i' d(o_a, o_b). \tag{6}$$

Eq. (6) can be solved for x_i' to compute the projection of object o_i with the formula

$$x_i' = \frac{d(o_a, o_i)^2 + d(o_a, o_b)^2 - d(o_b, o_i)^2}{2d(o_a, o_b)}. \tag{7}$$

Thus, the input of Fastmap is a set S of size n and, in each iteration, it computes the coordinates of all the n objects over the new axis. So, after k iterations, it produces a k-dimensional target space S' where each object $o_i \in S$ is mapped to a k-coordinate vector $p_i = (x_{i,1}', x_{i,2}', \ldots, x_{i,k}') \in S'$, where $x_{i,j}'$ is the jth projection of the image p_i of the object o_i.

In our case, we want to estimate the number of projections needed so that the target space reaches a mapping with a small enough *stress*, that is, preserving the distances sufficiently well. Thus, we modify the Fastmap algorithm so that it computes the *stress* of the target space after each new dimension is added. If the difference between the current and the previous *stress* values is significant, we compute another projection (thus increasing the dimensionality of the target space). Otherwise, the current dimension of the target space is reported as the estimation of the IDim of the original metric space.

3.4. Principal Component Analysis

Principal Component Analysis (*PCA*) [29] is a statistical procedure that projects the data onto new axes, called the principal components, where the axes are ordered by maximum to minimum variance. As the first components accumulate most of the variance, the original data can be projected using the first components controlling how much information we want to preserve or lose (and we can use more components if we want to preserve a larger amount of the data information).

A common application of PCA is to reduce the dimensionality of a vector dataset by neglecting the components with small variance, as they have minimum impact in the amount of information that the projected dataset will have. So, we can identify the number of selected components as the IDim of the dataset.

The crux of PCA is that it finds a set of basis vectors, where the first component follows the maximum variance direction, the second follows the next variance direction, and so on. That is, each component accounts for as much of the variability in the data as possible. The resulting vectors, called principal components, are an uncorrelated orthogonal basis set, because they are the eigenvectors of the data covariance matrix.

In the metric case, we do not necessary have objects with coordinates. So, the first step is to represent the object from a given space as a vector. For this purpose, we simply select a set of random pivots and compute the pivot table. Thus, each object is represented as a row in the table, a vector, where its components are the distances to every pivot. After this, we compute the principal components of the pivot table. In the process, the components are sorted by their importance, that is, in decreasing proportion of the variance of the data. So the first components should accumulate most of the variance.

We start the computation with an educated guess. In the synthetic metric spaces, we use a set of pivots two times bigger than the representational dimension. The underlying intuition is that we should not estimate an IDim bigger than the representational dimension, but, as we use random pivots, we grant the pivot set the chance of incorporating the maximum of distance information between the objects in the dataset. On the other hand, in the case of real world metric spaces we use a set of 20 pivots, as our experience says that the real world metric space under study has a much lower IDim.

Since we need to fix a threshold, we use the number of components that accumulates 90% of the variance of the dataset as the IDim estimation of the space. This threshold gave us the best results in our experiments. To perform the statistics computation we use R [30].

It is important to verify whether the IDim of the given metric space is preserved after representing the objects as vectors. To do so, we carry out a preliminary study which consists in calculating the intrinsic dimensionality computed with the estimator Distance Exponent over the pivot table. The results of the study are shown in Tables 1 and 2, where IDim Space is the IDim estimation that Distance Exponent suggests for the respective space and IDim PT is the estimation for the vectors in the pivot table.

As can be seen, the IDim computed with Distance Exponent over the pivot table is similar to the one computed on the original dataset, with the exception of the space of strings (ENG) and documents (DOCS). These results validate our application of PCA. The mismatch in the case of strings can be due to the discrete nature of the space, and in the case of documents, to the extremely concentrated histogram of distances. For example, several other estimators yield similar numbers for these two spaces.

3.5. Intrinsic search difficulty

Chávez et al. [1] introduced a measure of the intrinsic complexity of searching in general metric spaces, which is easy to estimate and is independent of the search algorithm.

Several authors [31–33] have proposed to use the *distance histogram* to characterize the hardness of searching in arbitrary metric spaces, yet the cost was tailored to a specific index. This measure [1] is the first quantitative definition. It depends only on the histogram and not on any assumption on the indexing method.

The intuition behind this measure is that, in random vectors in D dimensions, the histogram has a larger mean μ and a smaller variance σ^2 as D increases. In fact, it holds $D = c \cdot \mu^2/\sigma^2$ for some constant c [34]. Thus, the same formula could be used to estimate a dimension D from the mean and variance of the histogram of distances in a

Table 1
Verification of the pivot table IDim for synthetic spaces.

IDim/Space	C5	C10	C15	C20	G5	G10	G15	G20	G101
IDim_{Exp} (Space)	5.08	8.40	12.13	15.80	4.73	8.83	13.46	15.80	0.92
IDim_{Exp} (PT)	4.70	7.45	11.17	13.92	4.83	8.01	11.98	13.40	0.96

Table 2
Verification of the pivot table IDim for real world spaces.

IDim/Space	ENG	NASA	DOCS	HIS
IDim_{Exp} (Space)	4.96	3.70	3.52	4.70
IDim_{Exp} (PT)	9.29	3.02	0.57	3.54

general metric space. We do not have the histogram of the whole universe $\mathbb{U}$, but we can approximate it using the histogram of the dataset $S \subset \mathbb{U}$, considering the set S as a random sample of $\mathbb{U}$.

Definition 2 (*Intrinsic search difficulty*). Let μ be the mean and σ^2 be the variance of the histogram of distances of a metric space. Then, its *intrinsic search difficulty* is

$$\rho = \frac{\mu^2}{2\sigma^2}. \tag{8}$$

An obvious advantage of this measure, which has contributed to its popularity, is that ρ is easy to compute from a reasonable sampling of pairs in S. Other techniques require more complex and expensive computations.

