Multimedia Tools and Applications (2022) 81:30465–30491
https://doi.org/10.1007/s11042-022-12690-w

A meta-indexing method for fast probably approximately correct nearest neighbor searches

Simone Santini[1]

Received: 23 October 2020 / Revised: 16 February 2021 / Accepted: 21 February 2022 /
Published online: 6 April 2022

Abstract
In this paper we present an indexing method for probably approximately correct nearest neighbor queries in high dimensional spaces capable of improving the performance of any index whose performance degrades with the increased dimensionality of the query space. The basic idea of the method is quite simple: we use SVD to concentrate the variance of the inter-element distance in a lower dimensional space, Ξ. We do a nearest neighbor query in this space and then we "peek" forward from the nearest neighbor by gathering all the elements whose distance from the query is less than $d_\Xi(1 + \zeta\sigma_\Xi^2)$, where d_Ξ is the distance from the nearest neighbor in Ξ, σ_Ξ^2 is the variance of the data in Ξ, and ζ a parameter. All the data thus collected form a tentative set T, in which we do a scan using the complete feature space to find the point closest to the query. The advantages of the method are that (1) it can be built on top of virtually any indexing method and (2) we can build a model of the distribution of the error precise enough to allow designing a compromise between error and speed. We show the improvement that we can obtain using data from the SUN data base.

Keywords Indexing · Approximate nearest neighbor · Error modeling · Curse of dimensionality · Multimedia data base · Approximate search

1 Introduction and background

Indexing sets of points in high-dimensional spaces in sub-linear time is one of the long-standing open problems in multimedia databases and multimedia information retrieval. In its most general form, the problem is unsolved and, quite possibly, unsolvable. While a formal proof exists only for specific models such as pivot-based indices [53], one could paraphrase Charles A. Rogers by saying that "most mathematicians believe and all engineers know" that in high dimensional feature spaces no indexing scheme is significantly better

✉ Simone Santini
simone.santini@uam.es

[1] Escuela Politécnica Superior, Universidad Autónoma de Madrid, C/ Tomas y Valiente 11, 28049 Madrid, Spain

than linear scan when we try to work the most common types of queries: *nearest neighbor* and *range*. [1]

If we don't know how to index efficiently is not for lack of ingenuity in devising indexing schemes but, frustratingly, because of the characteristics of the spaces in which we execute the search.

In this paper, we present an efficient method for probably approximately correct nearest-neighbor determination in feature spaces of medium to high dimensionality. The algorithm can be implemented on top of any indexing method with sub-linear search time in low-dimensional feature spaces. The efficiency of the method depends on characteristics of the variance distribution of the features that we verify experimentally to be satisfied by most features of pracical interet.

Consider the following general setting. We have a data base D composed of elements in a space Ω to which the query also belongs. The domain is a metric space, equipped with a distance function ρ. We also assume that Ω is also equipped with a Borel measure μ such that $\mu(\Omega) = 1$, and that the data set D is drawn uniformly with respect to μ [52]. The triple (Ω, ρ, μ) is known as a space with metric and measure (mm-space). Note that, in practical applications, it is often the data base D that determines μ: μ is whatever measure makes D uniformly distributed. Also note that the set D itself can be seen as a discrete mm-space with distance function $\rho_{|D}$ and the counting metric $\mu(A) = |A|/|D|$ for $A \subseteq D$.

One important characteristics of an mm-space is its *contraction function* α. The idea is the following. Take a set $A \subseteq \Omega$ with $\mu(A) = 1/2$ and make A a bit bigger by taking elements at a distance at most ϵ from A. How much of Ω is taken up by the new set, and how much of Ω is left untouched? That is, for $A \subseteq \Omega$, let

$$A_\epsilon = \left\{ x \ : \ \inf_{a \in A} \{\rho(x, a)\} \leq \epsilon \right\} \tag{1}$$

with $A_0 = A$; then

$$\alpha(\epsilon) = 1 - \inf \left\{ \mu(A_\epsilon) \ : \ A \subseteq \Omega, \mu(A) \geq \frac{1}{2} \right\} \tag{2}$$

That is, $\alpha(\epsilon)$ is the largest fraction of the space (according to the measure μ) that is left out after we take a set of measure at least $1/2$ and grow it by ϵ. Note that $\alpha(0) = 1/2$.

Let Ω_n be a family of mm-spaces indexed by an integer parameter n. The spaces form a *Lévy family* if for all $\epsilon > 0$, $\lim_{n \to \infty} \alpha_n(\epsilon) = 0$ [37].

Many of the headaches in the design of a good multimedia indexing algorithm derive from the fact that most classes of feature spaces used in practical applications are, indeed, Lévy families. For example, $(\mathbb{R}^n, \rho, \mu_n)$, where ρ is any Minkowski distance and μ_n is the Gaussian measure with $\sigma = 1/\sqrt{n}$ is a Lévy family, and so are unitary spheres in $\mathbb{R}^n$ with the uniform measure [29, 40]. In both cases, if $\alpha_n(\epsilon)$ is the contraction function of the nth member of the family, one can show that there are $c_i, c_2 > 0$ such that

$$\alpha_n(\epsilon) \leq c_1 \exp(-c_2 n \epsilon^2) \tag{3}$$

[1] The original remark by Charles A. Rogers of University College, London. It refers to a conjecture on the optimal packing density of spheres in three dimensions set forth by J. Kepler in 1611 and finally proved in 1998. In the original, the people who "knew" where not the engineers but the physicists.

The contraction function imposes constraints on how "accurate" the functions are that we can use to "sample" the behavior of points in Ω. A function $f : \Omega \to \mathbb{R}$ is 1-Lipschitz ($f \in \text{Lip}_1(\Omega)$) if, for all $x, y \in \Omega$

$$|f(x) - f(y)| \leq \rho(x, y) \tag{4}$$

(see [22]). One can prove the following property:

Theorem 1 ([46]) *Let $f \in Lip_1(\Omega)$ and let M be its median, that is, the value such that*

$$\mu\{\omega : f(\omega) \leq M\} \geq \frac{1}{2} \quad \mu\{\omega : f(\omega) \geq M\} \geq \frac{1}{2} \tag{5}$$

Then, for all $\epsilon > 0$,

$$\mu\{\omega : |f(\omega) - M| > \epsilon\} < 2\alpha(\epsilon) \tag{6}$$

That is: in a highly concentrated space, all $\text{Lip}_1(\Omega)$ functions concentrate around their median.

One important observation, one which opens up a way of mitigating the effects of concentration, is that often the actual dimensionality of the data is much lower than that of the space in which the data are represented. That is, many a data sets in, for example, $\mathbb{R}^n$ are actually on—or close to—a manifold M of dimensionality $m \ll n$.

This phenomenon is captured by various measures of intrinsic dimensionality; one of the best known is due to Chavez et al. [18] and is defined as

$$d_I(\Omega) = \frac{m(\rho)}{2\sigma^2(\rho)} \tag{7}$$

where $m(\rho)$ is the mean value of the distance function ρ on Ω with respect to the measure μ and $\sigma^2(\rho)$ is its variance.

The intrinsic dimension may be orders of magnitude smaller than the space in which the features are defined. This observation opens a possibility to alleviate the dimensionality problem, at least if we accept only probably correct results: it is possible, in principle, to approximate the distance between two points of the space using the geodesic distance in a manifold of much smaller dimensionality. There are, however, two problems.

i) Although d_I gives a general indication of the dimensionality of a manifold that approximates the distribution of a data base D, it gives no indication on how to identify the manifold, nor gives it its exact dimensionality (it usually underestimates it). Moreover, the numerical characterization of a manifold that contains D points in $\mathbb{R}^N$ may require up to $O(DN)$ parameters, thus making an index based on it not competitive vis-à-vis sequential scanning, unless the manifold admits a compact representation.
ii) Answering a query using the distance in a sub-manifold results in an approximation and, therefore, a probability of error. Any algorithm that uses this kind of approximation should allow an estimation of the error probability, and it should provide means to control it, in a trade-off between efficiency and error probability.

The first problem is often circumvented by restricting the manifold to be a linear subspace of the feature space. One can use techniques such as singular value decomposition [25] to find sub-spaces of reduced dimensionality in which most of the variance of the data is concentrated. Computing distances in these subspaces is an approximation of the actual distance, so a reduced-dimensionality space can be used to find approximate solutions to a query avoiding searches in high dimensional spaces. Note that in this case the linear subspace will in general be of dimensionality much higher that d_I, since the lowest-dimensional

manifold is in general non-linear and the linear search finds a linear subspace that contains the manifold.

The algorithm that we present in this paper is a simple variation of this strategy that allows us to solve the second problem and to fine-tune the trade-off between error and performance.

After the presentation of the algorithm in Section 2, the paper is divided in two main parts: a theoretical one (Sections 3, 4, and 5), and an experimental one (Sections 6 and 7). In Section 3 we determine the theoretical error probability of the algorithm, that is, the probability that the algorithm returned the wrong nearest neighbor as a function of the algorithm parameters. In Section 4 we calculate the expected value of the distance error, that is, of the difference between the distance from the query of the item returned by the algorithm and that of the true nearest neighbor of the query, again as a function of the algorithm parameters. In Section 5 we develop a cost model for the algorithm.

The experimental part begins in Section 6 with validation: we use the SUN data set [63] and various features to determine experimentally the magnitudes calculated theoretically in Sections 3 and 4, and show that these are a good fit with the actual behavior of the algorithm. In Section 7 we determine experimentally the cost of indexing as a function of the algorithm parameters, both in terms of number of computations as well as in the number of block accesses. In the same section we compare the performance of the algorithm with that of other approximate indexing methods, and carry out an analysis of the differences. Section 8 places this paper in the context of related work, and Section 9 draws some conclusions. Details of the experimental methodology and set-up are given in Appendices A and B.

2 Approximate nearest neighbor

The starting point of our method is the idea that we introduced at the end of the previous section, namely, to identify a sub-space in which most of the distance is concentrated, and to use the distance in this sub-space as a basis for the approximate determination of the nearest neighbor.

Let (Ω, ρ, μ) be a mm-space, where $\Omega \sim \mathbb{R}^N$ and ρ is the Euclidean distance on $\Omega \times \Omega$. Let $\mathbf{D} \in \mathbb{R}^{N \times D}$ be the given data base, with $\mathbf{D} = [d_1 | \cdots | d_D]$; $d_i \in \Omega$ being the ith element in the data base. Assume that μ is a measure such that the d_i are uniformly distributed with respect to it.

