Information Systems 101 (2021) 101808

Contents lists available at ScienceDirect

Information Systems

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

Query filtering using two-dimensional local embeddings

Lucia Vadicamo [a], Richard Connor [b,*], Edgar Chávez [c]

[a] Institute of Information Science and Technologies (ISTI), CNR, Pisa, Italy
[b] School of Computer Science, University of St Andrews, St Andrews, Scotland, United Kingdom
[c] Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), Ensenada, Mexico

ARTICLE INFO

Article history:
Received 14 March 2020
Received in revised form 4 January 2021
Accepted 17 May 2021
Available online 21 May 2021
Recommended by Gottfried Vossen

Keywords:
Metric search
Extreme pivoting
Supermetric space
Four-point property
Pivot based index

ABSTRACT

In high dimensional data sets, exact indexes are ineffective for proximity queries, and a sequential scan over the entire data set is unavoidable. Accepting this, here we present a new approach employing two-dimensional embeddings. Each database element is mapped to the *XY* plane using the four-point property. The caveat is that the mapping is local: in other words, each object is mapped using a different mapping.

The idea is that each element of the data is associated with a pair of reference objects that is well-suited to filter that particular object, in cases where it is not relevant to a query. This maximises the probability of excluding that object from a search. At query time, a query is compared with a pool of reference objects which allow its mapping to all the planes used by data objects. Then, for each query/object pair, a lower bound of the actual distance is obtained. The technique can be applied to any metric space that possesses the four-point property, therefore including Euclidean, Cosine, Triangular, Jensen–Shannon, and Quadratic Form distances.

Our experiments show that for all the data sets tested, of varying dimensionality, our approach can filter more objects than a standard metric indexing approach. For low dimensional data this does not make a good search mechanism in its own right, as it does not scale with the size of the data: that is, its cost is linear with respect to the data size. However, we also show that it can be added as a post-filter to other mechanisms, increasing efficiency with little extra cost in space or time. For high-dimensional data, we show related approximate techniques which, we believe, give the best known compromise for speeding up the essential sequential scan. The potential uses of our filtering technique include pure GPU searching, taking advantage of the tiny memory footprint of the mapping.

1. Introduction

In general, metric indexes partition the data on the basis of the distances to one or more reference objects (*pivots*) so that, at query time, some partitions can be included or excluded from the search without the need to calculate the actual distances between the query and the data objects within that partition. The triangle inequality used together with the knowledge of the distances between the pivots and the data/query objects allows computing upper and lower bounds for these distances.

The concept of *local pivoting* is to partition a metric space so that each element in the space is associated with precisely one of a fixed set of pivots. The idea is that each object of the data set is associated with the pivot that is best suited to filter that particular object if it is not relevant to a query, maximising the probability of excluding it from a search. The notion does not in itself lead to a scalable search mechanism, but instead gives a good chance of exclusion based on a tiny memory footprint and a fast calculation. It is therefore most useful in contexts where main memory is at a premium, or in conjunction with another, scalable, mechanism.

In this paper we apply similar reasoning to metric spaces that possess the *four-point property* [1], which notably include Euclidean, Cosine, Triangular, Jensen–Shannon, and Quadratic Form spaces. This property allows computing bounds for the actual distance that are tighter that than that obtained using the triangle inequality [2]. We show a novel way of exploiting this situation: each element of the space can be associated with two reference objects, and a four-point lower-bound property is used instead of the simple triangle inequality. The probability of exclusion is strictly greater than with simple local pivoting; the space required per object and the calculation are again tiny in relative terms. Specifically, we store each object using a tuple of four values, as follows. From a finite metric space S, a relatively small set of reference objects $\mathcal{P}$ is selected. For all $p_j, p_k \in \mathcal{P}$, the distance $d(p_j, p_k)$ is calculated and stored. For each element s_i

* Corresponding author.
E-mail addresses: lucia.vadicamo@isti.cnr.it (L. Vadicamo), rchc@st-andrews.ac.uk (R. Connor), elchavez@cicese.mx (E. Chávez).

https://doi.org/10.1016/j.is.2021.101808
0306-4379/

in S, a single pair of reference objects $\langle p_{i_1}, p_{i_2}\rangle$ is selected. The distances $d(s_i, p_{i_1})$ and $d(s_i, p_{i_2})$ are calculated and used together with $d(p_{i_1}, p_{i_2})$ to *isometrically project* the objects p_{i_1}, p_{i_2}, s_i in the 2D Euclidean vectors $(0, 0)$, $\big(0, d(p_{i_1}, p_{i_2})\big)$, (x_{s_i}, y_{s_i}) respectively. The triangle inequality guarantees that such isometric embedding exists; moreover, the coordinates of vector (x_{s_i}, y_{s_i}) can be easily computed by exploiting the distances to the selected pivots (see Section 3 for the details). Thus the space S is represented as a set of tuples $\langle i_1, i_2, x_{s_i}, y_{s_i}\rangle$, indexed by i, therefore requiring only a few bytes per object.

When a query is executed, the distances $d(q, p_j)$ for each $p_j \in \mathcal{P}$ are first calculated. At this point, considering any $s_i \in S$ and the objects q, p_{i_1}, p_{i_2}, it is possible to compute a lower-bound for the unknown distance $d(q, s_i)$ with a cheap geometric calculation, without any requirement to access the original value $s_i \in S$. By exploiting the knowledge of the distances to the pivots p_{i_1}, p_{i_2} it is possible to compute the 2D projection $(x_{q,i}, y_{q,i})$ of the point q with respect to the pivots p_{i_1}, p_{i_2}. The four-point property guarantees that the Euclidean distance between $(x_{q,i}, y_{q,i})$ and (x_{s_i}, y_{s_i}) is a lower-bound for the actual distance $d(q, s_i)$.

We show that the resulting mechanism can be very effective. However, for a selection of m reference points, there exist $\binom{m}{2}$ pairs from which the representation of each data point can be selected. This number, of course, becomes rapidly very large even with modest increases in m. If for each element of S we can find a particularly effective pair p_{i_1}, p_{i_2}, within this large space, then this tiny representation of S can be used as a powerful threshold query filter. This exclusion mechanism leads to a sequential scan, which is virtually unavoidable in light of a recent conditional hardness result in [3] for nearest neighbour search, even in the approximate setup, for every $\delta > 0$ there exist constants $\epsilon, c > 0$ such that with preprocessing time $O(N^c)$ computing a $(1+\epsilon)$-approximation to the nearest neighbour requires $O(N^{1-\delta})$ time, with N the size of the database.

The above hardness result has been suspected for a long time by the indexing community, and it has been named the *curse of dimensionality*. It is known, for example, that a metric inverted index [4] has high recall rates only if a substantial part of the candidate results is revised. We aim our approach at this final part of query filtering or re-ranking.

The contributions of this paper are as follows:

1. We show that the outline mechanism is viable. For SISAP benchmark data sets [5] we show that exclusion rates of over 98% can be achieved using our small memory footprint and cheap calculations.
2. We use an observation of the mechanism applied in much higher-dimensional spaces which leads to two different approximate mechanisms which can be applied to *range* and *nearest-neighbour* search respectively. For both mechanisms, for a space which is completely intractable for metric indexing methods we can achieve a reduction of search cost of around 90%, in order to return around 90% of the correct results.
3. We examine the problem of finding the best pair of reference points per datum; this can be done well, but expensively, by an exhaustive search of the pair space; however the cost of this is quadratic with respect to the number of reference objects selected. We show that much cheaper heuristics are also effective.
4. Finally, we show an example of how the mechanism can be used as a post-filter adjunct to another mechanism. We describe its incorporation with the *List of Clusters* index. Using a pragmatic selection of reference objects it can be ensured that no new distances are measured at either construction or query time, which can nonetheless lead to a halving of the overall query cost.

Table 1
Notation used throughout this paper. Some concepts, such as planar projection, are introduced in later sections.

Symbol	Definition
$\lvert\cdot\rvert$	Size of a set
(U, d)	Metric space
$S \subseteq U$	Finite search space (database)
$N = \lvert S\rvert$	Size of the database
$o, s_i \in S$	Data objects
$q \in U$	Query object
$\mathcal{P} = \{p_1, \ldots, p_m\}$	Set of pivots (reference objects), $p_j \in U$
m	Number of pivots
t, t_1, t_2	Threshold distance, e.g. radius used in a range query search
k	Number of results of a nearest neighbour search
ℓ_2	Euclidean distance
$\langle p_{i_1}, p_{i_2}\rangle$	Pair of pivots used as reference points for the object s_i
$(x_{s_i}, y_{s_i}) \in \mathbb{R}^2$	Planar projection of the point s_i with respect to the pivots p_{i_1}, p_{i_2}
$\sigma(s_i) = \langle i_1, i_2, x_{s_i}, y_{s_i}\rangle$	Tuple used to represent the data point s_i
$(x_{q,i}, y_{q,i}) \in \mathbb{R}^2$	Planar projection of the point q with respect to the reference objects selected for the object s_i

A preliminary version of this work appeared in [6]. The present contribution gives a more detailed description of the proposed approach and a widely extended experimental evaluation. Moreover, it investigates the use of our approach also for approximate pre-filtering and ranked order nearest-neighbour queries.

The rest of the paper is structured as follows. Section 2 reviews related work and gives background information on the four-point property and lower-bound. Section 3 discuss properties of the planar projection used in this work to map metric data to 2D Euclidean space. Section 4 presents our proposed search strategies that rely on local embedding of the data into 2D coordinate space. Section 5 present results of a thorough experimental analysis of the proposed approaches. Section 6 draws conclusions. Table 1 summarises the notation used in this paper.