Pestov [35] presents a system of three axioms an intrinsic dimension function must satisfy. He proves that the intrinsic dimension measure ρ satisfies a weak version of these axioms. Later [36], he introduces a set of goals an intrinsic dimension function should fulfill, and ρ achieves many of them.

As the measure ρ has been designed for general metric spaces, we use it as is. We consider the dataset S and we compute all the distances $d(x, y), \forall x, y \in S$. Then we compute the average $\mu = \frac{1}{n^2}\sum_{x,y \in S} d(x, y)$ and the variance $\sigma^2 = \frac{1}{n^2}\sum_{x,y \in S}(d(x, y) - \mu)^2$. Finally, we obtain the value of $\rho = \frac{\mu^2}{2\sigma^2}$ and report it as the IDim of the metric space.

3.6. *Sparse spatial selection method*

This method is based on a very simple pivot selection strategy [37], called *Sparse Spatial Selection* (*SSS*). This strategy has the advantage that it is not required to know the number of pivots in advance.

Initially, the set of pivots contains only the first object of the collection. Then, for each element $x \in S$, x is chosen as a new pivot if its distance to every pivot in the current set of pivots is equal to or greater than αd^+, with α being a constant parameter (for indexing purposes this constant α takes values around 0.5) and d^+ the maximum distance between two objects in the space. That is, an object in the collection becomes a new pivot if it is located at more than a fraction of the maximum distance d^+ with respect to all the current pivots. For example, if $\alpha = 0.5$ an object is chosen if it is located further than a half of the maximum distance d^+ from the already selected pivots.

Therefore, the selected pivots will not be too close to each other. Forcing the distance between two pivots to be greater than or equal to αd^+, it is ensured that they are well distributed in the whole space. It is important to take into account that these pivots are not very far away from each other, neither very far from the rest of the objects in the collection (i.e., they are not necessarily *outliers*), but they are well distributed and cover the whole space.

An distinguishing feature of SSS is that an element x is compared against the pivots already selected and it becomes a new pivot if needed. In this way the number of pivots does not depend on the collection size but on its intrinsic dimensionality. This number of pivots is very similar to the optimum number for other strategies.

We note that surrounding a pivot, there is a ball of radius αd^+. Also, the method produces pivots as long as there is some part of the space that is not covered by any previous pivot. This resembles the fractal methods. So, we can use a similar technique to estimate IDim. Let $P(\alpha)$ be the number of pivots produced by SSS for a given value of α. So, we plot $\ln P(\alpha)$ versus $\ln \frac{1}{\alpha}$ and obtain the slope of the linear section of the curve by using linear regression with least squares over the experimental data $\left(\ln(P(\alpha)), \ln\left(\frac{1}{\alpha}\right)\right)$.

4. Experimental evaluation

We evaluate experimentally the seven IDim estimators described on general metric spaces. We consider two kinds of metric spaces, depending on the data source:

Synthetic: These are spaces generated artificially so that they present some interesting characteristic to be evaluated. For instance, uniformly distributed vectors in $\mathbb{R}^D$ with known dimension.

Real world: These are metric spaces obtained from real-world applications. For instance, a feature vector space of images obtained from a NASA image set.

4.1. *Synthetic metric spaces*

These are vector spaces with Euclidean distance. They are treated as metric spaces, as we do not consider the coordinate information. A first set is formed by vectors with uniform distribution, so that the representational dimension matches the IDim. Here, we can test the

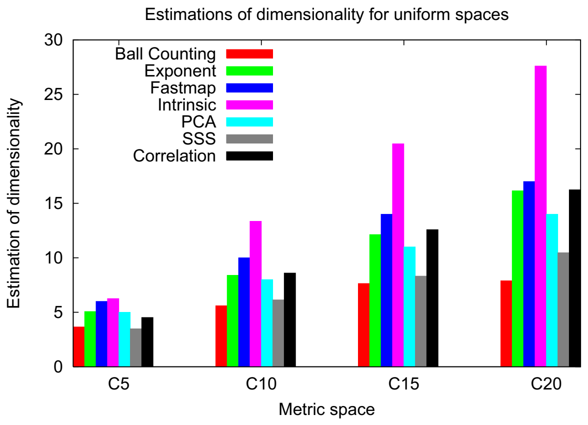


Fig. 2. Comparison of dimensionality estimations for uniform spaces.

estimators in a case where the IDim is known. A second set is formed by vectors with Gaussian distribution, so that the representational dimension is greater than the IDim (the more clustered is the space, the lower is the IDim). The distance is also Euclidean. Here, we aim to check whether the estimators give lower values as the IDim decreases.

4.1.1. *Uniformly distributed vectors with Euclidean distance*

We generate four datasets of 100,000 uniformly distributed vectors in the unitary cube $(0,1)^D$, with D=5, 10, 15, and 20. The spaces are called C5, C10, C15, and C20, respectively.

Fig. 2 depicts the estimations obtained with the seven IDim estimators, namely, Ball Counting, Distance Exponent, Fastmap, Intrinsic Search Difficulty, PCA, SSS, and Correlation, for these four metric spaces. As it can be seen, Ball counting becomes insensitive to the correct dimension in C20. The other six methods increase proportionally with D, but with different slopes. Fastmap is the one with the best fit, matching D almost perfectly, closely followed by Correlation and Distance Exponent. Intrinsic Search Difficulty based estimator shows a consistent factor multiplying D. Both PCA and SSS fit well in C5, but as IDim grows, the fit loses precision.

Search degradation as IDim grows: To verify that the dataset IDim is responsible for the search degradation, we pick C5 and extend its vectors with zeroes to produce spaces with 10, 15, and 20 representational dimensions, and study the search performance over it.