We decompose the matrix $\mathbf{D}$ using singular value decomposition as

$$\mathbf{D} = U \Sigma V' \tag{8}$$

where $V \in \mathbb{R}^{D \times N}$, $\Sigma = \text{diag}(\lambda_1, \ldots, \lambda_N)$, with $\lambda_i \geq \lambda_{i+1} > 0$ and where $U \in \mathbb{R}^{N \times N}$ is an orthogonal matrix whose columns are a basis of $\mathbb{R}^N$,

$$U = [u_1 | \cdots | u_N]. \tag{9}$$

It can be shown [25] that the transformation $y = U'x$ is a rotation of the space such that the variance of the data base $\mathbf{D}$ in the transformed space has the property:

$$\text{var}\left(u_i' \mathbf{D}\right) \geq \text{var}\left(u_{i+1}' \mathbf{D}\right) \tag{10}$$

Based on this observation, select a dimensionality $M < N$ and define the matrices

$$\begin{aligned} \mathbf{M}_M &= [u_1 | \cdots | u_M]' \\ \mathbf{N}_M &= [u_{M+1} | \cdots | u_N]' \end{aligned} \tag{11}$$

With these matrices we decompose $\Omega \equiv \Xi \times \Theta$ with

$$\begin{aligned} \Xi &= \text{span}(\mathbf{M}_M) \sim \mathbb{R}^M \\ \Theta &= \text{span}(\mathbf{N}_M) \sim \mathbb{R}^{N-M} \end{aligned} \tag{12}$$

Similarly, we can define the restrictions to Ξ and Θ of ρ: ρ_Ξ and ρ_Θ, respectively. Given an element $x \in \Omega$, we can represent it as the pair (y, z) with $y = \mathbf{M}_M x$ and $z = \mathbf{N}_M x$. The decomposition is such that the covariances of $\mathbf{D}$ (and of any stochastic variable uniform in μ) under these transformations are approximately diagonal:

$$\begin{aligned} \Sigma_\Xi &= \text{cov}(\mathbf{M}_m \mathbf{D}) \approx \text{diag}\left(\sigma^2_{\Xi,1}, \ldots, \sigma^2_{\Xi,M}\right) \\ \Sigma_\Theta &= \text{cov}(\mathbf{N}_m \mathbf{D}) \approx \text{diag}\left(\sigma^2_{\Theta,1}, \ldots, \sigma^2_{\Theta,N-M}\right) \end{aligned} \tag{13}$$

with

$$\sigma^2_{\Xi,i} \geq \sigma^2_{\Xi,i+1} \geq \sigma^2_{\Theta,j} \geq \sigma^2_{\Theta,j+1} \tag{14}$$

Define

$$\sigma^2_\Xi = \sum_{i=1}^{m} \sigma^2_{\Xi,i}, \quad \sigma^2_\Theta = \sum_{i=1}^{N-m} \sigma^2_{\Theta,i} \tag{15}$$

and

$$\nu = \frac{\sigma^2_\Xi}{\sigma^2_\Theta} \tag{16}$$

The parameter ν is a measure of how much variance is contained in the space Ξ. We shall always choose M so that $\nu > 1$. Our algorithm is then as follows:

i) Given a query point q, find its nearest neighbor in Ξ; let ρ_0^2 be the square of the distance from q to its nearest neighbor in Ξ;

ii) find all the points whose distance from q in Ξ is less than $\sqrt{\rho_0^2 + \alpha}$, where α is a suitable value that we shall discuss in the following; let T be the set of such points;

iii) scan the set T and, among the elements of T, find the nearest point to q in the complete feature space Ω.

More formally, we can describe the algorithm as in Fig. 1.

We assume that M is small enough that the "arg min" in line 2 and the range query in line 4 can profitably be done using a fast indexing method, while the general feature space has dimensionality such that the "arg min" in line 5 has to be done by scanning all the elements of the set T. Note also the use of ρ^2 in line 5: if we assume a Euclidean distance, computing ρ^2 is cheaper than computing ρ and does not change the result. On the other

```
nn(q, D, D')        (q ∈ Ω, D ∈ R^{N×D}, D' = M_M D)
1.  q' ← M_M q
2.  n ← arg min_{ξ∈D'} {ρ_Ξ(q', ξ)}
3.  u ← ρ_Ξ(q', n)
4.  T ← {x : x ∈ D, ρ_Ξ(q', x) < √(u² + α)}
5.  r ← arg min_{x∈T} {ρ²_Ξ(q, x)}
6.  return r
```

Fig. 1 The approximate nearest neighbor algorithm. Here, (Ξ, ρ_Ξ, μ_Ξ) is a nn-subspace of (Ω, ρ, μ) of dimensionality M. $\mathbf{D}'$ is the projection of the data base $\mathbf{D}$ on Ξ (computed offline). For the use of the distance in steps 2 and 4 and its square in steps 4 and 5, see the text

hand, indexing algorithms often rely on the triangle inequality, thus requiring in line 2 the use of the distance rather than its square. In the case of a different Minkowski distance L_p, we would use ρ^p in line 5; in the case of more complex distances, we can use any monotonic function of the distance that makes its computation cheaper.

The two important parameters of the algorithm are $M = \dim(\Xi)$ and α (the extra distance at which we search in step 4). These parameters determine the execution time and the probability of finding the actual nearest neighbor.

3 Error probability

The algorithm of the previous section finds the true nearest neighbor only if it belongs to the set T, that is, if its distance from the query in Ξ does not exceed the distance of the closest projection in Ξ by more than α. In this section, we develop an approximate expression for the probability that this doesn't happen, that is, for the probability that the algorithm return the wrong nearest neighbor. This expression is derived under considerable simplifying assumptions but, as we shall see in Section 6, it is a good fit to the data and is therefore very useful as a convenient design tool to determine the values of M and α.

Assume that ρ is Euclidean, μ is Gaussian, and that the query q is drawn uniformly from μ. Step 2 of the algorithm finds the nearest neighbor in Ξ—call it n—; its actual distance from the query is

$$\rho^2(q,n) = \rho^2_\Xi(q,n) + \rho^2_\Theta(q,n) \tag{17}$$

Suppose that the algorithm gives the wrong answer. The real nearest neighbor of q is item t, with a distance from q

$$\rho^2(q,t) = \rho^2_\Xi(q,t) + \rho^2_\Theta(q,t) < \rho^2(q,n) \tag{18}$$

Had t been in the set T that we build at step 4, the algorithm would have returned it as a solution. Since we are assuming that we get the wrong answer, it must be $t \notin T$, or

$$\rho^2_\Xi(q,t) > \rho^2_\Xi(q,n) + \alpha \tag{19}$$

Set

$$\begin{aligned} \Delta_\Xi &\triangleq \rho^2_\Xi(q,t) - \rho^2_\Xi(q,n) \\ \Delta_\Theta &\triangleq \rho^2_\Theta(q,t) - \rho^2_\Theta(q,n). \end{aligned} \tag{20}$$

With these definitions, we can write the probability of (18) conditioned on (19) as

$$\begin{aligned} \mathbb{P}_E &= \mathbb{P}\left\{ \rho^2(q,t) < \rho^2(q,n) \,\middle|\, \rho^2_\Xi(q,t) > \rho^2_\Xi(q,n) + \alpha \right\} \\ &= \mathbb{P}\left\{ \rho^2_\Xi(q,t) - \rho^2_\Xi(q,n) < \rho^2_\Theta(q,n) - \rho^2_\Theta(q,t) \,\middle|\, \rho^2_\Xi(q,t) > \rho^2_\Xi(q,n) + \alpha \right\} \\ &= \mathbb{P}\{\Delta_\Xi < -\Delta_\Theta \,|\, \Delta_\Xi > \alpha\} \\ &= \frac{\mathbb{P}\{\Delta_\Xi < -\Delta_\Theta, \Delta_\Xi > \alpha\}}{\mathbb{P}\{\Delta_\Xi > \alpha\}} \\ &\triangleq \frac{P_1}{P_2} \end{aligned} \tag{21}$$

If μ is Gaussian and the elements of the data base as well as the query are uniform in μ, then ρ^2_Ξ and ρ^2_Θ have χ^2 distributions with a number of degrees of freedom that depends on the dimensionality of the respective sub-space [34]. Here, we'll do our most important approximation: since the variance of the data is predominantly on the first axes of Ξ and Θ,

we shall assume that we can adequately approximate the distribution with just two degrees of freedom, that is, that ρ^2_Ξ and ρ^2_Θ have exponential distribution:

$$\begin{aligned} p_\Xi(u) &= \frac{1}{2\sigma^2_\Xi} \exp\left(-\frac{u}{2\sigma^2_\Xi}\right) \\ p_\Theta(u) &= \frac{1}{2\sigma^2_\Theta} \exp\left(-\frac{u}{2\sigma^2_\Theta}\right) \end{aligned} \tag{22}$$

The difference between squares of distances will then have a Laplace distribution:

$$\begin{aligned} \Delta_\Xi(u) &= \frac{1}{4\sigma^2_\Xi} \exp\left(-\frac{|u|}{2\sigma^2_\Xi}\right) \\ \Delta_\Theta(u) &= \frac{1}{4\sigma^2_\Theta} \exp\left(-\frac{|u|}{2\sigma^2_\Theta}\right) \end{aligned} \tag{23}$$

Under these hypotheses, with reference to the quantities P_1 and P_2 defined in (21), we have

$$\begin{aligned} P_2 &= \int_\alpha^\infty \Delta_\Xi(u)\, du = \frac{1}{4\sigma^2_\Xi} \int_\alpha^\infty \exp\left(-\frac{u}{2\sigma^2_\Xi}\right) du \\ &= \frac{1}{2} \exp\left(-\frac{\alpha}{2\sigma^2_\Xi}\right) \\ &\stackrel{\triangle}{=} \frac{1}{2} \exp\left(-\frac{\zeta}{2}\right) \end{aligned} \tag{24}$$

Here we have defined $\zeta \stackrel{\triangle}{=} \alpha/\sigma^2_\Xi$: ζ is the excess distance at step 4 measured in units of the variance of the data in the subspace Ξ. We can consider it as a distance measured in the "natural" units for the distribution. The value P_1 can be determined as:

$$\begin{aligned} P_1 &= \int_\alpha^\infty \Delta_\Xi(u) \int_{-\infty}^{-u} \Delta_\Theta(v)\, dv\, du \\ &\stackrel{\dagger}{=} \int_\alpha^\infty \Delta_\Xi(u) \int_u^\infty \Delta_\Theta(v)\, dv\, du \\ &= \frac{1}{16\sigma^2_\Xi\sigma^2_\Theta} \int_\alpha^\infty \exp\left(-\frac{u}{2\sigma^2_\Xi}\right) \int_u^\infty \exp\left(-\frac{v}{2\sigma^2_\Theta}\right) dv\, du \\ &= \frac{1}{2} \frac{\sigma^2_\Theta}{\sigma^2_\Xi + \sigma^2_\Theta} \exp\left(-\frac{1}{2}\frac{\sigma^2_\Xi + \sigma^2_\Theta}{\sigma^2_\Theta}\frac{\alpha}{\sigma^2_\Xi}\right) \\ &= \frac{1}{2(1+\nu)} \exp\left(-\frac{1+\nu}{2}\zeta\right) \end{aligned} \tag{25}$$

where the equality † is due to the symmetry of Δ_Θ. Putting this together we get

$$P_E = \frac{P_1}{P_2} = \frac{1}{1+\nu} \exp\left(-\frac{\nu}{2}\zeta\right) \tag{26}$$

Consequently, the probability of finding the true nearest neighbor is $1 - P_E$. These values depend on two parameters: ν and ζ. The first parameter depends on the characteristics of the features and on the size M that we choose for the reduced space Ξ in which we make the initial search; the second is the amount by which we "peek" beyond the distance of the

nearest neighbor in Ξ expressed in units of σ_Ξ. In many design situations, we might be interested in doing the opposite: given a desired error probability p, we want to determine the corresponding ζ:

$$\zeta = \frac{2}{\nu} \log \frac{1}{(\nu+1)p} \tag{27}$$

4 Average error

Due to the approximation that we are making, with probability P_E we do not find the true nearest neighbor t, but an approximate one n, with $\rho^2(q,n) > \rho^2(q,t)$. This approximation is the more acceptable the closer the item n is to the nearest neighbor, that is, the smaller the difference $\rho^2(q,n) - \rho^2(q,t)$. In this section we shall determine the probability density of $\rho^2(q,n) - \rho^2(q,t)$ and its expected value.