2. Background and related work

Pivot based indexes have populated the metric indexing scene for a long time. A standard approach is creating a *pivot table*, obtained by pre-computing and storing the distances between data objects and some pivots (reference objects). Each object is then represented as the vector of its distances to the pivots. Therefore, a pivot table is just the direct product of one dimensional projections obtained from a single pivot at a time. The *object-pivot distance constraint* [7], which is a direct consequence of the triangle inequality, ensures that each coordinate gives a lower bound to the actual distance from database points to the query. The lower bounds are used to exclude objects from the searching process without explicitly computing their distance to the query. For example, exclusion occurs when the distance between the object and the pivot, which is pre-calculated, and the distance between the query and the pivot, which is calculated at the start of a query evaluation, give a lower bound for the actual distance between the query and the object that is greater than a given threshold. A safe choice is to take the maximum over all the available lower bounds. In other words, given a set of pivots $\{p_1, \ldots, p_m\}$, an object o, and a query q, the $\max_j |d(o, p_j) - d(q, p_j)|$ is a lower-bound for the distance $d(o, q)$.

The most effective algorithm published for searching using a pivot table is AESA [8], which proceeds as following. All the $O(N^2)$ distances between every object in the database of N elements

is pre-computed. With this, every object in the database is a potential pivot. At query time a subset of the N pivots is selected, one at a time, using a heuristic which consist in selecting the $j+1$ pivot, the closest to the query, using as bound the j pivots known so far and the first pivot at random. The output of this heuristic is both a set of *good* pivots for the query, and the nearest object to it. Two things can be noticed from this basic approach, first, the number of pivots actually used is much smaller than N, and second, they are tailored for each query on the fly. However since the space usage is quadratic, the approach is impractical. Also notice that a sequential scan is implied to obtain the closest next pivot in the interaction. Linear space approaches of the same idea were used in [9], that stores distances from objects to only a fixed number m of pivots. The search procedure is nearly the same of AESA, except that not all the objects are used as pivots. A better heuristic for selecting the next pivot is proposed in [10]. The sequential scan can be avoided using a tree [11,12].

Selecting the best pivot for a given query is not possible offline. A weaker alternative is to select the best pivot for each database object, increasing the probability of exclusion at query time. Two options have been explored in the literature, in [13] each pivot in the pool only keep distances to objects in the extreme of the distribution, those objects near and far the pivot. This process is sub-optimal and may end with a few objects guarded by many pivots, and many objects guarded by a few or none pivots. A second alternative, ensuring some fairness in the coverage, was proposed in [14], this time each object can select the best pivot. This latter approach is called *extreme pivoting*. In those heuristics the gain is in filtering power, when the amount of available memory is fixed. Pivot tables are useful for post filtering in a hierarchical metric index, as in [15], or they can be used as a stand alone index using directly the table as in [16,17].

For post-filtering, when a primary index is applied to filter the data and only a small fraction of the database should be checked against the original metric, a table is useful. A high rate of exclusion will prevent the use of the more expensive distance computations, and moreover will require to fetch a smaller number of objects from secondary memory. Hence a small table, with just a couple of coordinates, is an excellent trade-off because it can be kept in main memory. In the same spirit as the extreme pivots for unidimensional mapping, in this paper we are aiming at building a table of small memory footprint using the so-called *four point property*.

2.1. The four-point property and supermetric spaces

Between the end of the 19th century and the beginning of the 20th century various mathematicians, including Menger [18], Wilson [19] and Blumenthal [20], founded the basis of the *distance geometry* that studies and characterises semimetric spaces based only on given values of the distances among finite subsets of points. A basic observation is that the triangular inequality can be expressed as a geometric condition: a semimetric space has the triangle inequality if and only if any three points of the space can be projected in a 2D Euclidean space while preserving all the three inter-point distances. In other words, a triangle can be drawn in a 2D plane so that the lengths of its edges are in one-to-one correspondence with the pairwise distances between the points. This property is also called the *three-point property* and the spaces that satisfy it are referred to as *isometrically 3-embeddable in 2-dimensional Euclidean space*. It turns out that a large class of metric spaces also satisfy the *four-point property*, i.e. they are *isometrically 4-embeddable in* 3*-dimensional Euclidean space* [1,21,22]. In a nutshell, for any four points of the space there exists an isometric embedding that maps those points to the vertices of a tetrahedron so that all the six pairwise distances between the points are preserved.

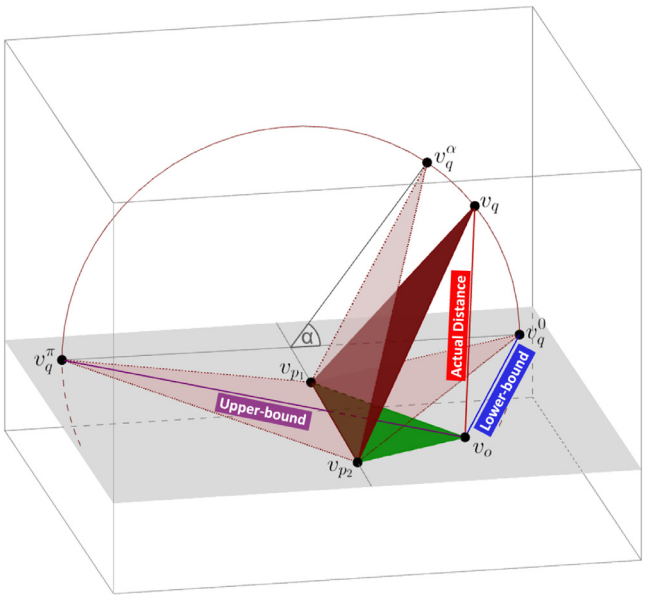

Fig. 1. Example four-point based projection. Two reference objects p_1, p_2 and two data objects o, q are first isometrically embedded in 3D Euclidean space. The mapped points are indicated with $v_{p_1}, v_{p_2}, v_o, v_q$ in the graph. By rotating v_q around the edge $\overline{v_{p_1} v_{p_2}}$ we obtain the points v_q^α where $0 \leq \alpha \leq \pi$ is the angle formed with the triangle $\triangle v_{p_1} v_{p_2} v_o$. A planar projection is obtained by setting $\alpha = 0$ or $\alpha = \pi$. The euclidean distances $\ell_2(v_q^0, v_o)$ and $\ell_2(v_q^\pi, v_o)$ are respectively a lower bound and an upper bound for the actual distance $d(o, q)$.

More recently we have applied these results in theoretical mathematics to the practical domain of metric search [2,23,24] and we coined the term *supermetric* to refer to spaces with the four-point property as, in terms of metric search, they are significantly more tractable. For this context, the important result is that the four-point property applies to many commonly-used distance metrics, including Euclidean, Cosine,[1] Jensen–Shannon, Triangular and Quadratic Form distances, all of which can be safely used in conjunction with the mechanisms described here.

One crucial outcome of our work is that in the context of pivot-based techniques the four-point property allow us to compute upper and lower bounds of the actual distance that are tighter than those obtained using the triangle inequality.

2.1.1. The four-point planar lower bound

For two points that have not been directly compared, q and o, it is shown in [2] how a lower bound of their distance can be established by comparing the distances between both points and two further reference points p_1 and p_2. In particular, thanks to the four-point property, we know that there exists an isometric embedding of the points q, o, p_1, p_2 into a 3-dimensional Euclidean space. The projected points v_q, v_o, v_{p_1}, $v_{p_2} \in \mathbb{R}^3$ form the vertices of a tetrahedron for which we know the edge lengths of two adjacent faces, i.e. the triangles $\triangle v_{p_1} v_{p_2} v_o$ and $\triangle v_{p_1} v_{p_2} v_q$ with the common base $\overline{v_{p_1} v_{p_2}}$. Because of the four-point property, the unmeasured distance $d(q, o)$ must form the sixth edge of the tetrahedron.

Since translations, rotations, reflections are isometric transformations, without loss of generality, we can assume that the triangle $\triangle v_{p_1} v_{p_2} v_o$ lies on the plane $\{Z = 0\}$, the points v_{p_1}, v_{p_2} are plotted on the X-axis (say at positions $(0, 0, 0)$ and $(d(p_1, p_2), 0, 0)$ respectively), the point v_o is above the X-axis, and the vertex v_q is above the plane $\{Z = 0\}$ (see for example Fig. 1). By knowing the

[1] For the correct formulation, see [23].