We perform 10 executions of the algorithms, building the index with 90% of the database elements and reserving the remaining 10% (chosen at random) for the queries. So, the query objects do not belong to the index. We average the results over the 10 executions. In each execution, the objects in the metric space are permuted at random. Therefore, each of the 10 indices uses a different dataset S, and the query objects are also different.

We use two pivot indices and two compact partition indices. For the pivot index family, we use the generic pivot algorithm and the Vantage Point Tree index [38,39].

In the case of the generic pivot algorithm, we choose at random a set of pivots $\mathcal{P}=\{P_1,P_2,\ldots,P_k\}\subset S$ of size $|\mathcal{P}|=k$. We store the kn distances between pivots and objects, and use them to filter out candidates using the triangle inequality. For each space, we experimentally determine the number of pivots that obtains the best search performance. Thus, the results shown for each case correspond to the best possible ones for this method, in the corresponding metric space.

On the other hand, the Vantage Point tree (VPT) is a tree recursively built by taking an arbitrary element p as the root. The distances from the root to every object in the database are computed $\{d(p,u), u\in\mathbb{U}\}$. Let M be the median of those distances. All objects such that $d(p,u)\le M$ are assigned to the left node and the rest to the right node. Then, we recurse until the number of elements is smaller than a certain bucket size. To solve a query in the VPT the query ball is tested to see if there could be candidates in the left and right nodes. It is possible to enter both subtrees.

For the case of compact partition based algorithms, we consider the LC [11], which is one of the best indexes for medium and high dimensions, and the Spatial Approximation Tree [40]. We use the LC variant that has a maximum size for each cluster. For each metric space considered, we experimentally determine the cluster size whose performance is the best, and this is the result shown in the plots.

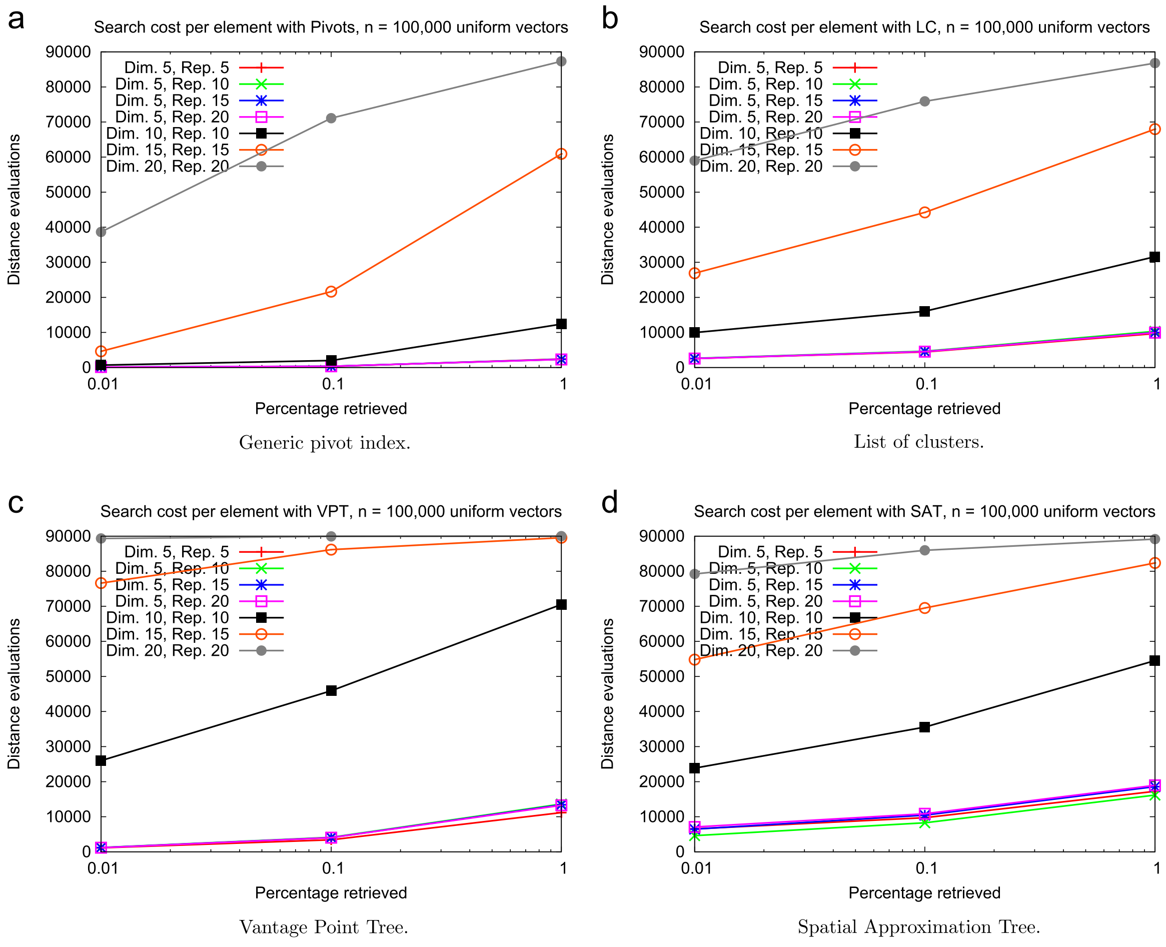
Finally, the Spatial Approximation Tree (SAT) is a data structure aiming at approaching the query spatially by starting at the root and getting iteratively closer to the query by navigating the tree. The SAT is built as follows. An element a is selected as the root, and is connected to a set of *neighbors* $N(a)$, defined as a subset of elements x in the dataset such that x is closer to a than to any other element in $N(a)$. The other elements (not in $N(a)\cup\{a\}$) are assigned to their closest element in $N(a)$. Each element in $N(a)$ is recursively the root of a new subtree containing the elements assigned to it.