Let $p(\xi)$ be the probability density of the distance error. With probability $1 - P_E$, we find the correct nearest neighbor therefore, with probability $1 - P_E$, $\xi = 0$. With probability P_E we get the wrong nearest neighbor, therefore for $\xi > 0$ we have

$$p(\xi) = P_E \pi(\xi) \tag{28}$$

Abusing the notation a bit, if X is a continuous random variable, we write $X \sim x$ if $X \in [x, x + dx]$. With this notation, we have

$$\begin{aligned} \pi(\xi) &= \mathbb{P}\left\{ \rho^2(q,n) - \rho^2(q,t) \sim \xi \,\middle|\, \rho^2_\Xi(q,n) - \rho^2_\Xi(q,t) > \zeta \right\} \\ &= \mathbb{P}\{\Delta_\Xi + \Delta_\Theta \sim \xi \,|\Delta_\Xi > \zeta\} \\ &= \frac{\mathbb{P}\{\Delta_\Xi + \Delta_\Theta \sim \xi, \Delta_\Xi > \zeta\}}{\mathbb{P}\{\Delta_\Xi > \zeta\}} \end{aligned} \tag{29}$$

Note that we are working in the ξ space, in which

$$\begin{aligned} \Delta_\Xi(\xi) &\sim \frac{1}{4} \exp\left(-\frac{|\xi|}{2}\right) \\ \Delta_\Theta(\xi) &\sim \frac{\nu}{4} \exp\left(-\nu\frac{|\xi|}{2}\right) \end{aligned} \tag{30}$$

In this space

$$\mathbb{P}\{\Delta_\Xi > \zeta\} = \frac{1}{4}\int_\zeta^\infty \exp\left(-\frac{\xi}{2}\right) d\xi = \frac{1}{2}\exp\left(-\frac{\zeta}{2}\right) \tag{31}$$

while

$$\begin{aligned} \mathbb{P}\{\Delta_\Xi + \Delta_\Theta \sim \xi, \Delta_\Theta > \zeta\} &= \int_\zeta^\infty \Delta_\Xi(u)\Delta_\Theta(\xi - u)\, du \\ &= \frac{\nu}{16}\int_\zeta^\infty \exp\left(-\frac{|u|}{2}\right)\exp\left(-\frac{\nu}{2}|\xi - u|\right) du \end{aligned} \tag{32}$$

We have to consider two cases:

i) $\xi < \zeta$, that is, $\xi - u < 0$ in the range of integration

$$\begin{aligned} \mathbb{P}\{\Delta_\Xi + \Delta_\Theta \sim \xi, \Delta_\Theta > \zeta\} &= \frac{\nu}{16}\int_\zeta^\infty \exp\left(-\frac{u}{2}\right)\exp\left(-\frac{\nu}{2}(u - \xi)\right) du \\ &= \frac{1}{8}\frac{\nu}{\nu+1}\exp\left(-\frac{\zeta}{2}\right)\exp\left(\frac{\nu}{2}(\xi - \zeta)\right) \end{aligned} \tag{33}$$

ii) $\xi > \zeta$, which entails that we have to divide the interval of integration in $[\zeta, \xi]$, and $[\xi, \infty]$:

$$\begin{aligned}\mathbb{P}\{\Delta_\Xi + \Delta_\Theta \sim \xi, \Delta_\Theta > \zeta\} &= \frac{\nu}{16}\left[\int_\zeta^\xi \exp\left(-\frac{u}{2}\right)\exp\left(-\frac{\nu}{2}(\xi-u)\right)\,du\right.\\ &\quad\left.+\int_\xi^\infty \exp\left(-\frac{u}{2}\right)\exp\left(-\frac{\nu}{2}(u-\xi)\right)\,du\right]\\ &= \frac{1}{8}\frac{\nu}{\nu-1}\left[\exp\left(-\frac{\xi}{2}\right)-\exp\left(-\frac{\zeta}{2}\right)\exp\left(-\frac{\nu}{2}(\xi-\zeta)\right)\right]\\ &\quad+\frac{1}{8}\frac{\nu}{\nu+1}\exp\left(-\frac{\xi}{2}\right)\end{aligned} \tag{34}$$

Yielding

$$\pi(\xi) = \begin{cases} \frac{1}{4}\frac{\nu}{\nu+1}\exp\left(\nu\frac{\xi-\zeta}{2}\right) & \xi < \zeta \\ \frac{1}{4}\frac{\nu}{\nu-1}\left[\frac{2\nu}{\nu+1}\exp\left(-\frac{\xi-\zeta}{2}\right)-\exp\left(-\nu\frac{\xi-\zeta}{2}\right)\right] & \xi > \zeta \end{cases} \tag{35}$$

The average error is then

$$\begin{aligned}\mathbb{E}[p(\xi)] &= P_E\mathbb{E}[\pi(\xi)]\\ &= P_E\int_0^\infty \xi\pi(\xi)\,d\xi\\ &= P_E\frac{1}{4}\frac{\nu}{\nu+1}\int_0^\zeta \xi\exp\left(-\nu\frac{\xi-\zeta}{}\right)\,d\xi\\ &\quad+P_E\frac{1}{4}\frac{\nu}{\nu-1}\left[\frac{2\nu}{\nu+1}\int_\zeta^\infty \xi\exp\left(-\frac{\xi-\zeta}{2}\right)\,d\xi-\int_\zeta^\infty \xi\exp\left(-\nu\frac{\xi-\zeta}{2}\right)\,d\xi\right]\\ &= \frac{2}{\nu+1}\left(\frac{\zeta}{2}+1\right)\exp\left(-\nu\frac{\zeta}{2}\right)+\frac{1}{\nu(\nu+1)}\exp\left(-\nu\zeta\right)\end{aligned} \tag{36}$$

5 Theoretical cost model

Whenever we consider the cost of an indexing algorithms, there are at least two magnitiudes that we might be interested in measuring: the number of floating point operations (typically, those involved in the computation of distances), or the number of random access to disk blocks. The first measure is especially relevant for random access memory data bases, while in the case of data bases stored on disk the second measure greatly dominates the first (with standard equipment, the cost in terms of disk access can be of the order of 10^5–10^6 times that of in-memory computation). Both measures have their application area, and both should be considered when doing the *a priori* evaluation of an algorithm outside of the context of a specific application.

In our algorithm, the cost in terms of disk access depends essentially on the cost model of the index on which we build. Developing a theoretical cost model for these indices goes beyond the scope of this paper, so we shall consider this cst mainly in the experimental part, in Section 7. Here, we shall develop the computational cost model of the algorithm.

There are three steps of the algorithm in which distances are measured: the nearest neighbor query of step 2, the range query used to build the set T at step 4, and the limited scope

nearest neighbor query of step 5. The first two queries are executed on the reduced dimensionality space Ξ, of dimensionality M, the last one on the whole space Ω of dimensionality $N \gg M$.

Not all these computations have the same cost. In step 5 we may compute the square of the distance, with a cost of N multiplications. Steps 2 and 4, on the other hand, are executed using an indexing method, many of which rely on the triangle inequality and therefore require the computation of a square root. A simple iterative algorithm like the following to compute $\sqrt{y_0}$,

$$\begin{aligned} y &\leftarrow x * x - y_0 \\ x &\leftarrow x - y/(2 * x) \end{aligned}$$

can compute the square root with a precision of 10^{-8} for $y_0 < 10^6$ in less than 10 iterations, that it, using less than 50 multiplications, so the cost of the distance computation in an M-dimensional space can be estimated in $I + M$, where I is of the order of 50.

For a data base of D elements in an M-dimensional space, let $S(D, M)$ be the number of distance computations in the indexed search. We assume that the cost of nearest neighbor and range searches are of the same order of magnitude, a reasonable assumption for most indexing methods. Note that $S(D, M) = O(D)$ for large M, while in general $S(D, M) = O(\log D)$ for small M. With these definitions, the average cost of one search can be estimated as

$$C(D, N, M) \sim (I + M)S(D, M) + N \cdot \mathbb{E}[|T|] \tag{37}$$

In order to determine the cost, we need to determine the expected value of $|T|$, $\tau \stackrel{\triangle}{=} \mathbb{E}(|T|)$. Consider $x \in \mathbf{D}$; in step 2 of the algorithm we determine the nearest neighbor in the Ξ space, n. In step 5, it is $x \in T$ if $\rho^2_\Xi(q, x) - \rho^2_\Xi(q, n) < \alpha$. The probability of this happening is

$$\begin{aligned} &\mathbb{P}\left\{\rho^2_\Xi(q, x) - \rho^2_\Xi(q, n) < \alpha \,\middle|\, \rho^2_\Xi(q, x) - \rho^2_\Xi(q, n) > 0\right\} \\ &= \mathbb{P}\{\Delta_\Xi + \Delta_\Theta < \alpha \,|\, \Delta_\Xi + \Delta_\Theta > 0\} \\ &= \frac{\mathbb{P}\{\Delta_\Xi + \Delta_\Theta < \alpha, \Delta_\Xi + \Delta_\Theta > 0\}}{\mathbb{P}\{\Delta_\Xi + \Delta_\Theta > 0\}} \\ &= 2 \cdot \frac{1}{4\sigma^2_\Xi} \int_0^\alpha \exp\left(-\frac{x}{2\sigma^2_\Xi}\right) dx \\ &= 1 - \exp\left(-\frac{\alpha}{2\sigma^2_\Xi}\right) \\ &= 1 - \exp\left(-\frac{\zeta}{2}\right) \\ &\stackrel{\triangle}{=} \beta(\zeta) \end{aligned} \tag{38}$$

The probability that, out of the D elements in the data base, k are in T is therefore

$$P_\zeta(k) = \binom{D}{k} \beta(\zeta)^k (1 - \beta(\zeta))^{D-k} \approx \frac{(K\beta(\zeta))^k}{k!} \exp(-D\beta(\zeta)) \tag{39}$$

where the Poisson approximation is valid for $large$ D. Then

$$\tau(\zeta) = \mathbb{E}[|T|] = \mathbb{E}[P_\zeta(k)] = D\beta(\zeta) = D\left(1 - \exp\left(-\frac{\zeta}{2}\right)\right) \tag{40}$$

If we are interested in the complexity for a prescribed error probability p, we plug in eq. (27) obtaining:

$$\begin{aligned}\tau(p) &= D\left(1-\exp\left(-\frac{1}{1+\nu}\log\frac{1}{2(1+\nu)p}\right)\right)\\ &= D\left[1-(2(1+\nu)p)^{\frac{1}{1+\nu}}\right]\end{aligned} \tag{41}$$

6 Validation

In this section, we validate the model of the previous section comparing its prediction with the measurements obtained from an instrumented implementation of the algorithm.