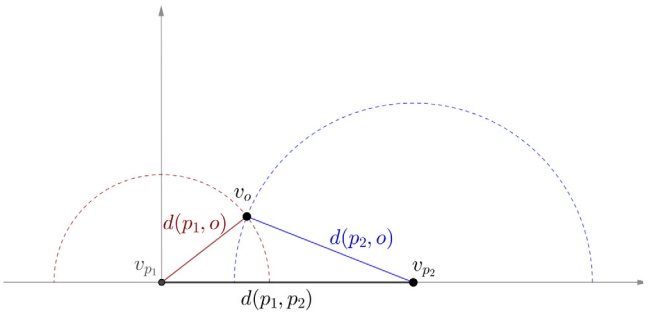

Fig. 2. Planar projection of a point o with respect to two pivots p_1, p_2.

distances $d(p_1, o)$ and $d(p_2, o)$ we can easily compute the coordinate of the vertex v_o as the intersection of two circumferences in the plane $\{Z = 0\}$. However, even if we know the distance $d(p_1, q)$ and $d(p_2, q)$ we cannot explicitly compute the coordinates of the vertex v_q because $d(q, o)$ is unknown. In facts, without measuring the actual distance $d(q, o)$ we cannot determine which is the exact angle formed between the triangle $\triangle v_{p_1} v_{p_2} v_q$ and the semi-plane containing the other triangle $\triangle v_{p_1} v_{p_2} v_o$. Nevertheless, the key observation here is that by rotating these triangles around the common base until they lie in the same plane (i.e. assuming that the unknown angle is equal to 0 or π) we can determine upper and lower bounds for the actual distance $d(q, o)$. In particular, by indicating with v_q^0 the projection of the point q in the same semiplane of v_p we have that $\ell_2(v_q^0, v_o)$ is a lower bound of $d(q, o)$. Other choices are possible: given a fixed angle α we can compute the coordinate of the vertex v_q^α so that the distances to the reference points are preserved and the angle formed by the two triangles is α. In that case we can use the distance $\ell_2(v_q^\alpha, v_o)$ as an approximation of $d(q, o)$ since it is a measure between the upper and the lower bounds of the actual distance. For example, in [25,26] the projection based on $\alpha = \pi/2$ is effectively used on different sets of data and search scenarios.

3. Planar projection and distribution of values in the 2D plane

In this work, we use the planar projection based on $\alpha = 0$, that is we project all the points in the same 2D plane, since this choice guarantees that the Euclidean distance between two projected objects is a lower-bound of the actual distance, and thus can be safely used for filtering purposes. The value of this is that, independently of the size of individual data values and the cost of the distance metric, any value can be represented, for a fixed choice of reference points, as a small 2D coordinate, and compared using 2D Euclidean distance. The result of this comparison, being a lower bound of the true distance, may mean that there is no requirement for the full comparison to be made. Of course, the value of the method depends heavily upon the probability of its success.

To visualise this property we use scatter diagrams constructed as follows. The two selected reference points p_1, p_2 are plotted on the X-axis according to the distance between them, and a data set is represented as points in the 2D space plotted above the X-axis, according to their respective distances from these reference points (for example see Fig. 2). The triangle inequality property is necessary to guarantee the ability to create such a plot. Specifically, the pivots are mapped to the points $v_{p_1} = (0, 0)$ and $v_{p_2} = (0, d(p_1, p_2))$. Any further data point o is mapped to $v_o = (x_o, y_o)$ using simple geometry, i.e.

$$x_o = \frac{d(o, p_1)^2 - d(o, p_2)^2}{2 \cdot d(p_1, p_2)} + \frac{d(p_1, p_2)}{2} \tag{1}$$

$$y_o = \sqrt{d(o, p_1)^2 - x_o^2} \tag{2}$$

Fig. 3 shows two versions of such a scatter plot created from a 10 dimensional Euclidean space, using the same data and reference points. Although the triangle inequality property guarantees the ability to create such a plot, the relationship among the plotted points is more subtle.

An example point s is selected from the centre of the diagram, coloured blue. For every other point plotted in the plane, we then consider whether it *might* be within a threshold distance t from this, based only on the distances calculated within the projected 2D plane. Here we have chosen $t = 0.24$, representing around one-millionth of the volume of the 10D space.

The diagrams are colour-coded so that those points which *may* be within that distance, i.e. those that cannot be excluded from a search, are highlighted, plotted in yellow. The four-point planar lower bound is illustrated on the right-hand side, clearly represented by a simple exclusion radius in the 2D plane (*Hilbert exclusion* [23]). On the left-hand side, only the triangle inequality property is used, giving much wider bounds (*hyperbolic exclusion*). Note that in this example, as the mapped metric space has the four-point property, the right-hand exclusion boundary is a guarantee. It is quite possible that the same diagram could be drawn using an underlying metric space which does not have the property, in which case the left-hand exclusion boundary must be used to guarantee correctness.

To apply these observations towards our goal, we note that if the blue-coloured point represents a datum, and each of the black points represents an independent query q, then it would be possible to filter the datum from nearly 70% of these queries based on this particular 2D projection. The only information required to test for this is the 2D position of the datum within the projection, and the two distances $d(q, p_1)$ and $d(q, p_2)$ where p_1 and p_2 are the reference objects upon which the projection is based. The next observation is that, if the blue data point was selected from nearer the periphery of the scatter, the proportion of filtered queries would be higher, as the density of scattered points is lower.

This observation leads to our outline strategy, as follows. For a set of m reference objects drawn from a supermetric space (U, d), a set of $\binom{m}{2}$ pivot pairs, and therefore 2D projections, exist. It is therefore possible to characterise a very large number of projections using a relatively small number of reference objects. For each element of the finite space S a "good" projection is determined, i.e. one that gives a relatively good filter ratio from a representative set of queries. Each datum in S can then be represented only by the identity of these pivots, and its XY coordinates under the projection defined by them. At query time, the m pivot distances are calculated once per query; then, for each element s_i of S, it may be possible to avoid the distance calculation $d(q, s_i)$ with reference to only these distances and the XY coordinates of s_i under the chosen projection.

3.1. Choice of reference objects

It appears that the *distribution* of objects in the 2D plane with respect to a given pair of reference points is fairly predictable. However, where *individual* data points land within the scatter varies widely.

Figs. 4 and 5 show diagrams to illustrate this. In each figure, a single set of data objects is plotted in the XY plane according to their distances from two randomly-selected reference points; the left and right sides of each figure represent the same data plotted against a different pair of reference points. The data is

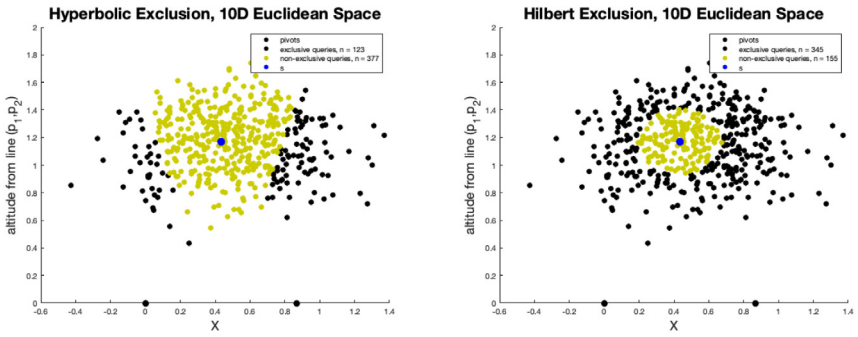

Fig. 3. 500 points from a generated Euclidean space plotted against randomly selected reference points. Left and right plots show the exclusion potential based on simple metric (left) and supermetric (right) properties for the datum s near the centre of the diagram. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

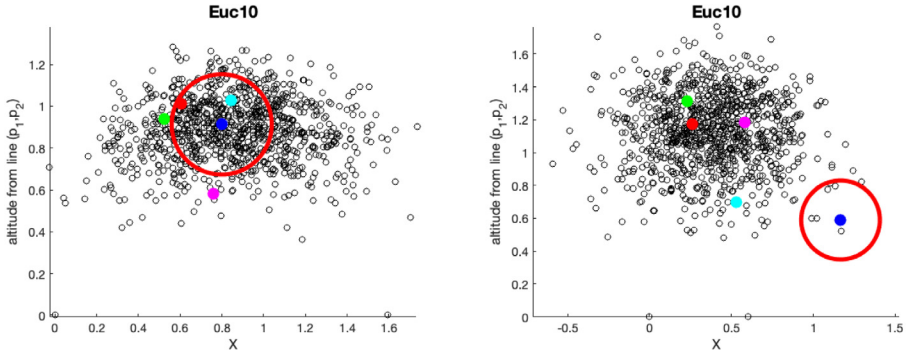

Fig. 4. The same 1000 points plotted in the 2D plane based on two different projections. The colour-coded points represent the same values under the different projections. The (X, Y) scatter in both cases is close to an uncorrelated normal distribution on each axis, but it can be seen that the mapping of an individual point is quite independent. The red circle represents a typical query threshold around the blue point: this data point cannot be a solution to any query projected outside this circle. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

taken respectively from a generated 10-dimensional Euclidean data set, and the *SISAP colors*[2] benchmark set.

In the two figures, a random selection of five data points has been made and these are highlighted in colour; that is, the coloured spots in the left and right sides of the figure represent the same data point mapped according to the different pair of reference points. It can be seen there is no strong relationship in the positions where the different coloured dots are plotted. This is to be expected, as much of the spatial relationship present in the higher dimensions is necessarily lost when projecting into only two dimensions.

An underlying hypothesis in our work is that the distribution of queries within a metric space (U, d) will be similar to the distribution of a representative selection taken from the finite space (S, d) to be queried. Thus, looking at the scatter diagrams in Figs. 4 and 5, the plotted points could represent the distribution of either data or queries with respect to those same reference points. The probability of successful elimination, for a given $q \in U$ and $s_i \in S$, therefore depends upon the choice of reference points, and the relative position of both s_i and q with respect to them. If the hypothesis is correct, then the notion of a "good" pair of reference points for an individual $s_i \in S$ corresponds to the (inverse) *density* of the 2D region where s_i lands, within a representative set. The density around a point can be measured precisely with the local intrinsic dimension [27] or the reverse K-nearest neighbours [28]. Both heuristics are costly and have additional parameters, a simpler method used here is to count the number of datum inside a ball around the sample being tested. If query and datum lie further than the query threshold within the 2D plot, then the datum cannot be a solution to the query; this is most likely to occur when either query or datum lie within a sparsely populated region of the plane. However of course only the datum is available for pre-processing. If queries and data follow the same distribution patterns, then the best pair of reference points per datum can be selected with reference to a representative set of data points from within S.