In Fig. 3, we show the cost of range queries retrieving 0.01%, 0.1% and 1% of the vector dataset per query, using the generic pivot index, the LC, the VPT, and the SAT, see Fig. 3(a), (b), (c), and (d), respectively. These results are compared with the ones for searching C10, C15, and C20. The four plots show that the search effort performed by the four tested indices remain almost unaltered when working on the four spaces of IDim 5 (independently of the representational dimension of the space), while the curves for C10, C15, and C20 show the usual degradation.

4.1.2. *Gaussian distributed vectors with Euclidean distance*

We generate 100,000 vectors in $\mathbb{R}^D$, where each coordinate has mean $\mu=1$ and variance $\sigma^2=0.1$, for $D=$ 5, 10, 15, and 20. In these spaces, there are no, a priori, clusters of elements. These spaces are called G5, G10, G15, and G20. Note that the object are not confined in the unitary cube.

We also generate 100,000 vectors in $\mathbb{R}^{101}$ with 200 clusters. In this space, the first 100 coordinates of each vector follow a $\mathcal{N}(\mu=1,\sigma^2=0.1)$ distribution. The 101th coordinate stores the cluster identifier. So, the cluster centers are essentially uniformly distributed in the last coordinate. Geometrically speaking, one can imagine that



Generic pivot index.

List of clusters.

Vantage Point Tree.

Spatial Approximation Tree.

Fig. 3. The search effort does not vary when the IDim of the space does not change.

this space is a sequence of 200 crisp clusters in a line immersed in $\mathbb{R}^{101}$. This space is called G101.

Fig. 4 shows the estimations obtained with the seven IDim estimators for these metric spaces. As can be seen, all the methods give increasing IDim values as the representational dimension grows (G5 through G20) as expected, but with different behaviors. Ball Counting and Distance Exponent become less sensitive to high dimensionality, from G15 to G20 they show a small increment. Intrinsic, SSS and Correlation give IDim values that grow steadily from G5 through G20, and we note that the Intrinsic Search Difficulty gives markedly lower values than in the uniform case. Fastmap and PCA show a large increment from G15 to G20, and we note that the IDim estimated by Fastmap in G20 is higher than in C20, which is unexpected.

We note that the lower the IDim, the more similar the dimensionality estimation. In fact, all the measures estimate the IDim of G5 around 4–5 and the IDim of G101 around 1. This last result is very interesting. We prepare the dataset G101 with the purpose of having a space with high representational dimension but with very low intrinsic dimension and all the estimators detect this fact.

4.2. *Real metric spaces*

We pick four spaces from the Metric Library [25][1] in order to estimate their IDims with the seven IDim estimators The selected spaces are varied:

Dictionary: This is a dictionary of 69,069 English words. In this space, we use a discrete function, the *Edit Distance* or *Levenshtein Distance* [27].

NASA: This is a set of 40,700 images from NASA, represented as feature vectors of 20 real coordinates per vector, under the Euclidean distance. They were generated from images downloaded from the NASA photo and video archive site, used in contests conducted by the *Center for Discrete Mathematics and Theoretical Computer Science* (DIMACS).[2] To obtain images from the videos, cuts are detected based on the transition of the color histogram and then representative images

[1] Available at http://www.sisap.org/library/dbs/

[2] Available at http://www.dimacs.rutgers.edu/Challenges/Sixth/software.html

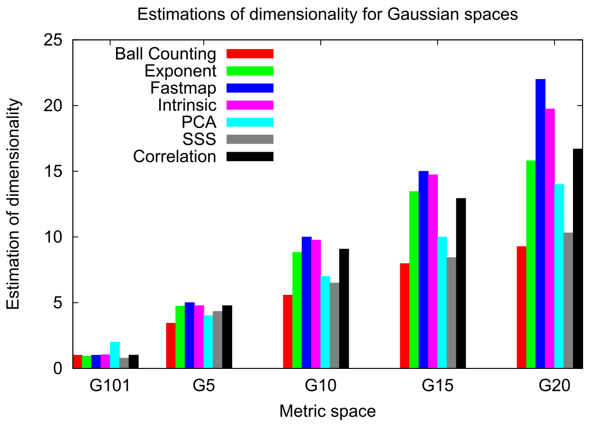


Fig. 4. Comparison of dimensionality estimations for Gaussian spaces.

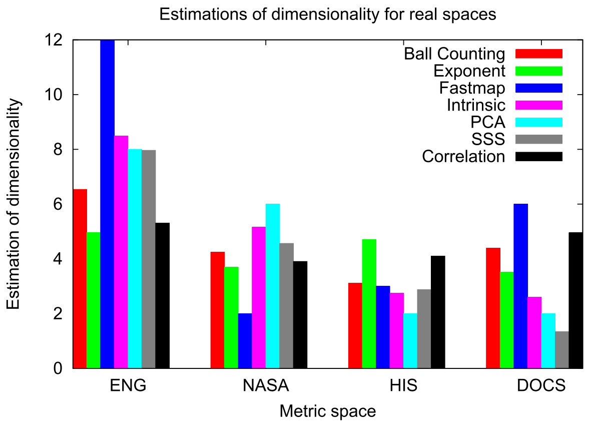


Fig. 5. Comparison of dimensionality estimations for real metric spaces.

are extracted when the changes in the histogram reach a given threshold. Later, the images are split into four sub-regions, i.e., upper-left, upper-right, lower-left and lower-right, and histograms of the subregions are calculated in order to take account of the composition of the image. The four histograms are concatenated to compose a 36-dimensional feature vector. Finally, using principal component analysis the feature vectors are reduced to 20-dimensional vectors.[3]

Histograms: This is a dataset of 112,682 histograms of medical images, each one composed by 8-D color histograms of 112 real components.[4] As any quadratic form function can be used as the distance in this case, we also have chosen the Euclidean distance, as is the simplest alternative.