We validate the model of error and cost using the data from of the SUN data set [63], containing 130,000 images, more than enough to obtain statistically significant results. The main reason for using this data set is that it comes with a variety of pre-computed features of different size and statistical characteristics (see Table 1), allowing us to validate the model under a variety of conditions using features computed in a standard way, thus increasing replicability.

The validation was carried out on all 12 features but, in order to simplify the presentation, we shall present the data relative to only two of them, representative of the range of sizes and performance obtained: *Geo Texton*, and *Tiny Image*. The former is one of the features with the best fit to the theoretical model, the latter is the one with the worst fit.

Our main hypothesis, the one on which the algorithm is based, is that the variance along each axis decreases rapidly with the axis index after SVD has been applied to the data base. Figure 2a shows the results for features of the SUN data base, which confirms our hypothesis.

The rate at which the variance decreases depends on the feature, the slowest descent being for *HoG*, the fastest for *Geo Texton*. The corresponding values of ν are shown in Fig. 2b. Note that for $m \geq 5$ the condition $\nu > 1$ is met for all the features with the exception of *HoG*.

Table 1 The individual features of the SUN data set with their individual and their total size

Feature	Size	
Dense SIFT	784	
Geo color	784	
Geo map 8 × 8	256	
Geo texton	512	
Gist	512	
Gist padding	512	
Hog 2 × 2	6,300	
Lbp	798	
Lbphf	1239	
Line hist	230	
Texton	10,752	
Tiny image	3,072	
	25,751	Total

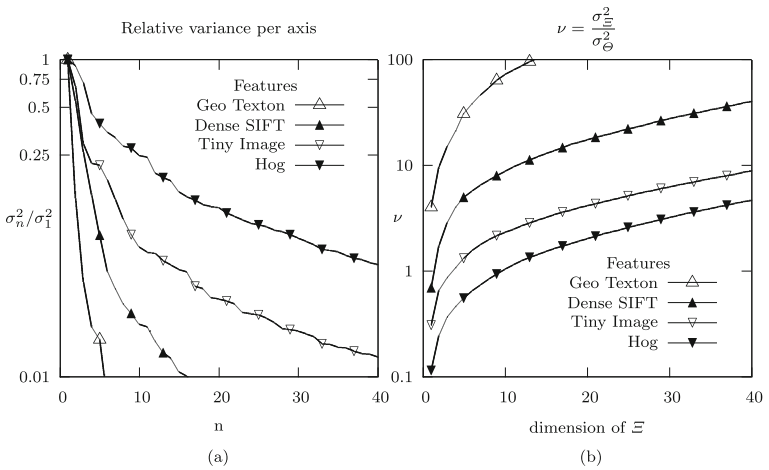

Fig. 2 In (a) above, the behavior of the variance on each axis of the feature space, transformed using SVD, for four features of the SUN data base. The abscissa is the axis number (the axes are ordered by decreasing eigenvalue) and the value reported is σ_n^2/σ_1^2, where σ_n^2 is the variance on the nth axis. Because of the normalization, the value for $n = 1$ is 1 for all features. In (b), below, the corresponding values of $\nu = \sigma_\Xi^2/\sigma_\Theta^2$ for increasing values of the dimensionality of Ξ

Figure 3 compares the sampled distribution of ρ^2 with the theoretical model (22) for *Geo Texton* and *Tiny Image*.

The model captures reasonably well the general trend of the distribution for *Geo Texton*, but it is quite off at low and medium distances for *Tiny Image*. Note, however, that in our model we never use the distribution of ρ^2 directly; what we do use is the distribution of the difference between the square of two distances, which we assume to follow a Laplace distribution.

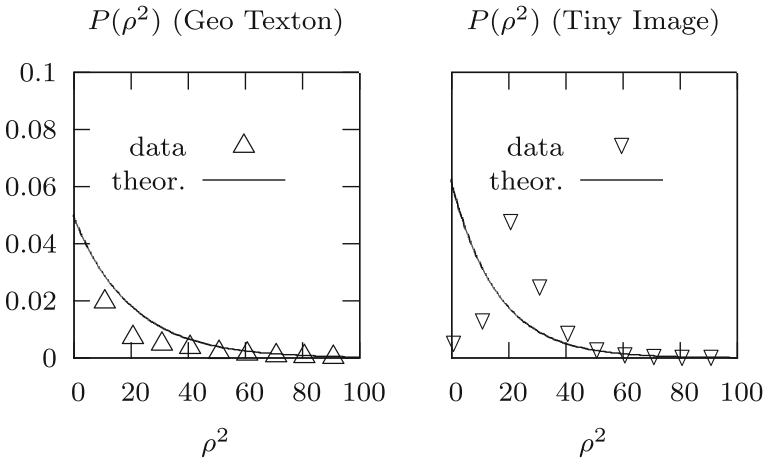

Fig. 3 The fit between the model (22) and the sampled distribution of the square of the distance between points (ρ^2) for two feature spaces of the SUN data set: *Geo Texton* (size 512), and *Tiny Image* (size 3,072). The sampled distributions were estimated using a sample of 15,000 images

The sampled distribution of $\Delta = \rho^2(q, x) - \rho^2(q, y)$ as well as the theoretical model (23) are shown in Fig. 4. The fit is in this case much better than that of ρ^2, although it still presents discrepancies for *Tiny Image*.

Figures 5 and 6 show the sampled probability of success (actually finding the nearest neighbor) versus the theoretical model as a function of ν (and therefore of $\dim(\Xi)$) for various values of ζ and for the features *Geo Texton* and *Tiny Image*. Note that for *Geo Texton*, ν grows very rapidly, so that one can have a high probability of success with spaces Ξ of very low dimensionality. In the case of *Tiny Image*, ν grows slowly, as can be expected given the slower drop in variance shown in Fig. 2.

Figures 7 and 8 show the average error in the distance between the true nearest neighbor and the approximate one for *Geo Texton* and *Tiny Image*. The sampled data are compared with the theoretical model. The model is in both cases a good fit to the data. Note that in the case of *Tiny Image* the values of ν that we get for given values of $\dim(\Xi)$ are about one order of magnitude smaller than those of *Geo Texton* and, correspondingly, the values of $\mathbb{E}(\Delta\zeta)$ is about one order of magnitude larger.

7 Cost

In order to make some indicative measurements of the cost of approximate indexing, we need to select an indexing method on which to base our algorithm. This method will play a double role: on the one hand, it will serve as the reference method against which we shall compare: we shall store the whole vectors in the index and determine the cost of search. We shall call this the *full indexing* method, and will be the yardstick against which we shall measure the cost of our algorithm. One the other hand, we shall use it as the indexing method that we need for the initial searches in the space Ξ.

In our tests, for this purpose, we shall use *K-D trees* [11], which are an easy to implement, well-established and well-known method whose performance is in line with those of

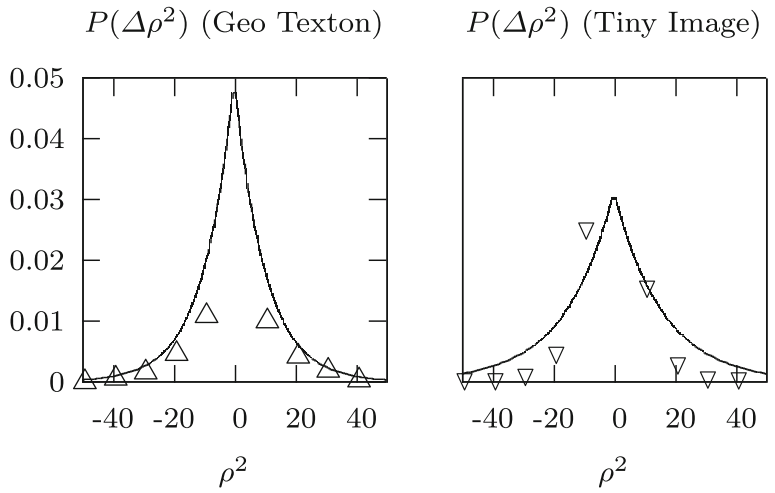

Fig. 4 The fit between the model (23) and the sampled distribution of the distance difference $\Delta = \rho^2(q, x) - \rho^2(q, y)$ for two feature spaces of the SUN data set: *Geo Texton*, and *Tiny Image*

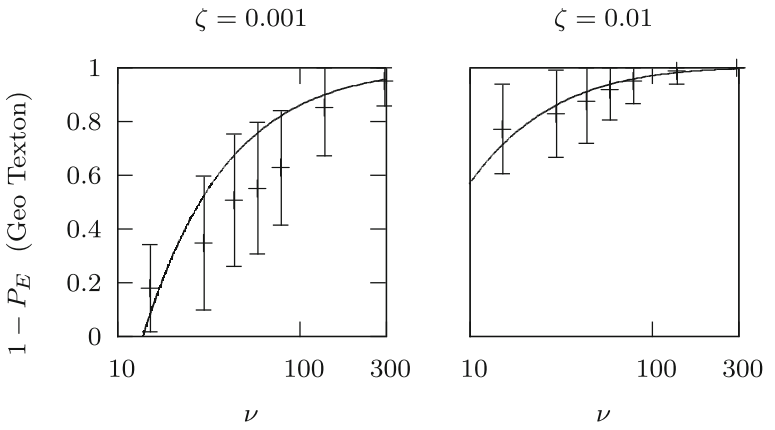

Fig. 5 Sampled probability of success compared with the theoretical model for *Geo Texton*. The graphs show the model as a function of ν (and therefore as a function of $M = \dim(\Xi)$) for various values of ζ

most partition-based methods. The use of other indices, such as M-trees [20] would not appreciably impact the performance in terms of operations, while it would have a very minor effect on disk accesses. The results found in the following would remain valid, and the comparisons would change slightly in favor of our algorithm.