In Figs. 4 and 5, the reference point pair used for the right-hand side of each figure has been selected to maximise the "goodness" of the scatter with respect to one of the plotted objects, coloured in blue. If the other objects plotted in each case are regarded as representative queries, then it can be seen that the filtering power of the reference points used for the right-hand diagrams in each case is far greater than that of those used for the left-hand diagram, and these reference objects are therefore a good choice for this particular datum. To show the concept is quite general, Fig. 6 shows the same data as Fig. 4, with further choices of reference object pairs chosen to give good scatters for two of the other coloured objects.

To construct these figures,[3] a random selection of 150 reference points was made, and each of the 11,175 (i.e. $\binom{150}{2}$) possible pairs was tested; the selection was made according to the density of 2D mapping around the area of the object in question. This technique gives unquestionably good results, but is of course very expensive as the selection must be made independently for each element of the set S which will be queried. The question of how to make a good pragmatic choice of reference objects per datum is discussed later in more detail.

[2] See Section 5.

[3] MatLab code is at https://bitbucket.org/richardconnor/low_dim_embedings_is.

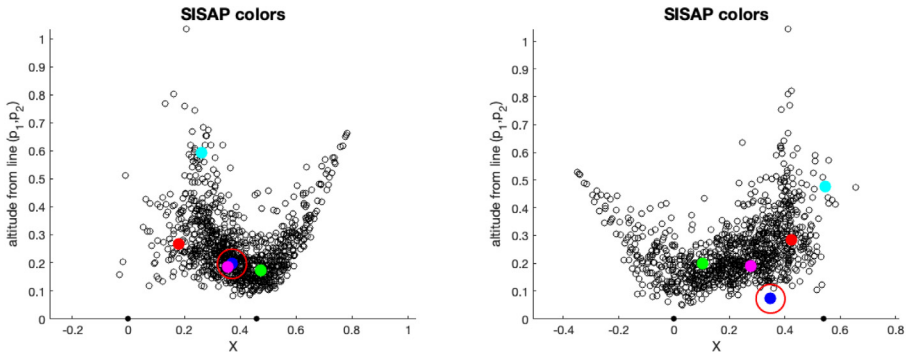

Fig. 5. 1000 elements from the *colors* data set are projected onto a 2D plane using different pairs of reference points. The different coloured spots in each diagram show how the same data elements are projected differently with a different choice of reference objects. The red ring depicts a near-neighbour threshold distance around the blue point: the right-hand projection is much better for this element as 99.8% of the set is projected outside this boundary, and can therefore allow filtering, as opposed to 85.2% in the left-hand projection. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

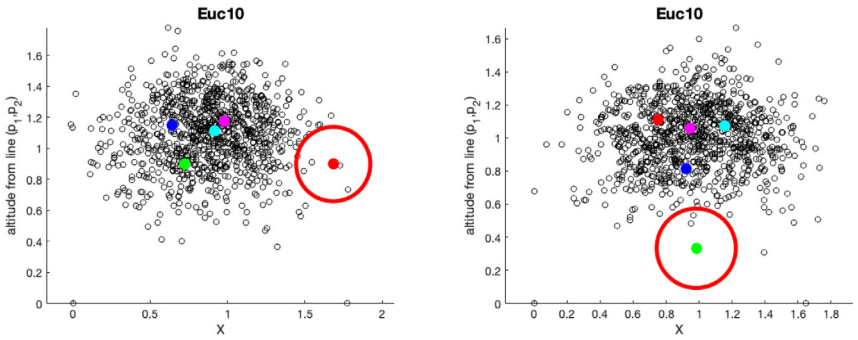

Fig. 6. In these diagrams, two further projections are made over the same data as that of Fig. 4. Here the reference objects used to construct the projection are chosen to minimise the local density of the projection around the red and green highlighted points respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

One early observation however is that existing reference object selection strategies (e.g. [29,30]) are not effective in this context. Such strategies aim to find a set of reference objects that give good overall separation for all queries with respect to a large set of data, while in our mechanism we aim to find a good pair of reference objects per datum. By considering the above figures, it can already be seen that the criteria for these purposes are quite different. In Fig. 4 for example, the pair of reference objects which give the best separation for the datum are very close together, and give a much worse separation for most of the other objects in the rest of the data set.

4. Local embedding and search strategies

By exploiting the planar projection and the four-point planar lower bound, we propose a mechanism with the following properties:

- at pre-processing time:
  - for a metric space (U, d) and a finite search space $S \subset U$, we first select a distinguished set of m reference points $\mathcal{P} \subset U$. The value of m is chosen according to properties of the space, as discussed in Section 5.
  - for each of the $\binom{m}{2}$ pairs of points $p_j, p_k \in \mathcal{P}$, the distance $d(p_j, p_k)$ is calculated and stored in a lookup table
  - for each $s_i \in S$, we select a specific pair of points $\langle p_{i_1}, p_{i_2} \rangle$, where $p_{i_1}, p_{i_2} \in \mathcal{P}$, to act as reference points specifically for that value s_i – various different strategies can be used for making this choice, as discussed later
  - a 2D Euclidean projection is performed, where the object p_{i_1} is mapped to the Cartesian point $(0, 0)$, p_{i_2} is mapped to $(d(p_{i_1}, p_{i_2}), 0)$ and s_i is mapped to the unique point (x_{s_i}, y_{s_i}), $y_{s_i} \geq 0$ which preserves the distances $d(s_i, p_{i_1})$ and $d(s_i, p_{i_2})$ – note that the coordinates x_{s_i} and y_{s_i} can be easily computed using Eqs. (1) and (2) with $o = s_i$, $p_1 = p_{i_1}$, and $p_2 = p_{i_2}$
  - the representation of the value s_i, which we refer to as $\sigma(s_i)$, is the tuple $\langle i_1, i_2, x_{s_i}, y_{s_i} \rangle$.
- at query time
  - for a given query q, each distance $d(q, p_j), p_j \in \mathcal{P}$ is calculated
  - for each given value s_i which is a potential solution:
    - the tuple $\sigma(s_i)$ is retrieved
    - from the i_1, i_2 values of the tuple, the distances $d(q, p_{i_1})$ and $d(q, p_{i_2})$ are retrieved
    - in conjunction with the retrieved distance $d(p_{i_1}, p_{i_2})$, the 2D projection $(x_{q,i}, y_{q,i})$ is calculated using Eqs. (1) and (2) with p_{i_1}, p_{i_2} as pivots and $o = q$
    - the Euclidean distance $\ell_2((x_{q,i}, y_{q,i}), (x_{s_i}, y_{s_i}))$ is then a lower bound for $d(q, s_i)$, which may therefore allow s_i to be removed from any further consideration during evaluation of the query.

Note that the main cost of the mechanism at query time is the calculation of distances to the elements of $\mathcal{P}$, which is not only relatively small but, important to note, amortised over the checking of any $s_i \in S$ with respect to a single query q. The size

of $\sigma(s_i)$ is four numeric values, and independent of the size of the data. The cost of the lower-bound calculation is purely algebraic, and inconsequential compared to the typical cost of a distance calculation in a complex space.

To quantify this as far as possible, it can be noted that the cost of the filtering operation, for a given datum s and query q, is essentially the cost of assessing whether a tetrahedron can be formed from six given distances, all of which are known before the filtering calculation occurs. $d(p_1, p_2)$ is stored in a table; $d(p_1, s)$ and $d(p_2, s)$ are stored within the representation of s; $d(p_1, q)$ and $d(p_2, q)$ are evaluated once at the start of the query process, and t (the filter threshold) is pre-determined. Using distance geometry, this could be determined by evaluation of the Cayley–Menger determinant using these six distances, which will give a real (i.e. non-complex) outcome if and only if it is possible to form a tetrahedron. We have measured[4] the cost of this as a little under 0.9 μs. We optimise this by storing the (X, Y) coordinates projected for s and calculating the 2D Euclidean distance to those projected for the query, using optimised versions of Eqs. (1) and (2), which brings the cost down to around 0.3 μs. Note that this cost is independent of either the size of the original data, or the cost of the original metric. To put this in context, the cost of evaluating Euclidean distance over a 4096-dimension *fc6* layer is around 50 μs, without allowing for memory transfers, making the filtering worthwhile even if only 1% of these calculations are saved.