Documents: This space has 1265 documents, represented as vectors according to the vector model of documents used in the Information Retrieval field. To compare documents we use the *cosine distance*. Each vector has a coordinate for each vocabulary term in the collection, and documents can be seen as points in a vector space of high representational dimension. The documents are files obtained form the TREC-3 collection.[5]

We start the experimental evaluation in real metric spaces by estimating their IDims. These results are shown in Fig. 5. As it can be seen, all the methods shown coincide in that the English dictionary apparently has the highest IDim. Also, in these real world metrics we can detect two groups of estimators that report similar values for IDims. The first is composed of Intrinsic Search Difficulty, PCA and SSS, and the second by Ball Counting, Distance Exponent and Correlation. On the other hand, Fastmap shows an erratic behavior.

To measure the intrinsic hardness of the searching, we consider the same four indices as before, using range queries:

Dictionary: As the metric is discrete, we use radii 1, 2, 3, and 4, retrieving on average about 0.003%, 0.037%, 0.326%, and 1.757% of the database.

NASA: In this continuous metric we use radii 0.012, 0.285, and 0.53, retrieving on average approximately 0.01%, 0.1%, and 1% of the dataset.

Histograms: This metric is also continuous. To retrieve on average approximately 0.01%, 0.1%, and 1% of the dataset, we use query radii 0.051768, 0.082514, and 0.131163.

Documents: The distance is also continuous. We use query radii 0.14, 0.15, and 0.195, which retrieve on average 0.01%, 0.1%, and 1% of the database.

Fig. 6 shows the correlation between the search cost with the Pivot index (on the left) and the List of Clusters (on the right), and the estimation reported for each considered IDim estimator, namely, Ball counting, Distance Exponent, Fastmap, Intrinsic Search Difficulty, PCA, SSS, and Correlation.

Fig. 7 illustrates the correlation between the search cost with the Vantage Point Tree (on the left) and the Spatial Approximation Tree (on the right) with respect to the seven estimators.

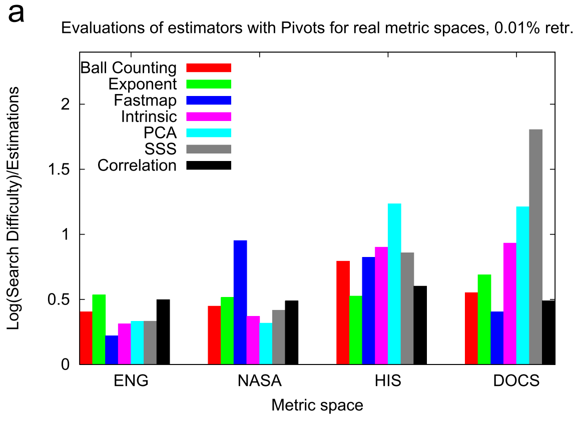
We plot the ratio between the logarithm of the search cost, measured with distance computations, and the estimations of the IDim. This measures how close is the logarithm of the actual search costs to the predicted IDim: if the search cost is consistently $s = c^d$, where d is the predicted IDim and c is a constant, then the plots should always be close to $\log c$. Thus the best methods are those that give roughly the same value regardless of the index used. In Table 3, we show the mean and standard deviation of log(Search Difficulty)/IDim obtained for each estimator. To compute the table, we consider the four real world metric spaces, the three radii, and the four indices considered for each estimator.

[3] More details on this dataset can be obtained from http://www.dimacs.rutgers.edu/Challenges/Sixth/participants.html#KS

[4] Available at http://www.dbs.informatik.uni-muenchen.de/~seidl/DATA/histo112.112682.gz

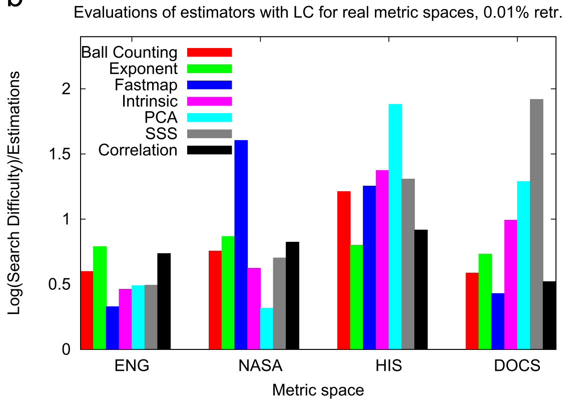
[5] Available at http://trec.nist.gov

a



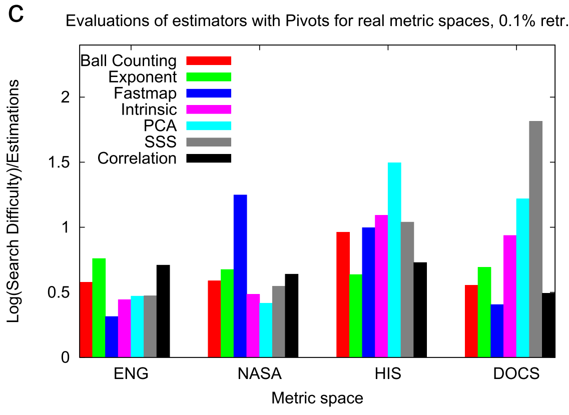
Retrieving 0.01%.

b



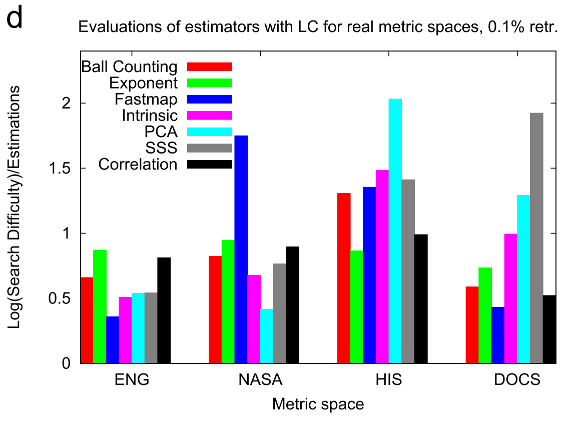
Retrieving 0.01%.