As we mentioned in Section 5, there are basically two ways of measuring the cost of a search algorithm. If the algorithm is executed in central memory, then one usually isolates the most expensive operation (in our case: the computation of the distance) and counts how many times it is executed. If the algorithm is executed on disk, then the cost of accessing a random sector overshadows all other costs, and one usually counts disk accesses. In this section, we shall determine the performance of our algorithm using the two measures. The methodological basis of our measurements are laid out in Appendix A.

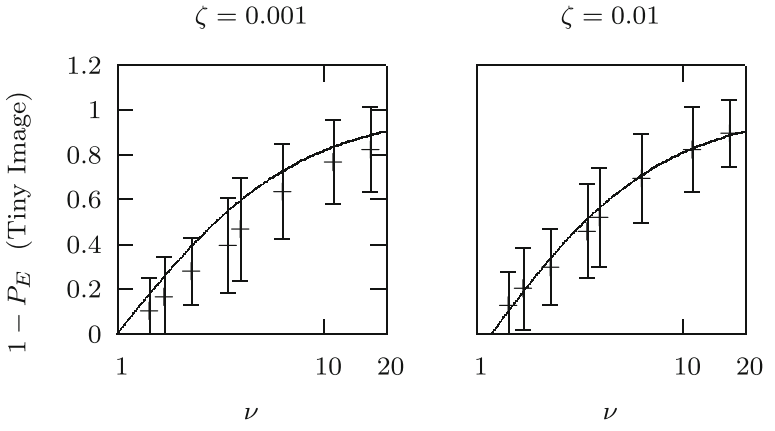

Fig. 6 Sampled probability of success compared with the theoretical model for *Tiny Image*. The graphs show the model as a function of ν (and therefore as a function of $M = \dim(\Xi)$) for various values of ζ

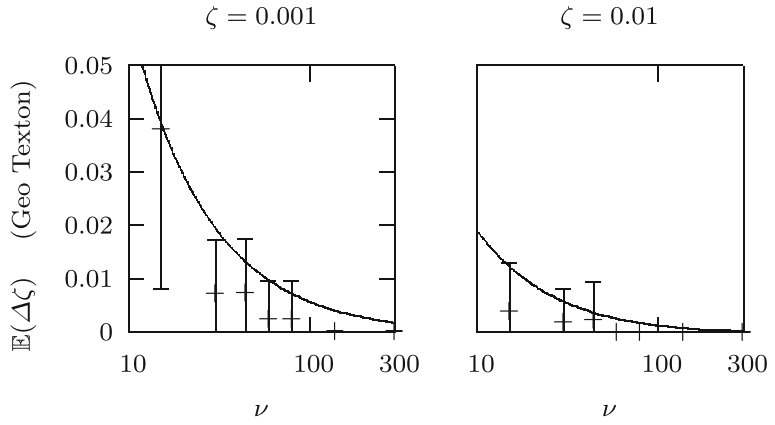

Fig. 7 Sampled average error between the distance of the true neighbor compared with the theoretical model for *Geo Texton*. The graphs show the model as a function of ν (and therefore as a function of M) for various values of ζ

7.1 Computational Cost

In this section, we use the cost model (37). For the exact search, carried out on the whole feature vector of dimensionality N, there is no T list, so $\tau = 0$ and the cost is $(I + N)S(D, N)$. In the test, we measure $S(D, N)$ during the search in the kd-tree, by applying suitable instrumentation to the algorithm. For approximated searches with $\dim(\Xi) = M$, (37) applies. The value of $S(D, M)$ depends on the number of points that fit in a node of the kd-tree that is, considering that a node is stored in a disk cluster, on the size of a cluster. We assume that the disk has clusters of $B = 25,000$ bytes and that each element of the vector is a floating point number represented in $f = 4$ bytes. So, if we are dealing with vectors of size N, the number of vectors per node will be $V = \lfloor B/(f \cdot N) \rfloor$. Table 2 shows the number of vectors

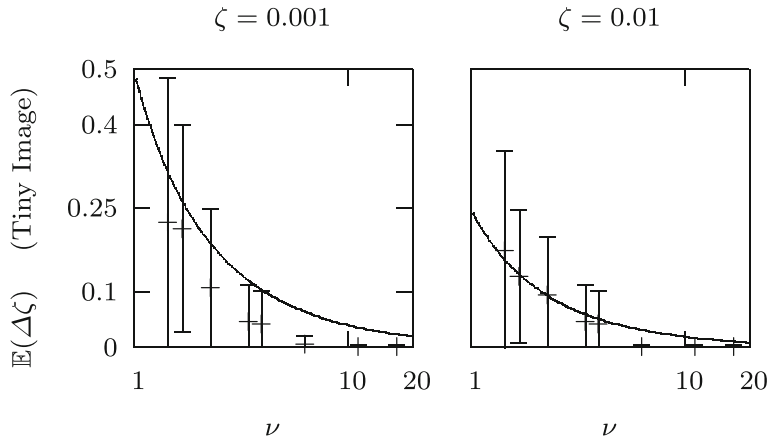

Fig. 8 Sampled average error between the distance of the true neighbor compared with the theoretical model for *Tiny Image*. The graphs show the model as a function of ν (and therefore as a function of M) for various values of ζ

Table 2 The number of points that fit in a disk cluster ($B = 25,000$ bytes, $f = 4$ bytes) for the dimensionalities of the space that we shall use in our tests. The first seven values are dimensions of the reduced space Ξ that we use for approximate search, while the last two columns are the dimensionalities of the full *Geo Texton* and *Tiny Image* features, which we use for the full search

	Reduced spaces Ξ							Geo	Tiny
Dim.	5	10	20	30	50	100	200	512	3,072
V	1,250	625	312	208	125	62	31	12	2

per node for various values of the dimensionality of the feature space. The last two columns represent the dimensionality of *Geo Texton* and *Tiny Image*, that is, they are the number of vectors per node (and per cluster) in the full searches.

Figure 9 shows the average computational cost (37) as a function of D for the full indexing scheme and for the approximate indexing for various values of M and for $\zeta = 0.001$ (Fig. 9a) and $\zeta = 0.01$ (Fig. 9b).

For the sake of clarity, in the figures we show only the variance of the full indexing at few points, but the difference between full indexing and approximated have been determined to be significant ($p < 0.01$). The full index, as expected, has a complexity that grows linearly with the size of the data base. The approximate index also grows more or less linearly, due essentially to the fact that $|T|$ grows linearly with D, as shown in (40). In both cases, we can obtain a probability of error of less than 5% and an error in the distance of less than 0.01 with a cost an order of magnitude less than the full index (Table 3).

The situation is similar for *Tiny Image* (Fig. 10) although, given the slow descent of the per-axis variance and the slow rise of ν (Fig. 2) the error probability and the error in the distance are bigger: in order to get a 7% error rate, we need $M \sim 200$. With so large a space, it is likely that the increased performance is not due to the more efficient operation of the K-D tree (on a 200-dimensional space, the performance of partition methods is severely degraded), but to the lower cost of distance computation due to the reduced dimensionality. The difference in performance is, in this case as well, roughly one order of magnitude.

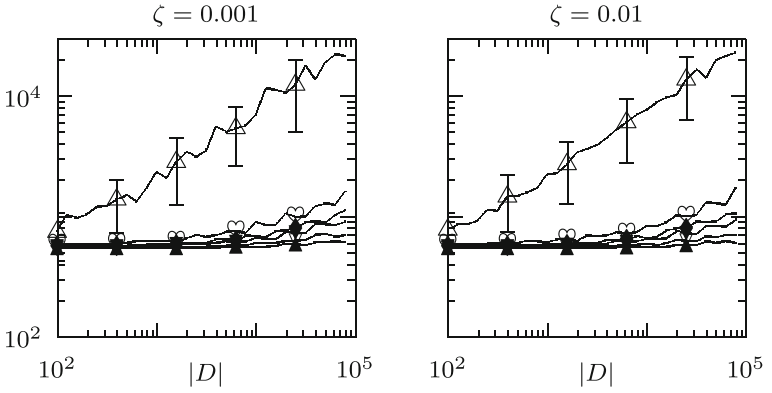

Fig. 9 Complexity, in number of operations necessary to compute the distances, of the full index and the approximate index for *Geo Texton*, for various values of M. For the data relative to the different curves, see Table 3. The "bumps" in the curves, evident mainly in the full indices, corresponds to the change in height of the K-D tree

Table 3 Data relative to the performance curves. For each one of the features, the curves with the symbol indicated in the first column are relative to the dimensionality of Ξ indicated in the second column. This determine the value of ν and, for given ζ, values of the probability of error P_E and the average distance error $\mathbb{E}(\Delta\zeta)$

			$\zeta = 0.001$		$\zeta = 0.005$	
Curve	M	ν	P_E	$\mathbb{E}(\Delta\zeta)$	P_E	$\mathbb{E}(\Delta\zeta)$
Geo texton						
△	512	∞	0	0	0	0
▲	5	29.1	0.15	0.02	0.08	0.019
▽	10	65.1	0.05	$8 \cdot 10^{-3}$	0.01	$7 \cdot 10^{-3}$
▼	20	135.5	0.02	$4 \cdot 10^{-3}$	10^{-3}	$3 \cdot 10^{-3}$
♦	30	209.3	$8 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	10^{-4}	$1.7 \cdot 10^{-3}$
♡	50	374.3	$2 \cdot 10^{-3}$	10^{-3}	10^{-5}	$6 \cdot 10^{-4}$
Tiny image						
△	3,072	∞	0	0	0	0
▲	20	2.3	0.3	0.18	0.28	0.18
▽	30	2.9	0.25	0.14	0.22	0.14
▼	50	4.0	0.19	0.1	0.18	0.1
♦	100	6.4	0.13	0.06	0.11	0.05
♡	200	11.42	0.07	0.04	0.05	0.03

7.2 Disk access

The cost of distance computations is a good indicator of performance for in-memory data bases, but if the data are stored on disk, the cost of a search is essentially that of random accesses to clusters. In order to determine the efficiency of our indexing method for disk operations, we assume again that the data are stored in a K-D tree in a disk with clusters of $B = 25,000$ bytes. The internal nodes are stored in memory while the leaves, which contain