The mechanism is purely filtering: for any query q, datum s_i, and distance t, it may be possible to demonstrate that $d(q, s_i) > t$. However that is the extent of its value; the lower-bound values produced do not comprise a proper metric in their own right, and at the moment we do not foresee the construction of a scalable search over these values. The value of the mechanism is that the space requirement is tiny, independent of the size of the original data, and that the cost of the filtering operation is small. In some contexts, especially with large and/or high dimensional data, it may be that the best possible query strategy is to first perform a filter over the whole data set, and then re-check any unfiltered data against the metric space. However the mechanism is more flexible than this, and may be used in any of the following ways:

exact pre-filter For a given query and threshold distance, the data representations of the full set can be scanned and any data discovered to be beyond the threshold can be filtered out. The residual cost of an exact search is then largely dictated by the cost of performing distance calculations to those unfiltered objects in the original metric space. This strategy is tested in Section 5.1.

approximate pre-filter A class of approximate searches may be defined by tightening the restriction given by the calculated lower bound. If the filtering operation is defined as $\ell_2((x_{s_i}, y_{s_i}), (x_{q,i}, y_{q,i})) \leq \gamma\, t$, for some $\gamma < 1$, then clearly less unfiltered values will be returned which will improve the cost of the search. However the geometric guarantee is no longer in place, and so some correct results may be lost. For this mechanism to be effective, we have to understand the probabilistic ratio between the calculated lower bound and the actual small distances within the space. This strategy is tested in Section 5.2.

ranked order nearest-neighbour queries Scanning the entire data, ordering by individual lower-bound values, will produce an ordering in the database w.r.t. the distance to the query. This ordering does not necessarily coincide with the ordering in the original metric space, especially as the values are drawn from many different projected spaces. However if the orders are close enough, then taking the first k', computing the exact distances and keeping just the $k < k'$ closest, will give us a good approximation of the k-NN. For this mechanism to be effective, we require to understand the type of order induced by the lower bounds. This strategy is tested in Section 5.3.

exact post-filter Finally, the lower-bound filter can be incorporated within another, scalable, mechanism as a post-filtering mechanism. Scalable indexing mechanisms work by excluding whole subsets of the data from a search, until at some point original distances require to be checked. In any such mechanism, a significant number of distance calculations will have been made before this phase of the search, for example to distinguished reference objects at each node during the navigation of a tree structure. These objects are known at pre-processing time, and therefore may be re-used as a distinguished set of reference points for the filtering mechanism. The small σ representations can be stored within an indexing data structure with little extra cost. Then, whenever the calculation $d(q, s_i)$ is required at query time, the filtering operation may be cheaply performed first, in case the actual distance calculation may be avoided. This strategy is tested in Section 5.5.

4.1. Selection of best reference pair

It is possible to use a statistical technique to select a good reference point pair per individual datum. A sample set of data is used, the *witness* set.

For a query over a finite metric space (S, d), first a set of m objects is taken from S and used to form a set $\mathcal{P}$ comprising numbered reference points p_i. For a given set of m reference objects, each of the $\binom{m}{2}$ pairs p_i, p_j is considered. For each, a 2D Euclidean space is built, exactly corresponding to those depicted in the earlier figures. Each space is built using the data from the witness set, according to the distances of each element to the pair of reference objects. These spaces may be efficiently searched using normal metric indexing techniques, and as the space is a genuine 2D space very efficient mechanisms such as the KD-Tree [31] can be used.

Each element of the data set is now considered as a query against each of these $\binom{m}{2}$ metric indexes, and the one with the least *local density* is selected to represent that element. There are various mechanisms for assessing local density, for example the smallest number of results for a threshold query, or the largest distance in the result set of a kNN query. We tested various ways over some different data sets and found relatively little difference in the cost or outcome, and settled on the strategy of picking the pivot pair which gave the largest distance to the third-nearest 2D point.

While this mechanism is effective, it is of course extremely expensive, with a quadratic cost according to the number of reference points. In general, for high-dimensional queries, a relatively large number of reference points will be required. We discuss linear geometric approximations in Section 5.4.

4.2. Build cost

The dominant cost is in searching the 2D pair space at build time; the tables show results up to 150 reference points which of course also requires 150 distance calculations per datum. However these distance calculations are likely to be amortised within another search mechanism as shown in Section 5.5.

[4] Times are measured on 3.1 GHz Intel i7 quadcore processor, but will vary with context.

Table 2
Data set summary.

Name	Dimensions	Metric
nasa	20	Euclidean
colors	112	Euclidean
euc10	10	Euclidean
euc20	20	Euclidean
sift	128	Euclidean
gist	480	Jensen–Shannon

The cost of searching the pair space however increases quadratically with the number of reference points, making it infeasible for larger numbers. This cost is almost independent of the cost of distance calculations or size of data in the metric space: the cost of searching $\binom{m}{2}$ 2D spaces becoming quickly predominant as m increases. The cost is perfectly quadratic, in our experiments we have measured the cost $C(m) = 0.007\ \mathrm{m}^2$ milliseconds for m pivots; even with only 150 reference points this is approaching 0.2 s per datum.

This leaves an interesting problem. The number of reference points does not typically constitute a performance problem in terms of distance calculations; the large cost is in the exhaustive search for the best pair of reference points. The reason the cost is high is because there are a huge number of potential pairs, which is the reason the mechanisms works so well. We have shown tremendous potential when the best pair of points is calculated from the very large number of pairs available. If we can find a way of finding these cheaply, ideally in a manner that scales linearly rather than quadratically with the number of reference points, the mechanism should become even more useful.

In the context of searching a very large, high-dimensional, data set, then thousands of extra distance calculations are unlikely to be significant, but this would result in a huge potential space of reference point pairs that is intractable to search; thus we seek linear-scaling solutions using geometric analysis instead. For once, it is not reasonable to assume an arbitrary amount of pre-processing time is acceptable in order to achieve a small improvement in query time.

5. Experimental evaluation

To evaluate the potential of the proposed mechanism, experimental evaluation is performed on the following metric spaces:

nasa, colors The SISAP *nasa* and *colors* [5] are two benchmarks for metric indexing and searching approaches. The *nasa* set contains 40,150 real vectors of dimension 20, each obtained from images downloaded from the NASA photo and video archive site. The *colors* set contains 112,682 feature vectors of dimension 112. Each vector is a colour histogram of a medical image. These data are compared with the Euclidean metric.

euc10, euc20 These are uniform spaces, generated with a Gaussian distribution across 10 and 20 dimensions respectively. They are compared with the Euclidean metric. Their size is arbitrary, being generated according to experimental need.

sift We use the *ANN_SIFT1M*[5] data set that contains one million *SIFT* [32] image descriptors. Each descriptor comprises 128 floating point values. These descriptors are ℓ_2-normalised vectors and are normally compared using the Euclidean distance. For this space we have a pre-generated ground truth for 1000 queries, for each of which the 100 nearest neighbours have been calculated.

Table 3
The proportion of non-filtered data for each of the experimental spaces, using different-sized pools of reference objects. The pair used for each datum is selected from all the available pairs, that is $\binom{m}{2}$ for m reference objects.

m	$\binom{m}{2}$	nasa	colors	euc10	euc20	sift	gist
2	1	0.133	0.269	0.477	0.957	0.617	0.963
5	10	0.100	0.164	0.328	0.894	0.512	0.942
10	45	0.075	0.094	0.213	0.811	0.511	0.897
25	300	0.039	0.059	0.095	0.659	0.462	0.877
50	1225	0.027	0.043	0.05	0.531	0.416	0.834
75	2775	0.020	0.036	0.037	0.461	0.397	0.829
100	4950	0.017	0.032	0.03	0.419	0.376	0.818
VPT for comparison		0.098	0.127	0.314	0.982	0.498	0.964

gist This data is generated from a set of one million images (the MirFlickr collection, [33]). 480-dimensional GIST descriptors have been previously extracted and are publicly available.[6] Previous work [34] has shown that the best metric for searching this data is the Jensen–Shannon distance, whose cost over this data is around 25 times that of Euclidean distance. For this space we also have the 100NN ground truth for 1000 selected queries.

Table 2 gives a summary of the data sets used in the experiments.

As the data sets are diverse, a uniform setup was applied to each to ensure that results are comparable. For each set, randomly selected non-intersecting subsets were extracted from which data, queries, and reference points are extracted for each experiment. As most of the experiments described do not require to show scalability, a data size of only 10,000 values was selected, against which 1000 queries are evaluated in each experiment. For the experiments in Section 5.3 the SIFT and GIST spaces are also used with one million objects.

A nearest-neighbour *ground truth* is also calculated. For each query, the closest values from the data are calculated and stored. When queries are performed during experiments, the nearest-neighbour distance is used to conduct a fixed threshold search over the data, therefore returning a single result per query. This technique is preferable to using a search threshold which is fixed for all queries, as there is considerable variation among the local densities of the queries, and this approach would therefore give widely varying numbers of results per query.

All experiments are executed using 64-bit Java v8, executed single-threaded on an Apple MacBook Pro with a 3.1 GHz Intel Core i7 processor and 16 GB of memory, disconnected from the network when appropriate. The source code is available at https://bitbucket.org/richardconnor/low_dim_embedings_is or from the authors.

5.1. Exact pre-filtered queries

Our first experiments show the effect of using different numbers of reference points, from which the "best" pair (as described in Section 4.1) is chosen from each of the $\binom{m}{2}$ combinations that are available for a selection of m reference points. Table 3 shows, for each data set, the proportion of the data which requires to be accessed (this is, which is not pre-filtered) after executing the filtering mechanism. Each row shows this value when the reference points are selected from different sized sets. Notice that the value m is the number of available reference objects; the number of available pairs is of course generally much greater.

5 http://corpus-texmex.irisa.fr.

6 http://press.liacs.nl/mirflickr.

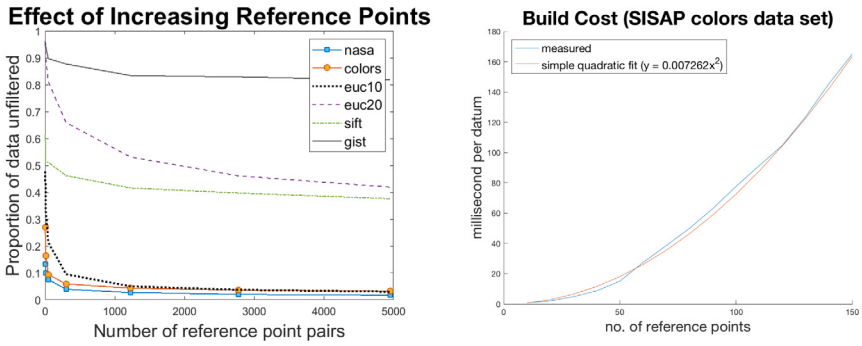

Fig. 7. The left-hand graph shows the proportion of non-filtered data varying the number of available reference pairs. The right-hand graph shows the build cost varying the number of reference points.