c



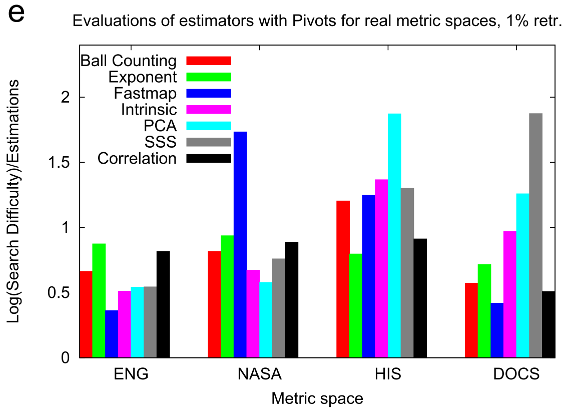
Retrieving 0.1%.

d



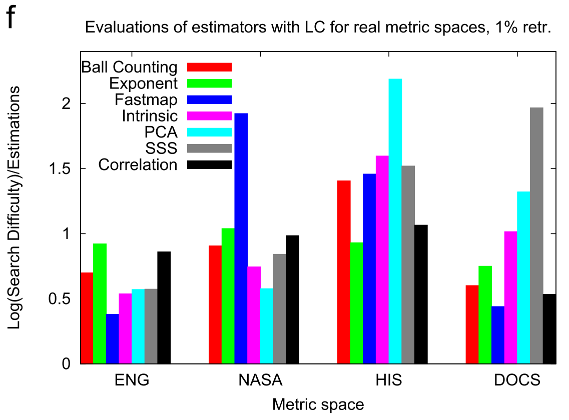
Retrieving 0.1%.

e



Retrieving 1%.

f



Retrieving 1%.

Fig. 6. Comparison of Ball Counting, Distance Exponent, Fastmap, Intrinsic Search Difficulty, PCA, SSS, and Correlation IDim estimators, for real metric spaces. On the left, using Pivots. On the right, using List of Clusters.

As on the synthetic spaces, Distance Exponent and Correlation turn out to be the best predictors for the four tested indices, as the ratio between log(Search Difficulty) and the IDim estimation remains similar in all the real world spaces tested considering the different query radii and indices. This can be corroborated in Table 3: Distance Exponent and Correlation have the lowest standard deviations, just 0.135 and 0.200, respectively. As it can be observed, the other methods are not stable enough with respect to search costs. In fact, their standard deviations