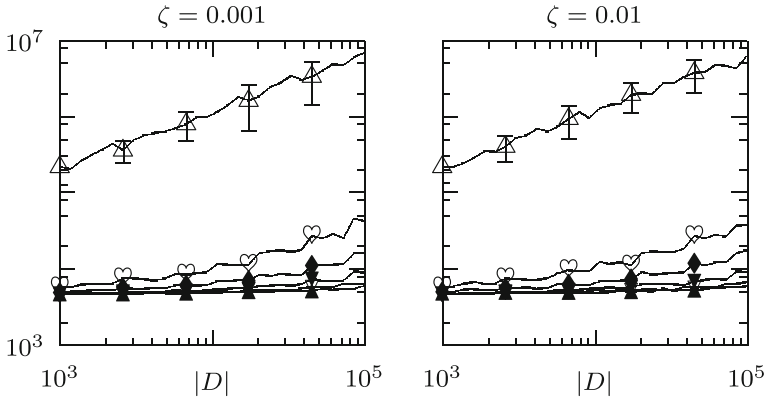

Fig. 10 Complexity, in number of operations necessary to compute the distances, of the full index and the approximate index for *Tiny Image*, for various values of M. For the data relative to the different curves, see Table 3

the data, are stored on disk. The number of points that can be retrieved with a random page access is given in Table 2. In the case of full indexing, the disk accesses that we do to obtain the necessary nodes constitute the whole cost of the search. In the approximate search, we use the K-D tree on the reduced space (and, therefore, with more points per cluster) to build the set T, then we need to access the points of T in the full space in order to find the approximate nearest neighbor. In order to make this access more efficient, we store the full vector corresponding to the points in a node of the tree in consecutive clusters so that they can be read with a single random access. Thus, once the points of T have been found in the tree, the full representation can be obtained with a minimum of random accesses on disk.

Figure 11 shows the number of random cluster accesses for *Geo Texton*, for $\zeta = 0.001$ and $\zeta = 0.01$.

Note again that, with the exception of $M = 50$, the number of disk accesses is about one order of magnitude smaller than that of full search. Similar considerations hold for *Tiny Image*, whose results are shown in Fig. 12.

All these comparisons assume that the K-D tree for the approximate algorithm is stored on disk. However, the smaller size of the data points may make it feasible to store it in central memory, at least for the smaller values if M, leaving only the full representation on disk. In this cases, the cost of the approximate indexing drops dramatically: our tests show that in most cases the approximate nearest neighbor can be found with less than two disk accesses for *Geo Texton* and less than five disk accesses for *Tiny Image*.

7.3 Comparative test

The state of the art in indexing (see Section 8) is huge, and a full comparison between a new model and a representative sample (considering and clarifying all the variables and the different presuppositions involved) could be the subject of a paper in its own right. Here, we do an experimental comparison with two recent methods of different nature that share some of our presuppositions: *Hierarchical Navigable Small World Graphs* (HNSW, [45]) applied to a sub-space of $\mathbb{R}^D$ obtained by SVD, and the Inverted Multi-Index (IMI, [5]). As a reference, we use Geo Texton with $M = 20$ and $\zeta = 0.01$, which gives an error probability $P_E = 0.02$. The parameters of the other methods were adjusted by trial and error so that the measured error probability would be the same P_E. The details are available in Appendix 13.

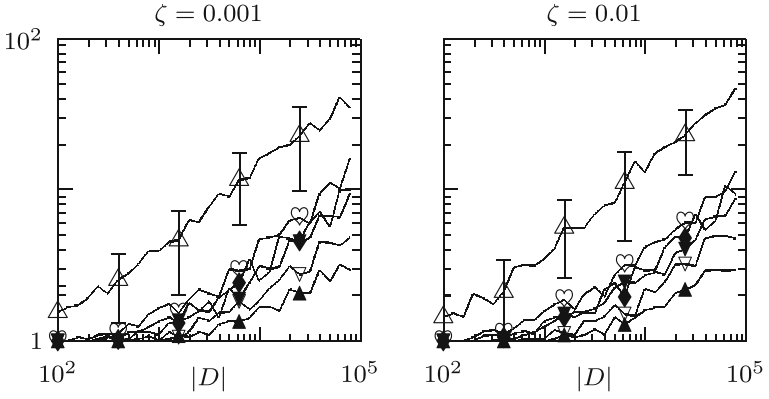

Fig. 11 Complexity, in number of random block accesses on disk, of the full index and the approximate index for *Geo Texton*, for various values of M. For the data relative to the different curves, see Table 3

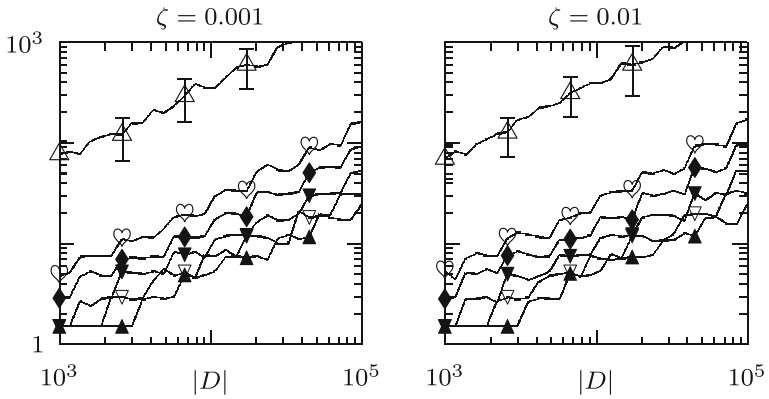

Fig. 12 Complexity, in number of random block accesses on disk, of the full index and the approximate index for *Tiny Image*, for various values of M. For the data relative to the different curves, see Table 3

Note that we do not consider the time necessary to build the index which, in the case of IMI, is dominated by the k-means necessary to build the code book, making the method ill-suited for dynamic data bases.

The results for computational and disk access cost are shown in Fig. 13.

The method presented here (labeled "meta" in Fig. 13) outperforms HNSW and has performance similar to IMI. In both cases, the main advantage of our method is that we can work out the initial search with very small vectors, smaller than those we can use with the other methods to obtain comparable performance. This increase the number of points that we can store in a cluster, providing an advantage that the following post-processing (heavier in our method that in the others) does not completely eliminate. It should be noted, however, that these tests have been made without disk cache, and that HNSW, because of its characteristics, will benefit from caching more than the other methods. The system designer

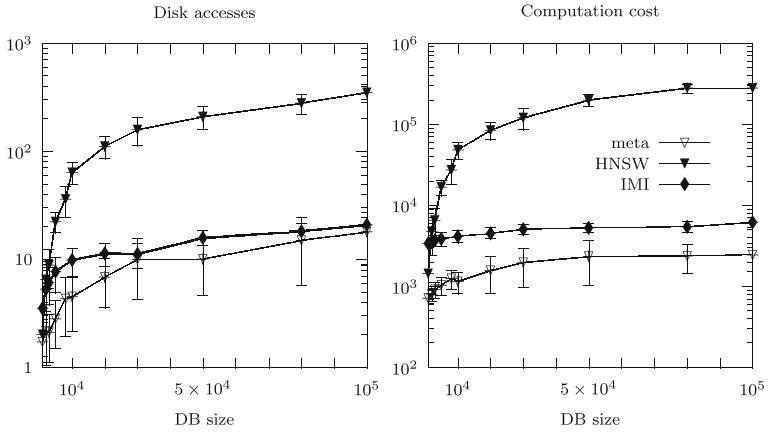

Fig. 13 Performance comparison of the method presented here with HNSW and IMI (see text). The method presented here has been used with $M = 20$ and $\zeta = 0.01$, corresponding to $P_E = 0.02$. The parameters of the other methods have been adjusted by trial and error for each data base size (the parameters vary with the size of the data set for both methods) so as to obtain the same error

should therefore consider carefully the characteristics of its hardware and operating system before deciding what indexing to use.

8 Related work

Indexing in high-dimensional spaces has been an important topic in multimedia and data bases for the past 25 years, at least since the QBIC [26] search engine showed that query-by-example was a viable technique for retrieval. Times have changed, and many multimedia techniques have been developed, but indexing in high-dimensional spaces remains a crucial topic for which there seems to be no "silver bullet" such as B-trees are for ordered data [9].

One of the earliest multidimensional indices was [15], in which a tree (called the BKT, or *Burkhard-Keller Tree*) was introduced, suitable for searches in metric spaces with discrete-valued distances. Other early solutions for multi-dimensional spaces were the *grid files* [50], and IBIM (*Interpolation-Based Index Maintenance*) [16], a hash method for range queries in multi-dimensional data. The idea of designing has functions that map nearby points to the same disk page has been used ever since. A relatively recent incarnation, *Locality Sensitive Hashing* (LSH, [56]) adapts it to probably correct indices with a bounded error probability. The general idea of LSH bears some resemblance to ours, but the need to store multiple random projections limits its disk efficiency.

One of the most common classes of indexing methods is based on space partitioning: a tree is built in which each node "covers" a portion of the space, and the children of the node cover parts of the region covered by the parent (possibly overlapping and not covering the whole region covered by the parent). Many of the early methods of this class were originally developed for geographic data, that is, for two-dimensional spaces. The first such structure to gain widespread recognition was the R-Tree [30], in which each node represents a rectangle, and the children of the node represent rectangles included in the parent's. The children do not partition the parent: depending on the data distribution, areas of the parent without data points could be left uncovered, while there may be overlaps between the rectangles of the children. While the former is an advantage (no need to represent portions of the space in which there are no points), the latter is a burden (when traversing the tree, one might need to traverse several sub-trees in parallel, degrading performance). Over the years, many improvements have been superimposed to the basic R-Tree design, either to improve its heuristics [10], to refine the tree structure [12] or in general to improve their performance [31, 36, 55]. One of the problems one encounters when trying to use R-Trees for high-dimensional feature spaces is the expensive representation of the nodes. Each node is a rectangle, and can be represented by fixing two opposite corners. For a N-dimensional space, this requires $2N$ floating point numbers; for high-dimensional spaces and standard cluster sizes, this means that few internal nodes can fit in a cluster. Some variations palliate this problem: the SS-Trees [62] use spheres ($N + 1$ floating point numbers per node), while G-Trees [39] use space partitions to organize multidimensional data in a B-Tree. K-D Trees [11] splits one axis at each node, a split that can be represented by an integer (the axis number) and one floating point number (where does the split occurs). A variation of this idea uses random partitions of the feature space [65], while solutions like the *Set Compression Tree* [3] use a structure similar to the K-D Tree to compress a set of vectors into a compact representation, thus fitting more points in a disk cluster. Almost all the methods proposed store the data on disk, but some solutions have been devised for data in central memory [24].