The same queries were executed using a balanced Vantage Point Tree (VPT, [35]), to give a comparison with a common indexing approach. While in all cases the VPT accesses more data with even a small number of reference points, this is not a fair comparison in terms of overall efficiency, as in cases where the VPT accesses a relatively small proportion of the data, the scalability of the indexing will dominate for larger data collections. However for the spaces on the right hand side of the table, the proportion of access is too high for such mechanisms to work, and a sequential scan becomes faster than indexing. In these cases, our mechanism should give the advantage.

Fig. 7 shows the same data in a line graph. The plot is made against the number of pairs available, rather than the number of reference points. It can be seen that, as the number of potential reference pairs becomes greater, the mechanisms becomes more effective. However this seems to asymptotically approach a limit depending on characteristics of the space itself.

Fig. 7 also shows the build cost, which as previously mentioned is quadratic with respect to the size of the data and does require to be taken into account; even with these small data sets (10k elements) the cost of evaluating the best pairs from 100 reference objects is around 15 min. These measurements are taken from the *colors* data set, however the cost is almost completely independent of the data size and metric cost. In Section 5.4 we examine ways of reducing this when a smaller cost is pragmatically required.

Considering the two sides of Fig. 7, it is clear that the choice of m depends on the context of use. In general, a high value for m leads to a much higher pre-processing cost, while at some point does not detract much from the query cost. The choice therefore will depend very much on the relative importance of these two costs, as well as the characteristics of the space itself.

It is worth noting that cost of the pre-filter mechanism itself is effectively constant: the difference between 2 and 100 initial distance measurements is almost immaterial in any of these searches. As noted above, the core query-time mechanism takes only around 0.3 μs per datum, and so the majority of the cost of a sequential search is directly proportional to the number of distance calculations required within the original space.

5.2. Approximation by reduced lower bound

The lower bound described up to this point gives a mathematically safe way of excluding data elements from a search. As has been seen, in some cases it is extremely effective and allows a very large proportion of the data to be filtered, but in other cases it is less effective; as always, in the spaces with higher dimensionality.

While this is the tightest lower bound justified by the analysis presented so far, there is experimental evidence that in pragmatic terms the bound is sometimes larger than necessary. Fig. 8 shows one such piece of evidence.

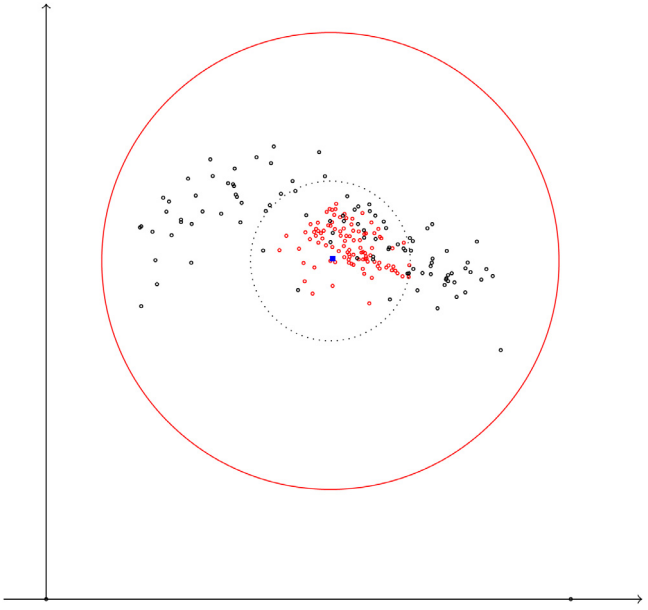

Fig. 8. 2D projection of objects taken from the SIFT metric space with respect to two randomly selected pivots. The pivots are plotted in the X axis. A query point (blue square in the graph) is plotted in the plane. The 100 nearest neighbours to the query from within a set of one million object are coloured in red. A further 100 randomly selected objects from the set are coloured in black. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The figure shows the 2D projection of objects taken from the SIFT metric space. An arbitrarily selected reference pair are plotted along the X axis, and a query with a known ground truth is plotted in the plane at the centre of the circles (blue square in the graph). The ground truth gives the 100 nearest neighbours from within a set of one million objects. The 2D projections of the 100 nearest neighbours are coloured in red in the graph. Another set of 100 randomly selected objects from the SIFT metric space are projected in the same plane and are coloured in black. Around the query two circles are drawn. The larger circle has a radius t_1 corresponding to the actual distance from the query to its one-hundredth nearest neighbour (i.e. the distance calculated in the original SIFT metric space). The smaller circle has a radius t_2 corresponding to the maximum Euclidean distance between the planar projection of the query and the planar projections of its 100 nearest neighbours (i.e. the largest lower-bound calculation from all 100 of the near-neighbour objects).

The different spread of the two sets of objects is very clear; the near-neighbours are tightly clustered towards the centre, while the others are spread much more widely. Using the exact pre-filtering mechanism, none of these objects could be safely filtered as they all lie within the outer circle, which represents the geometric guarantee given by the four-point lower bound. In other words, the Euclidean distance between the 2D projected points is a lower-bound of the actual distance and thus, in theory, it can be exploited to filter the search results (points in the 2D plane with distance to the query greater than the t_1 can be filtered out). However, in the example shown in Fig. 8, none of the considered points can be excluded from the search using this lower-bound because their planar projections are within distance t_1 to the planar projection of the query. However it is clear that, for this pair of reference objects and query at least, a much smaller pragmatic bound could be used which would allow useful filtering but without erroneously filtering any correct results (e.g, using a threshold distance equal to t_2 instead of t_1 when searching in the 2D plane).

Having made this observation, the next experiment shows the effect of reducing the effective lower-bound radius. Specifically, an object s_i is filtered out if $\ell_2((x_{s_i}, y_{s_i}), (x_{q,i}, y_{q,i})) > \gamma\, t_1$, for some factor $\gamma < 1$. For each data set, decreasing values of the lower bound threshold are selected as the filter radius. Specifically, we tested decreasing the factor γ, between 1.0 and 0.1 in intervals of 0.1. Two outcomes are measured: the proportion of the whose data set which is not filtered (i.e. accessed data), and the proportion of correct object, out of the 100 known nearest neighbours, which are correctly returned (i.e. the recall). Results are reported in Table 4.

The results show that the observed effect exists in the general case, and furthermore is more pronounced in the higher dimensional spaces. In particular GIST, which is the least tractable space, allows a value almost as low as half the deduced lower bound to be used whilst still achieving full recall. The interesting observation here is that, for the first time, we see some significant reduction in cost, saving 75% of the distance calculations which would be required for an exhaustive scan of the data. The results for SIFT also show the same reduction is possible, in this case allowing all the correct results to be returned for a cost saving of over 90%. Of course the drawback is that this mechanism does not guarantee to give all correct results for any value of $\gamma < 1.0$ and thus the results are essentially approximate, albeit with a seemingly very high probability of success.

5.3. Ranked order nearest-neighbour queries

The observations of the previous section also give rise to a further hypothesis, which is that a correlation may exist between the lower-bound filter distance and the true distance from the original metric space. Once again there is no geometric guarantee of this, but it may happen if objects that are closer in the metric space fall closer to the centre of the circle in the 2D projection. In fact there is less geometric justification that we know of for this situation: when reducing the effective lower bound as in Section 5.2 all of the objects with which we are dealing have been projected from one metric space to another. We now propose to compare the distances between objects that have been projected into different spaces.

The strategy is to perform an approximate k-nearest neighbour search as follows. For a query q, a scan is made over the full data and the four-point lower-bound is calculated for each $d(q, s_i)$. For some $k' > k$, the k' objects with the smallest lower-bound values are returned. These k' objects are then checked against the query in the original metric space, and the k closest objects from these are returned. As a scan of the very small data representations may be made at relatively low cost, this will give a viable mechanism for an approximate k-nearest neighbour query if there is a good correlation between the lower-bound distances, independent of the projected space, and the true distances.

To test this hypothesis, the SIFT and GIST spaces were used. To perform a useful experiment a larger data set is required, in order to accommodate a significant number of useful nearest-neighbour objects for each query. For each of these data collections one million objects is available, along with a ground truth of 1000 queries. For each query, the ground truth makes available the 100 nearest neighbours within the collection.

Fig. 9 shows the outcome for these 1000 queries over the SIFT and GIST data sets. For each query, for values of k' between 10,000 and 100,000, the k' smallest lower-bound values are returned from an initial scan of the data representations. These object identifiers returned represent respectively between 1% and 10% of the data which requires to be accessed to determine the 100 smallest true distances. For each object identifier returned, a check is made to determine if it was in the 100 nearest neighbours for that query as given by the ground truth. The number that are in the 100NN is then treated as a percentage of the correct recall. Individual results were kept for each of the 1000 queries to allow the distribution of outcomes to be determined. These are displayed in Fig. 9 as a series of box plots, showing the recall when between 1% to 10% of the lower-bound calculations are retained.

As can seem from the Figure, for SIFT it is the case that with only 10% of the distance calculations being made, the mechanism gives a mean recall of 0.99, with the lower quartile at 0.97. At the other end of the scale, if the situation demands that only 2% of the distances can be afforded, then this still yields a median of 0.72 and a lower quartile at 0.63 recall.