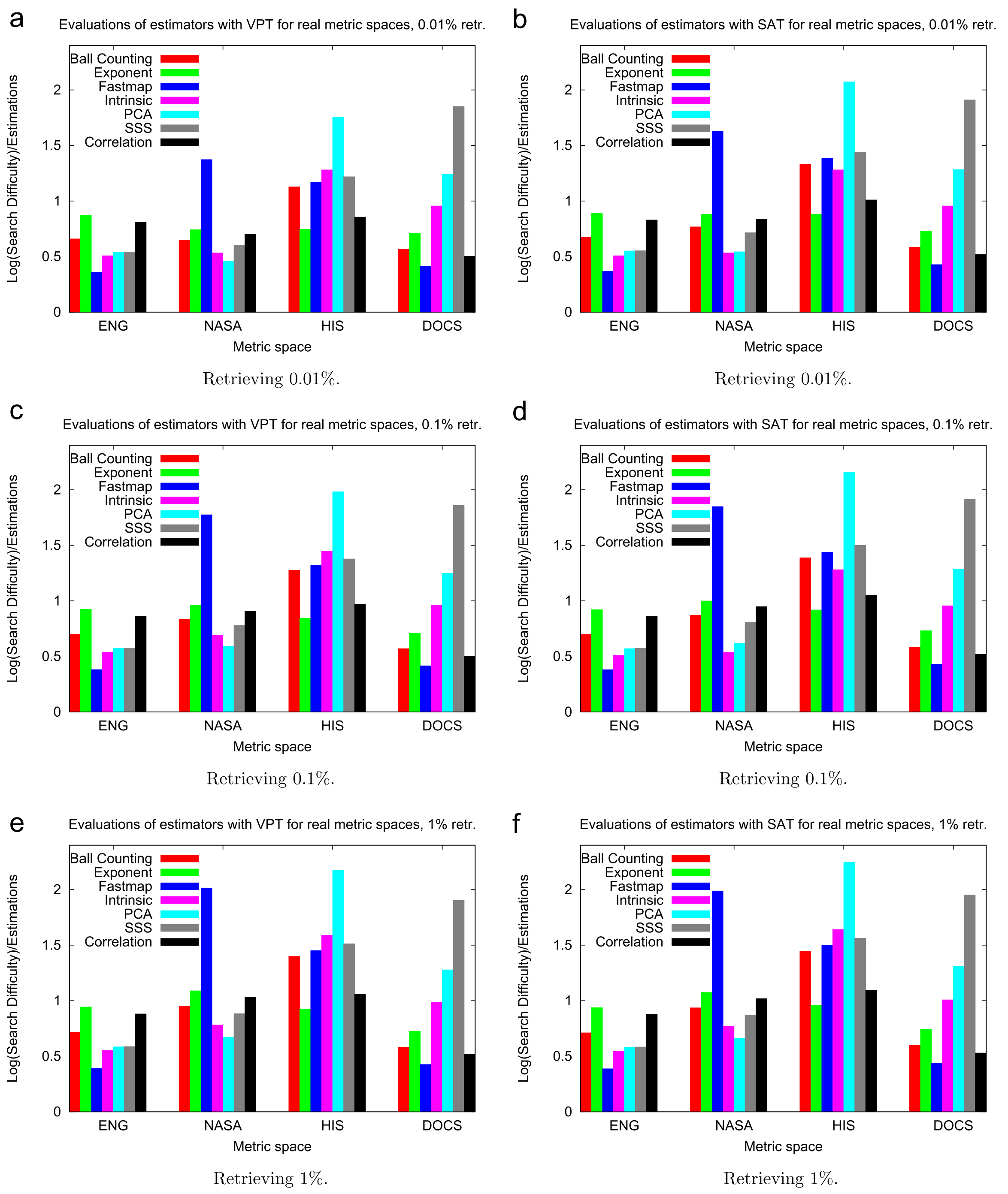


Fig. 7. Comparison of Ball Counting, Distance Exponent, Fastmap, Intrinsic Search Difficulty, PCA, SSS, and Correlation IDim estimators, for real metric spaces. On the left, using Vantage Point Tree. On the right, using Spatial Approximation Tree.

range from 0.289 for Ball Counting to 0.605 for PCA. This is because in some cases they underestimate the search difficulty, and in the others they make an overestimation.

The search cost, however, depends both on the database size and on the output size (which in turn depends on the query radius). Therefore, dividing the cost of the search by these measures may give a more stable measure of search difficulty. We repeat this evaluation considering the relation between the fraction of the database visited when solving a query and the estimation of IDim (this is, $-\log(\text{Search Difficulty}/\text{DBS})/IDim$, where DBS stands for database size) and, on the other hand, the number of distance evaluation per each object in the query answer set (this is, $\log(\text{Search Difficulty}/\text{QOS})/IDim$, where QOS stands for query output size). We summarize these results in Tables 4 and 5. The reduced standard deviations confirm

Table 3
Comparison of log(Search Difficulty)/Estimation for the seven IDim estimators.

IDim estimator	Mean	Standard deviation
Ball Counting	0.810	0.289
Distance Exponent	0.822	0.135
Fastmap	0.924	0.592
Intrinsic Search Difficulty	0.861	0.364
PCA	1.068	0.605
SSS	1.121	0.556
Correlation	0.773	0.200

Table 4
Comparison of log(Search Difficulty/DBS)/Estimation for the seven IDim estimators.

IDim estimator	Mean	Standard deviation
Ball Counting	0.218	0.175
Distance Exponent	0.212	0.149
Fastmap	0.287	0.302
Intrinsic Search Difficulty	0.255	0.196
PCA	0.279	0.247
SSS	0.283	0.186
Correlation	0.204	0.156

Table 5
Comparison of log(Search Difficulty/QOS)/Estimation for the seven IDim estimators.

IDim estimator	Mean	Standard deviation
Ball Counting	0.427	0.173
Distance Exponent	0.448	0.164
Fastmap	0.398	0.281
Intrinsic Search Difficulty	0.491	0.280
PCA	0.599	0.406
SSS	0.681	0.553
Correlation	0.407	0.137

that the estimations are indeed more stable, especially when dividing by the database size. Still, we obtain the same conclusions: the two most stable intrinsic dimensionality estimators are *Distance Exponent* and *Correlation*.

5. Conclusions

The *Intrinsic Dimension* (IDim) of metric spaces measures their search difficulty, independent of the type of index used. Computing the IDim is useful to determine whether a metric space can be indexed at all (or we must resort to sequential scanning or approximate methods), which kind of index would perform better, and what search performance to expect.

We have analyzed seven IDim estimators in a practical perspective. Some were defined for *D*-dimensional coordinate spaces, and we have adapted them to the more general metric spaces. We compared their predictions with the actual search cost using various synthetic and real-life metric spaces, so as to verify which are better at predicting the search difficulty.

Although our results are preliminary, they suggest that all the methods considered obtain appropriate estimations over synthetic metric spaces, because their values grow as the dimension increases. However, if we compare the estimations with the real search costs, the Distance Exponent [12,13] and Correlation [14] turn out to be more stable. This is corroborated in Table 3 that shows the mean and standard deviation of the ratio between log(Search Difficulty) and the IDim estimation for the seven estimators. The standard deviations computed for Distance Exponent and Correlation are just 0.135 and 0.200, respectively, and are the two lowest ones. On the other hand, the other estimators, namely, Ball Counting (our adaptation of Box Counting [21]), Fastmap [28], the simple measure proposed by Chávez et al. [1], PCA, and SSS [37] sometimes obtain values less than the logarithm of search costs and other times greater than them. These conclusions are also supported by Tables 4 and 5.

As future work, we plan to analyze other estimators. For instance, we can study the concentration dimension [35].

Acknowledgments

We gratefully acknowledge the anonymous referees who helped us to improve the presentation.