All these methods assume that the data are elements in a vector space. Other methods make the logically weaker assumption that the data are elements of a metric space that is, they make no use of the vector components, but index based solely on the distance values between items (note however that a set of data for which only distances are known can always be approximately be represented as a set of vectors using techniques such as multidimensional scaling [23, 58, 59]). The *BKT* [15], which we have already mentioned, is a method of this kind. Other indices of this class include the *Fixed Query Tree* [7] and their fixed height variation (FHQT, [6, 8]), *Fixed Query Arrays* [17], which is essentially a compact representation of FHQT, *Vantage Point Trees* (VPT, [19, 60]), and their generalization, *Excluded middle Vantage Point Forest* [64].

The best known and most successful method in this class is probably the M-tree [20], a tree in which each node contains a representative and a radius representing the region of space it covers. Sub-trees rooted at that node cover regions included in that of the parent. Thanks to a careful use of the properties of the distance function—in particular, the triangle inequality—storage and indexing can be done in the absence of a vector representation of the data. The M-tree has been shown to often outperform partition methods based on vector spaces.

Methods such as the MVP-tree [14] and the ball tree [51] are based on similar principles. These methods represent so far the practical state of the art for partition-based exact search in high dimensional data [47].

Partition methods are probably the most studied and well known, but they are not the only ones. In [33] a structure is proposed in which memory units summarize portions of the data base in a single vector. In [49] a fast algorithm based on projections, and supported by a specially designed data structure, is developed to answer ϵ-range queries, while in [54] specialized disk allocation methods are studied. More recent methods, such as [41, 42] are based on the creation of suitable one-dimensional indices in randomly chosen dimensions. The indices are created in such a way that elements close to each other are placed relatively close in the index. Alternatively, multidimensional indices of reduced size are created [43, 57].

Most of the so-called "curse of dimensionality" derives from the concentration of the Minkowski distance in high dimensions. In [1] the problem is addressed at its roots: a distance function is defined that remains discriminative in high dimensions. The method presented here would make the best of the characteristics of this distance.

Experimental analysis shows that for most workloads, indices stop being competitive vis-à-vis serial scan for dimensions as low as 10-20 [61], and that the very notion of "nearest neighbor" might be meaningless [13], although [38] points out that the situation might not be so dire, as most of the data sets of interest have *intrinsic* dimension much lower than the dimension of the space in which they are represented, and that the important dimension for indexing is the intrinsic. The method that we have presented here is based on the same property.

The difficulty in creating efficient indices for exact nearest neighbor queries has created a great interest in approximate algorithms such as the one we presented here.

The *Balanced box-decomposition tree* (BBD-Tree [4]) is a structure than can be created in time $O(N D \log D)$ that can find an approximate nearest neighbor with a distance from the query at most $(1+\epsilon)$ times that of the true nearest neighbor in time $O(C \log D)$, with $C \leq N\left[1+\frac{6N}{\epsilon}\right]^N$. The BBD-Tree guarantees that the $(1+\epsilon)$ bound is met; in [21] this constraint is relaxed: the algorithm only makes a probabilistic promise: with probability δ it will return a point whose distance from the query is at most $(1+\epsilon)$ times that of the true

nearest neighbor. The method presented here does a similar claim, but our model allows us to derive the complete probability distribution of the error ζ. A hard limit similar to that in [21] can be derived by thresholding the integral of ζ. Comparison between the two methods is not immediate, but a comparison of Figs. 9 and 10 with Figure 4 in [21] suggest that the method presented here is more efficient (in terms of operations) for features for which ν grows reasonably rapidly. Other tree-based methods for approximate search include spill-trees [44] as well as variations of K-D trees [48].

A different class of methods is based on hashing, that is, on computing disk addresses directly from the coordinates, rather than on partitioning the space [2, 28]. One interesting class is that of Locality Sensitive Hashing (LSH), which we already mentioned [32] based on principles similar to linear hashing [15] and IBIM. In their more recent versions, they provide theoretical guarantees and performances comparable to the method presented here. Given that what we present here is a "meta" method built on top of a nearest neighbor index, an interesting experiment would be to build the present model on top of LSH.

Methods not based on trees include hashing [2, 28] or decompositions of the space into sub-spaces that can be quantized for efficiency [27].

9 Conclusions

The method that we have presented here can be considered, more than an indexing method, a meta-indexing strategy. It can be used to improve the performance of any indexing method as long as two basic premises are met: (1) the performance of the indexing method degrades as the dimensionality of the feature space increases, and (2) most of the variance of the inter-item distances is reasonably concentrated in a relatively low-dimensional sub-space of the feature space (the space that we have called Ξ). If these conditions are met, the method presented here can piggyback on the index to offer the designer a fine control of the trade-off between efficiency and accuracy.

Several directions of possible development remain open. The first is the dimensionality reduction. In our work, we have used SVD, which makes the effectiveness of dimensionality reduction dependent on the distribution of the data: the method is very efficient for some features (such as *Geo Texton* in our example), less so for others (*Tiny Image*), depending on the distribution of the data. However, the Johnson-Lindenstrauss Lemma [35] states that one can embed a data set of D points into a space of dimension $O(\log\ D)$ with little distortion of the pair-wise distance. Since for most features $\log\ D \ll N$, the Lemma opens the possibility of defining sub-spaces Ξ of low dimensionality that may work in a more general setting than those created using SVD. This relates also to the observations that we made in Section 1 on the intrinsic dimension d_I. The intrinsic dimension, much like the Johnson-Lindestrauss lemma guarantees that a manifold of limited dimensionality exists that allows a faithful representation of the data, but gives no indication on how to find it.

Furthermore, as we already mentioned (page 3), the numerical characterization of a manifold that contains D points in $\mathbb{R}^N$ may require up to $O(N D)$ parameters, making an index based on it not competitive. One interesting direction of work is the study of non-linear manifolds with compact representation that can be used in lieu of our linear sub-space Ξ.

Another possible development is the use of approximate nearest neighbor method for the initial search on Ξ for example, as we have already mentioned in the previous section, to build it on top of LSH. This, of course, will require a change in the analytical model, as the initial nearest neighbor on Ξ may itself contain an error. This also entails that, *cæteris*

paribus, the probability of error will be greater if the search on Ξ is itself approximate. In order to obtain results comparable to those of the method with exact indexing, one has to increase $\dim(\Xi)$ or ζ. The question is whether the increased performance deriving from the use of the approximate indexing will compensate the need to increase $\dim(\Xi)$ and ζ. Only the development of the appropriate model will allow us to answer this question.

Appendix A: Methodological Considerations

The paper presents an algorithm for approximate indexing, one that will be used as part of a retrieval system with certain characteristics. The system in which the algorithm will be inserted, that is, the *context* of the algorithm will clearly have an impact on its performance (it is a common general observation that an algorithm that is the best in a specific system may not be the best in a different system). When testing the algorithm, one should try to abstract as much as possible from the context and determine the performance of the algorithm in "splendid isolation", to cite Sir Wilfrid Laurier. These considerations place several constraints on the tests that we can and cannot make. The most immediate are three.

1. We can't do any user study, as the response of a user depends on the whole system, of which the algorithm in question is but a small part. Testing a system as a whole would multiply the number of uncontrolled variables, making a test virtually meaningless.
2. We cannot use standard IR measures such as precision, as these depend, among other things, on the quality of the features that the system is using. An indexing algorithm receives a query point in the feature space and returns the k nearest points in the data base. Whether these points represent semantically close elements or not is a matter independent of the algorithm and that therefore should not enter our considerations.
3. We cannot measure performance using execution time, as the execution time depends on a number of variables, including the CPU speed, the amount of central memory, the disk efficiency, that of the operating system, and many more. Again, too many uncontrolled variables.

It is not possible to evaluate our algorithm in complete isolation, since it is a methods that sits on top of an exact k-nn indexing structure. One possibility would have been to use exhaustive search as the underlying structure, but in this case the linear search time would have masked one of the most important advantages of our methods: the reduction of the dimensionality of the data that are searched. A more realistic evaluation would use a method that falls prey of the curse of dimensionality for high dimensions but maintains a logarithmic performance at low dimensions. The KD-trees that we use are a good compromise in this sense and, being pretty much a classic on which much work has been done, they are a stable basis on which we can test the method. Note that we have intentionally avoided the use of hashing methods as their performance are more discontinuous as the dimensions increase, and their very strong performance at low dimensions may lead to overestimating the performance of our method. Our goal, it is worth reminding, is to test the method in different conditions, and not to maximize speed: that is a job for the system designer, based on the relative indications that we provide.

For the same reason, we do not implement the index on disk, but we simulate the disk operations in central memory. This allows us to control all the variables of the test, in particular, to avoid caching and to control the size of the disk cluster (which, short of reformatting

the disk for each test, we can't do in an actual implementation). This choice limits the number of feature points that we can use but, thanks to the control that we gain, we don't have to use very large data sets to get statistically significant results. Here too we should remember that the size of the sample is not an arbitrary parameter subject only to the dubious principle "more is better", but a value determined by the overall experimental design.

In order to have a measure independent of the performance of the operating system and the computer, we use two measures: the number of multiplications in the computation of distances and the number of disk block reads in the hypothesis of no page cache.

Appendix B: Comparison details

In order to compare our method with the reference ones, we need to embody them in our simulation framework in such a way to create as fair as possible a comparison with our own. We give here a few details of how this is done. We assume that the reader is familiar with the methods we are comparing with.

B.1 HNSW

This method, presented in [45] is based on a hierarchy of graphs $G_i = (V_i, E_i)$ such that $N_{i+1} \subseteq N_i$. The nodes are points in the feature space. In our implementation, we perform SVD to reduce the dimensionality of the space: if $\mathbb{R}^D$ is the space, we take a number of dimensions $D' \leq D$, choosing those with the highest variance. This introduces an error, so we shall have to balance D' and the other parameters of the method to achieve the desired error probability. The method doesn't allow us to predict the error probability, so we set the parameters by trial and error until the measured error probability was the same as that of the implementation of our method we are comparing with. The structure of a node is the following:

Each node whose maximum level is L has L adjacency list (one per level), which are also stored in disk sectors (we allocate whole sectors to each list, even if the list doesn't fill it, favoring speed over disk occupancy); on the other hand, several nodes can be stored in the same sector. Since the algorithm tends to access nearby nodes sequentially, we allocate nodes using a KD-tree so that nearby nodes tend to be in the same sector. Note that the KD-tree is not part of the same algorithm, it is only used to determine the node allocation in disk sectors.

B.2 Inverted Multi-Index

In this case, the feature space $\mathbb{R}^D$ is divided in two parts isomorphic to $\mathbb{R}^{\frac{D}{2}}$ each, and each vector is represented accordingly as the Cartesian product of two vectors, $v = [v_1, v_2]$.