Results for GIST are not quite so good, but this is not surprising given the previous results over this high-dimensional set. Once again it is possible to obtain 90% of the correct results while performing only 10% of the distance calculations required for an exhaustive search.

One final point in this section is that, while the results in terms of percentage access vs. recall are similar in outline to those shown in the previous experiment (Section 5.2), the mechanism described here has the significant advantage that the search performed in a pure nearest-neighbour search, while for the previous results a query threshold is required to be known before the start of the query.

5.4. A geometric approach to reference pair selection

A number of intuitively-derived methods for the selection of first and second reference points were tested. In all cases, sets of 10, 50, 150 and 500 objects were chosen to act as reference points, and these were scanned linearly in two passes according to the following strategies. The intent is to find a strategy that gradually improves with respect to the number of reference points, but where the construction cost remains linear.

The strategies used for each of two linear-cost scans were as follows:

1. random, to act as a benchmark
2. for each data point, associate the closest reference point
3. for each data point, associate the farthest reference point
4. for each of the m reference points, associate it with the $\frac{1}{m}$ closest subset of the data (and do not consider these data points again)
5. for each of the m reference points, associate it with the $\frac{1}{m}$ farthest subset of the data (and do not consider these data points again)

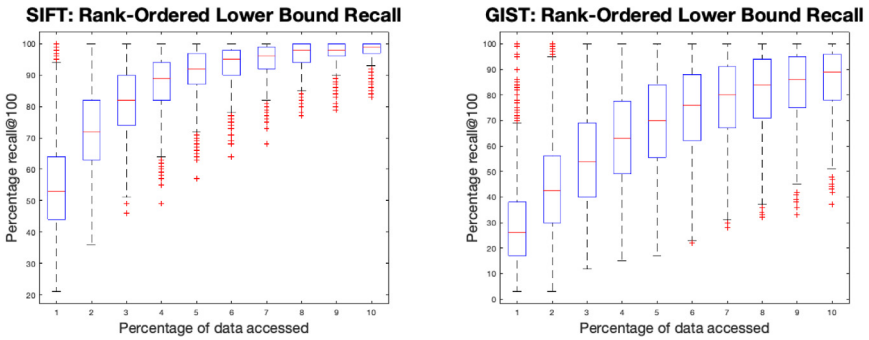

Fig. 9. Ranked order results for SIFT and GIST collections of 1M data objects. For each of 1000 queries, the four-point lower bound is used as a filter to find a fixed-size set of promising candidates, which are then checked within the original space. The X-axis gives the percentage of the 1M data which is accessed, the Y-axis shows a boxplot of the percentage recall, for a 100 nearest-neighbour ground truth, across all queries.

Table 4
Results for approximate pre-filtering. For each data set and factor γ we measured the proportion of not-filtered data (*access*) and the proportion of correct objects which are correctly returned (*recall*).

γ	colors		nasa		euc10		euc20		sift		gist	
	Access	Recall	Access	Recall	Access	Recall	Access	Recall	Access	Recall	Access	Recall
1	0.032	1	0.017	1	0.03	1	0.419	1	0.376	1	0.818	1
0.9	0.023	1	0.012	0.99	0.02	1	0.307	1	0.319	1	0.741	1
0.8	0.016	0.98	0.008	0.95	0.014	0.96	0.206	1	0.26	1	0.638	1
0.7	0.011	0.91	0.005	0.89	0.009	0.87	0.127	0.99	0.2	1	0.51	1
0.6	0.007	0.79	0.003	0.74	0.005	0.74	0.07	0.93	0.141	1	0.366	1
0.5	0.004	0.63	0.002	0.61	0.003	0.57	0.035	0.8	0.087	0.99	0.225	0.99
0.4	0.002	0.45	0.001	0.43	0.002	0.39	0.015	0.58	0.045	0.95	0.113	0.89
0.3	0.001	0.26	0.001	0.28	0.001	0.25	0.006	0.33	0.018	0.77	0.043	0.63
0.2	0	0.11	0	0.13	0	0.11	0.002	0.14	0.005	0.47	0.011	0.29
0.1	0	0.03	0	0.03	0	0.03	0	0.03	0.001	0.13	0.002	0.06

6. having selected a first reference point, choose the second to minimise the altitude (Y-coordinate) of the plotted 2D apex point
7. having selected a first reference point, choose the second to minimise the horizontal displacement (X-coordinate) of the plotted 2D apex point

The first five strategies were tried for each of first and second reference point choice, whereas the last two were used only for the choice of the second point; thus a total of 35 different strategies were tested.

Methods (2) and (3) in any combination proved no better than random, and actually became slightly worse with a larger number of reference points; we believe this is because of non-uniformity within the sets and the presence of outliers in the reference points. This problem was fixed by use of methods (4) and (5), where the closest or farthest $\frac{1}{m}$ of the data is associated with each reference point.

Table 5 shows a few of the results. The first row shows a purely random choice for comparison. The second shows method (4) used for the first pivot, and method (6) for the second. Finally the third row shows the use of method (4) for the first pivot and method (5) for the second, which gives the best compromise for these data sets and thresholds. The final effect of achieving 97% exclusion – as much as is achieved by a very sophisticated indexing structure over the full data set – through a linear cost construction of a 10-byte data representation is really a significant achievement. Note that in the cost comparisons, the "random" benchmark cost is effectively zero; at 500 pivots the cost of either mechanisms is restricted to around 0.1 ms per datum independent of the size of the data set, when the thorough search described in Section 4.1 would have cost 1.75 s.

Table 5
Results shown only for the lowest standard threshold of the *colors* data set (i.e. $t = 0.052$ which return 0.01% of the data), other results are consistent. We give the build cost (msec per object) and exclusion rate for some of the strategies tested.

Pivot strategy			Number of Pivots			
First	Second		10	50	150	500
Random	Random	Build cost	0.0008	0.0008	0.0010	0.0016
		Exclusion	0.929	0.925	0.926	0.924
Low dist	Low alt	Build cost	0.014	0.022	0.045	0.124
		Exclusion	0.930	0.958	0.967	0.966
Low dist	High dist	Build cost	0.023	0.030	0.045	0.089
		Exclusion	0.946	0.962	0.971	0.973

5.5. Incorporation within list of clusters

Finally, we report results where our mechanism is incorporated with another, scalable, indexing mechanism. We have chosen a well-known indexing structure, and give a very simple technique which extends this using the four-point exclusion mechanism as a post-filter. That is, the mechanism is embedded within the original structure to act as an internal filter, avoiding the calculation of original-space distances where the lower-bound calculation makes this unnecessary.

For this purpose we choose the List of Clusters [36], generally regarded as the most scalable mechanism known. We have measured this, with and without our optimisation, over the SISAP benchmark data sets *colors* and *nasa*, to perform threshold search

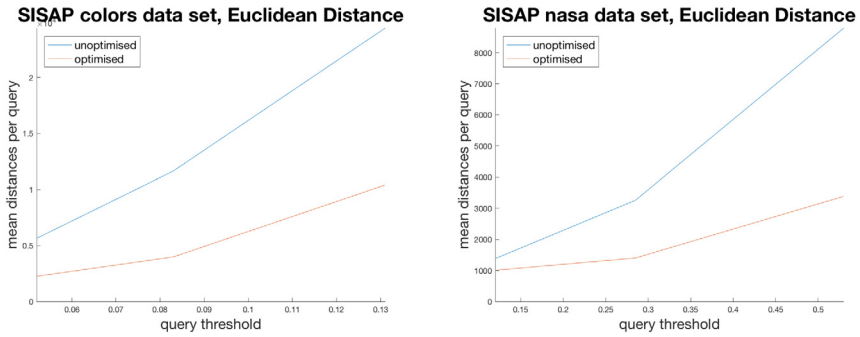

Fig. 10. SISAP benchmark space results with and without optimisation.

using the three standard benchmark thresholds[7]; we show a very significant improvement in performance.

As the list of clusters is built, at each node a pivot point is selected and a fixed number of objects, those being closest to this pivot point, are stored in an associated "bucket". Especially towards the start of this process, the cover radius of these objects from the pivot point is likely to be very small, therefore maximising the probability of the bucket being excluded from a search. When each cluster is constructed, the distances between every object in that cluster, and every pivot point from the root to that point in the list, will have been to be calculated as a part of the construction algorithm.

To this structure, we add only our small representations of the objects within each bucket, and cause no extra distance calculations at either build or query time. The local pivot point is used as the first reference point, and the furthest pivot from the so-far constructed spine of the tree as the second. This gives an approximation to the geometric technique (low dist, high dist) described in Section 5.4, and the only extra construction-time cost is the calculation of the 2D coordinate from these distances; in experiments, this was literally undetectable. The extra space cost is 10 bytes per object, for the *colors* data set representing an increase of around 1%.

At query time, the mechanism is used in the normal way based on the measured distance between the query and each pivot point down the spine of the list. In cases where the local "cluster" requires to be searched, then the four-point representations are first checked. The four-point representation of the query requires only the calculation of the 2D representative point, as all of the distances required have already been measured as the query algorithm progresses down the spine of the list. The lower-bound computation then comprises a 2-dimensional ℓ_2 distance. If the lower-bound distance is greater than the query threshold, there is no requirement to access the corresponding object and check its true distance against the query object. This saves not only an expensive distance calculation, but also the movement of the object within memory.

Table 6 shows the number of distance calculations made against the original data sets, along with the percentage improvement shown; the same values are plotted in Fig. 10. It can be seen in almost all cases that the query cost is better than halved, in return for only a small increase in memory size.

Table 6
Improvement shown on List of Clusters using four-point post-filtering. Values given are mean number of distance calculations per query.