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, J. Marroquín, Searching in metric spaces, ACM Comput. Surv. 33 (3) (2001) 273–321.
[2] G. Hjaltason, H. Samet, Index-driven similarity search in metric spaces, ACM Trans. Database Syst. 28 (4) (2003) 517–580.
[3] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search: The Metric Space Approach, Advances in Database Systems, vol. 32, Springer, New York, NY, USA, 2006.
[4] H. Samet, Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling), Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
[5] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: Proceedings of 23rd Conference on Very Large Databases (VLDB), 1997, pp. 426–435.
[6] V. Dohnal, C. Gennaro, P. Savino, P. Zezula, D-index: distance searching index for metric data sets, Multimed. Tools Appl. 21 (1) (2003) 9–33.
[7] T. Skopal, J. Pokorný, V. Snásel, PM-tree: pivoting metric tree for similarity search in multimedia databases, in: ADBIS (Local Proceedings), 2004.
[8] G. Navarro, N. Reyes, Dynamic spatial approximation trees for massive data, in: T. Skopal, P. Zezula (Eds.), Proceedings of 2nd International Workshop on Similarity Search and Applications (SISAP), IEEE CS Press, Prague, Czech Republic, 2009, pp. 81–88.
[9] G. Navarro, R. Uribe, Fully dynamic metric access methods based on hyperplane partitioning, Inf. Syst. 36 (4) (2011) 734–747.
[10] G. Navarro, N. Reyes, Dynamic list of clusters in secondary memory, in: Proceedings of 7th International Workshop on Similarity Search and Applications (SISAP), Lecture Notes in Computer Science, vol. 8821, 2014, pp. 94–105.
[11] E. Chávez, G. Navarro, A compact space decomposition for effective metric indexing, Pattern Recognit. Lett. 26 (9) (2005) 1363–1376.
[12] C. Traina Jr., A. J. M. Traina, C. Faloutsos, Distance Exponent: A New Concept for Selectivity Estimation in Metric Trees, Research Paper 99–110, School of Computer Science, Carnegie Mellon University (03/1999 1999).
[13] C. Traina Jr., A.J.M. Traina, C. Faloutsos, Distance exponent: A new concept for selectivity estimation in metric trees, in: Proceedings of

16th International Conference on Data Engineering (ICDE), 2000, p. 195.
[14] F. Camastra, A. Vinciarelli, Estimating the intrinsic dimension of data with a fractal-based method, IEEE Trans. Pattern Anal. Mach. Intell. 24 (10) (2002) 1404–1407.
[15] C. Bustos, G. Navarro, N. Reyes, R. Paredes, An empirical evaluation of intrinsic dimension estimators, in: Proceedings of 8th International Conference on Similarity Search and Applications (SISAP), Lecture Notes in Computer Science, vol. 9371, Springer, Glasgow, Scotland, UK, 2015, pp. 125–137.
[16] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
[17] F. Camastra, Data dimensionality estimation methods: a survey, Pattern Recognit. 36 (12) (2003) 2945–2954.
[18] K. Fukunaga, Introduction to Statistical Pattern Recognition, 2nd ed., Academic Press Professional, Inc. San Diego, CA, USA, 1990.
[19] R. Bellman, Adaptive Control Processes: A Guided Tour, Princeton University Press, Princeton, NJ, 1961.
[20] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag New York, Inc., New York, NY, USA, 1995.
[21] B. Mandelbrot, Fractals: Form, Chance and Dimension, W. H. Freeman, San Francisco, 1977.
[22] J.P. Eckmann, D. Ruelle, Ergodic theory of chaos and strange attractors, Rev. Mod. Phys. 57 (1985) 617.
[23] E. Ott, Chaos in Dynamical Systems, Cambridge University Press, Cambridge, New York, 1993.
[24] D. Kaplan, L. Glass, Understanding Nonlinear Dynamics, Springer-Verlag, New York, 1995.
[25] K. Figueroa, G. Navarro, E. Chávez, Metric Spaces Library, Available at http://www.sisap.org/Metric_Space_Library.html, 2007.
[26] H.V. Jagadish, A retrieval technique for similar shapes, in: SIGMOD Conference, ACM Press, Denver, CO, USA, 1991, pp. 208–217.
[27] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Sov. Phys. Dokl. 10 (8) (1966) 707–710.
[28] C. Faloutsos, K.-I. Lin, Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets, in: Proceedings of 1995 ACM SIGMOD International Conference on Management of Data, ACM Press, San Jose, CA, USA, 1995, pp. 163–174.
[29] I.T. Jolliffe, Principal Component Analysis, 2nd ed., Springer Series in Statistics, Springer, New York, NY, USA, 2002.
[30] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2013.
[31] S. Brin, Near neighbor search in large metric spaces, in: Proceedings of 21st Conference on Very Large Databases (VLDB'95), 1995, pp. 574–584.
[32] E. Chávez, J. Marroquín, Proximity queries in metric spaces, in: Proceedings of 4th South American Workshop on String Processing (WSP'97), Carleton University Press, Valparaíso, Chile, 1997, pp. 21–36.
[33] P. Ciaccia, M. Patella, P. Zezula, A cost model for similarity queries in metric spaces., in: Proceedings of 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 1998, pp. 59–68.
[34] P. Yianilos, Excluded Middle Vantage Point Forests for Nearest Neighbor Search, Technical report, NEC Research Institute, Baltimore, MD, in: 6th DIMACS Implementation Challenge: Near Neighbor Searches Workshop, ALENEX'99, 1998.
[35] V. Pestov, Intrinsic dimension of a dataset: what properties does one expect? in: 2007 International Joint Conference on Neural Networks (IJCNN), 2007, pp. 2959–2964. http://dx.doi.org/10.1109/IJCNN.2007.4371431.
[36] V. Pestov, An axiomatic approach to intrinsic dimension of a dataset, Neural Netw 21 (2–3) (2008) 204–213, http://dx.doi.org/10.1016/j.neunet.2007.12.030. in: Advances in Neural Networks Research: 2007 International Joint Conference on Neural Networks (IJCNN)..
[37] N. R. Brisaboa, A. Fariña, O. Pedreira, N. Reyes, Similarity search using sparse pivots for efficient multimedia information retrieval, in: 8th IEEE International Symposium on Multimedia (ISM), IEEE CS, San Jose, CA, USA, 2006, pp. 881–888.
[38] P. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, in: Proceedings of 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93), SIAM Press, Austin, TX, USA, 1993, pp. 311–321.
[39] T. Chiueh, Content-based image indexing, in: Proceedings of 20th Conference on Very Large Databases (VLDB'94), 1994, pp. 582–593.
[40] G. Navarro, Searching in metric spaces by spatial approximation, Very Large Databases J. 11 (1) (2002) 28–46.