Each sub-space is represented by a code of size K so that each point in the feature space is represented as a pair of code vectors $[u_i, v_j]$. The pair has associated a list of the data base points that belong to the Voronoi polygon of $[u_i, v_j]$. The value of K is chosen so that each of these lists is relatively small (tens to hundreds of points). A query q is similarly divided as $[q', q'']$. The closest codes in the two subspaces are found and a suitable number of pairs $[u_i, v_j]$ is analyzed until the lists associated to them have a high probability to yield the nearest neighbor.

We determine the code books (the time to build the index is considered in our comparisons) and store them in a KD-tree to speed up the search for the closest pair. The associated lists are stored in sequential order to fill up clusters, in order to reduce access time.

The structure of the index is the following. The two code books $\{u_i\}$ and $\{v_j\}$ are stored in two KD-trees. The index grid (Figure 2 of [5]) is divided in sub-squares, each one fitting in a disk cluster:

The lists are composed of points of the data base packed in cluster, as in the previous case, each cluster contains elements from a single list. The parameters (especially K and T, see the original paper) were set by trial and error to obtain, for each data base size, the same probability of error as the method with which we are comparing.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Aggarwal CC, Philip SY (2000) The IGrid index: Reversing the dimensionality curse for similarity indexing in high dimensional space. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 119–29
2. Andoni A, Indyk P (2006) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: null. IEEE, pp 459–68
3. Arandjelović R, Zisserman A (2014) Extremely low bit-rate nearest neighbor search using a set compression tree. IEEE Trans Pattern Anal Mach Intell XX(XX):XX
4. Arya S, Mount DM, Netanyahu NS, Silverman R, Angela YW (1998) An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J ACM (JACM) 45(6):891–923
5. Babenko A, Lempitsky V (2014) The inverted multi-index. IEEE Trans Pattern Anal Mach Intell 37(6):1247–60
6. Baeza-Yates R (1997) Searching: An algorithmic tour. In: Ken A, Williams J (eds) Encyclopedia of computer science and technology. Marcel Dekker, New York, pp 331–59
7. Baeza-Yates R, Cunto W, Manber U, Wu S (1994) Proximity matching using fixed-query trees. In: Proceedings of the 5th combinatorial pattern matching (CPM '94), vol 807 of lecture notes in computer science. Springer, pp 198–212
8. Baeza-Yates R, Navarro G (1998) Fast approximate string matching in a dictionary. In: Proceedings of the 5th South american symposium on string processing and information retrieval (SPIRE -98), pp 14–22
9. Bayer R, McCreight E (1970) Organization and maintenance of large ordered indices. Technical Report D1-82-0989. Mathematical and Information Sciences Laboratory, Boeing Scientific Research Laboratories
10. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The r^*-tree: An efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD international conference on the management of data
11. Bentley JL (1990) K-D trees for semidynamic point sets. In: Proceedings of the Sixth ACM annual symposium on computational geometry
12. Berchtold S, Keim D, Kriegel HP (1996) The X-tree: An index structure for high-dimensional data. In: Proceedings of the 22nd very large data bases conference (VLDB), pp 28–39
13. Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When is "nearest neighbor" meaningful? In: International conference on database theory. Springer, pp 217–35
14. Bozkaya T, Özsoyoglu M (1997) Distance-based indexing for high-dimensional metric spaces. In: Proceedings of the 1997 ACM SIGMOD conference on management of data, pp 357–68

15. Burkhard W, Keller R (1973) Some approaches to best-match file searching. Communications of the ACM 16(4):230–6
16. Burkhard W (1983) Interpolation-based index maintenance. In: Proceedings of the ACM symposium on principles of database systems (PODS), pp 76–89
17. Chávez E, Marroquin JL, Navarro G (2001) Fixed queries array: A fast and economical data structure for proximity searching. Multimedia Tools and Applications 14(2):113–35
18. Chávez E, Navarro G, Baeza-Yates R, Marroquín JL (2001) Searching in metric spaces. ACM Computing Surveys (CSUR) 33(3):273–321
19. Chiueh T (1994) Content-based image indexing. In: Proceedings of the 20th very large data bases conference (VLDB), pp 582–93
20. Ciaccia P, Patella M, Zezula P (1997) M-tree: An efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd VLDB (Very Large Data Bases) conference, Athens, Greece
21. Ciaccia P, Patella M (2000) Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In: Proceedings if the international conference on data engineering (ICDE)
22. Cobzaş Ş, Miculescu R, Nicolae A (2019) Lipschitz functions, vol 2241. Springer, Berlin
23. Cox T, Cox M (1994) Multidimensional Scaling. Chapman & Hall, London
24. Cui B, Ooi BC, Su J, Tan K-L (2004) Main memory indexing: The case for BD-tree. IEEE Transactions on Knowledge & Data Engineering 7:870–4
25. Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. Journal of the American Society for Information Science 41(6):391–407
26. Flickner M, Sawhney H, Niblack W, Ashley J, Huang Q, Dom B, Gorkani M, Hafner J, Lee D, Petkovic D, Steele D, Yanker P (1995) Query by image and video content: The QBIC system. IEEE Computer
27. Ge T, He K, Ke Q, Sun J (2013) Optimized product quantization for approximate nearest neighbor search. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2946–53
28. Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of the 25rd VLDB (Very Large Data Bases) conference, Athens, Greece
29. Gromov M (1999) Metric strucures for Riemann and non-Riemann spaces, volume 152 of Progress in Mathematics. Birkhauser Verlag
30. Guttman A (1984) R-trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD international conference on the management of data, pp 47–57
31. Hoel EG, Samet H (1993) Data-parallel R-tree algorithm. In: Proceedings of the 23rd international conference on parallel processing, pp 49–53
32. Huang Q, Feng J, Zhang Y, Fang Q, Ng W (2015) Query-aware locality-sensitive hashing for approximate nearest neighbor search. Proceedings of the VLDB Endowment 9(1):1–12
33. Iscen A, Furon T, Gripon V, Rabbat M, Jégou H (2018) Memory vectors for similarity search in high-dimensional spaces. IEEE Transactions on Big Data 4(1):65–77
34. Johnson NL, Kotz S, Balakrishnan N (1994) Continuous univariate distributions. Wiley, New York
35. Johnson W, Lindenstrauss J (1984) Extensions of Lipschitz maps into a Hilbert space. Contemporary Mathematics 26:189–206
36. Kamel I, Faloutsos C (1992) Parallel R-trees. Technical Report CS-TR-2820. Univesity of Maryland, College Park
37. Kazukawa D, Shioya T (2020) High-dimensional ellipsoids converge to gaussian spaces. arXiv preprint arXiv:2003.05105
38. Korn F, Pagel B-U, Faloutsos C (2001) On the Dimensionality curse and the Self-similarity blessing. IEEE Transactions on Knowledge and Data Engineering 13(1)
39. Kumar A (1994) G-tree: A new data structure for organizing multidimensional data. IEEE Transactions on Knowledge and Data Engineering 6(2):341–7
40. Ledoux M (2001) The concentration of measure phenomenon, volume 89 of Mathematical surveys and monographs. American Mathematical Society, Providence
41. Ke L, Malik J (2016) Fast k-nearest neighbour search via dynamic continuous indexing. In: International conference on machine learning, pp 671–9
42. Ke L, Malik J (2017) Fast k-nearest neighbour search via prioritized DCI. In: Proceedings of the 34th international conference on machine learning, vol 70, pp 2081–90
43. Li M, Zhang Y, Sun Y, Wang W, Tsang IW, Lin X (2018) An efficient exact nearest neighbor search by compounded embedding. In: International conference on database systems for advanced applications. Springer, pp 37–54
44. Liu T, Moore AW, Ke Y, Gray AG (2005) An investigation of practical approximate nearest neighbor algorithms. In: Advances in neural information processing systems, pp 825–32

45. Malkov YA, Yashunin DA (2018) Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence
46. Milman VD, Schechtman G (1986) Asymptotic theory of finite-dimensional normed spaces (with an Appendix by M. Gromov), vol 1200. Springer, Berlin
47. Moore AW (2000) The anchors hierarchy: Using the riangle inequality to survive high-dimensional data. In: Twelfth conference on uncertainty in artificial intelligence. AAAI Press
48. Muja M, Lowe DG (2009) Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1) 2(331-40):2
49. Nene S, Nayar S (1997) A simple algorithm for nearest neighbor search in high dimensions. IEEE Trans Pattern Anal Mach Intell 19(9):989–1003
50. Nievergelt J, Hingerberger H (1984) The grid file: An adaptable, symmetric multikey file structure. ACM Transactions on Database Systems 9(1):38–71
51. Omohundro SM (1987) Efficient algorithms with neural network behavior. Journal of Complex Systems 1(2):273–347
52. Pestov V (2007) Intrinsic dimension of a dataset: What properties does one expect? In: Proceedings of the 22nd inernational joint conference on neural network, pp 1775–80
53. Pestov V (2008) An axiomatic approach to intrinsic dimension of a data set. Neural Networks 21:204–13
54. Prabhakar S, Aggrawal D, Abbadi AE (1998) Efficient disk allocation for fast similarity searching. In: Proceedings of ACM SPAA '98
55. Sellis T, Roussopoulos N, Faloutsos C (1987) The R+ tree: A dynamic index for multidimensional objects. In: Proceedings of the 13th very large data bases conference (VLDB)
56. Slaney M, Casey M (2008) Locality-sensitive hashing for finding nearest neighbors [lecture notes]. IEEE Signal Processing Magazine 25(2):128–31
57. Sun Y, Wang W, Qin J, Zhang Y, Lin X (2014) Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. Proceedings of the VLDB Endowment 8(1):1–12
58. Thurstone LL (1927) A law of comparative judgement. Psychological Review 34:273–86
59. Warren S (1965) Torgerson. Multidimensional scaling of similarity. Psychometrika 30:379–93
60. Uhlmann J (1991) Satisfying general proximity/similarity queries with metric trees. Inform Process Lett 40:175–9
61. Weber R, Schek HJ, Blott S (1998) A quantitative analysis and performance study for similarity search in high-dimensional spaces. In: Proceedings of the 24th VLDB (Very Large Data Bases) conference, New York, USA, pp 194–205
62. White D, Jain R (1996) Similarity indexing with the SS-tree. In: Proc 12th IEEE international conference on data engineering
63. Xiao J, Hays J, Ehinger KA, Oliva A, Torralba A (2010) Sun database: Large-scale scene recognition from abbey to zoo. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 3485–3492
64. Yianilos P (1999) Excluded middle vantage point forests for nearest neighbor search. In: DIMACS implementation challenge, ALENEX -99
65. Zhong Y (2015) Efficient similarity indexing and searching in high dimensions. arXiv preprint arXiv:1505.03090