Threshold	Standard			Optimised		
	$t_{0.01\%}$	$t_{0.1\%}$	$t_{1\%}$	$t_{0.01\%}$	$t_{0.1\%}$	$t_{1\%}$
SISAP colors	5645	11 649	24 401	2256	3987	10 402
SISAP nasa	1381	3258	8790	1007	1402	3384

6. Conclusions

We presented a method to obtain good distance bounds between a query and all the database elements using a minimally-sized representation comprising only two reference object identifiers, and two floating point values, per database object. The two floating point values are the coordinates in a two-dimensional Euclidean space where a lower-bound for the actual distance to a query can be efficiently computed. The combination of the very large space of object pairs available from a relatively small set of reference objects, and the observation that each pair gives a significantly different projection of the space, combines to allow a very high rate of successful exclusion for a typical range search, with exclusion rates of 99.6% and 99.9% obtained respectively for the SISAP benchmark *colors* and *nasa* data sets, with only 150 reference objects being used. For a data size of around 10 bytes per object and a cheap arithmetic check these results are impressive.

We also applied our mechanism in much higher dimensional spaces in conjunction with either range or nearest-neighbour search. Our results are promising, for example on SIFT data our mechanism gives a mean recall of 0.99 with only 10% of the distance calculations being made.

The mechanism we describe is well-suited to a CUDA implementation which should give an effective small constant runtime on a GPU. The memory footprint required for the data set is ten bytes per object: two short integers to identify the reference objects,[8] and two single-precision floats for the *XY* coordinates. This gives a runtime memory requirement of only 10 MB per million objects in the data set.

This memory represents the data to be queried and is thus invariant per query, therefore the cost of loading this memory from the CPU host to the GPU device is not a part of the per-query cost. Modern CUDA standards have the capability to influence persistence of data in the L2 cache, where this data could be placed to give higher bandwidth and lower latency. To perform a query, data passed from host to device is just the distances between the query and each of the reference objects; in our example we used a maximum of 100 of these, implying a per-query transfer of less than 0.5 kB before the parallel computation starts. The result, in the simplest possible implementation, is an array of bits, one per data object, where a bit would be set if the corresponding object is a potential solution; this represents 125 kB per million objects. Typical modern GPU processors give latency of around some ten of microseconds, with bandwidth

[7] Threshold values which return 0.01%, 0.1% and 1% of the data sets respectively. For the SISAP colors, the thresholds are $t_{0.01\%} = 0.052$, $t_{0.1\%} = 0.083$, $t_{1\%} = 0.131$. For the SISAP nasa, the thresholds are $t_{0.01\%} = 0.12$, $t_{0.1\%} = 0.285$, $t_{1\%} = 0.53$.

[8] assuming no more than 256 reference objects are used.

around 1GB per second. The actual cost of the SIMD computation is a relatively small number of 2D Euclidean distance calculations per processor, where this number is the size of the data divided by the number of available processors.

In summary, since it is theoretically impossible to avoid a sequential scan for nearest neighbour search, even in the approximate sense, a cheap exclusion mechanism that is easily parallelisable is competitive. We remark that this mechanism can be used in conjunction with probabilistic methods requiring post-filtering or re-ranking, like metric inverted files. We have given one successful example of this: for an almost immeasurably small increase in build cost and memory, the performance of the List of Clusters indexing structure has been shown to be radically improved. It is likely that many similar examples exist.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was partially supported by VISECH project (ARCO-CNR, CUP B56J17001330004) co-funded by the Tuscany region.

References

[1] L.M. Blumenthal, Theory and Applications of Distance Geometry, Clarendon Press, 1953, p. 347.
[2] R. Connor, L. Vadicamo, F.A. Cardillo, F. Rabitti, Supermetric search, Inf. Syst. 80 (2019) 108–123, http://dx.doi.org/10.1016/j.is.2018.01.002.
[3] A. Rubinstein, Hardness of approximate nearest neighbor search, in: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, ACM, 2018, pp. 1260–1268, http://dx.doi.org/10.1145/3188745.3188916.
[4] G. Amato, C. Gennaro, P. Savino, Mi-file: Using inverted files for scalable approximate similarity search, Multimedia Tools Appl. 71 (3) (2014) 1333–1362, http://dx.doi.org/10.1007/s11042-012-1271-1.
[5] K. Figueroa, G. Navarro, E. Chávez, Metric spaces library, 2007, Online http://www.sisap.org/library/manual.pdf.
[6] E. Chávez, R. Connor, L. Vadicamo, Query filtering with low-dimensional local embeddings, in: Similarity Search and Applications, Springer International Publishing, Cham, 2019, pp. 233–246.
[7] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search: The Metric Space Approach, Vol. 32, Springer Science & Business Media, 2006.
[8] E. Vidal, New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa), Pattern Recognit. Lett. 15 (1) (1994) 1–7, http://dx.doi.org/10.1016/0167-8655(94)90094-9.
[9] M.L. Micó, J. Oncina, E. Vidal, A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements, Pattern Recognit. Lett. 15 (1) (1994) 9–17, http://dx.doi.org/10.1016/0167-8655(94)90095-7.
[10] K. Figueroa, E. Chávez, G. Navarro, R. Paredes, Speeding up spatial approximation search in metric spaces, J. Exp. Algorithmics 14 (2010) 6:3.6–6:3.21, http://dx.doi.org/10.1145/1498698.1564506.
[11] W.A. Burkhard, R.M. Keller, Some approaches to best-match file searching, Commun. ACM 16 (4) (1973) 230–236, http://dx.doi.org/10.1145/362003.362025.
[12] R. Baeza-Yates, W. Cunto, U. Manber, S. Wu, Proximity matching using fixed-queries trees, in: M. Crochemore, D. Gusfield (Eds.), Annual Symposium on Combinatorial Pattern Matching, Springer Berlin Heidelberg, 1994, pp. 198–212, http://dx.doi.org/10.1007/3-540-58094-8_18.
[13] C. Celik, Priority vantage points structures for similarity queries in metric spaces, in: H. Shafazand, A.M. Tjoa (Eds.), EurAsia-ICT 2002: Information and Communication Technology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 256–263, http://dx.doi.org/10.1007/3-540-36087-5_30.
[14] G. Ruiz, F. Santoyo, E. Chávez, K. Figueroa, E.S. Tellez, Extreme pivots for faster metric indexes, in: Proceedings of the 6th International Conference on Similarity Search and Applications (SISAP 2013)- Volume 8199, (SISAP 2017), 2013, pp. 115–126, http://dx.doi.org/10.1007/978-3-642-41062-8_12.
[15] T. Skopal, J. Pokorný, V. Snášel, Nearest neighbours search using the pm-tree, in: L. Zhou, B.C. Ooi, X. Meng (Eds.), Database Systems for Advanced Applications, Springer Berlin Heidelberg, 2005, pp. 803–815, http://dx.doi.org/10.1007/11408079_73.
[16] E. Chavez, J.L. Marroquin, R. Baeza-Yates, Spaghettis: an array based algorithm for similarity queries in metric spaces, in: 6th International Symposium on String Processing and Information Retrieval. 5th International Workshop on Groupware (Cat. No.PR00268), 1999, pp. 38–46, http://dx.doi.org/10.1109/SPIRE.1999.796576.
[17] E. Chavez, U. Ruiz, E. Tellez, Cda: Succinct spaghetti, in: Similarity Search and Applications, Springer International Publishing, 2015, pp. 54–64, http://dx.doi.org/10.1007/978-3-319-25087-8_5.
[18] K. Menger, Untersuchungen über allgemeine metrik, Math. Ann. 100 (1) (1928) 75–163.
[19] W.A. Wilson, A relation between metric and euclidean spaces, Amer. J. Math. 54 (3) (1932) 505–517.
[20] L.M. Blumenthal, A note on the four-point property, Bull. Amer. Math. Soc. 39 (6) (1933) 423–426.
[21] K. Menger, New foundation of euclidean geometry, Amer. J. Math. 53 (4) (1931) 721–745, URL http://www.jstor.org/stable/2371222.
[22] I.J. Schoenberg, Metric spaces and completely monotone functions, Ann. of Math. 39 (4) (1938) 811–841, URL http://www.jstor.org/stable/1968466.
[23] R. Connor, F.A. Cardillo, L. Vadicamo, F. Rabitti, Hilbert exclusion: improved metric search through finite isometric embeddings, ACM Trans. Inf. Syst. (TOIS) 35 (3) (2016) 1–27.
[24] R. Connor, L. Vadicamo, F.A. Cardillo, F. Rabitti, Supermetric search with the four-point property, in: Proceedings of 9th International Conference on Similarity Search and Applications (SISAP 2016), in: Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 51–64, http://dx.doi.org/10.1007/978-3-319-46759-7_4.
[25] R. Connor, A. Dearle, L. Vadicamo, Modelling string structure in vector spaces, in: Proceedings of the 27th Italian Symposium on Advanced Database Systems, Sun SITE Central Europe, 2019.
[26] L. Vadicamo, C. Gennaro, F. Falchi, E. Chávez, R. Connor, G. Amato, Re-ranking via local embeddings: A use case with permutation-based indexing and the nsimplex projection, Inf. Syst. (2020) 101506, http://dx.doi.org/10.1016/j.is.2020.101506.
[27] M.E. Houle, Local intrinsic dimensionality iii: Density and similarity, in: International Conference on Similarity Search and Applications, Springer, 2020, pp. 248–260.
[28] O. Pedreira, S. Marchand-Maillet, E. Chávez, Reverse k-nearest neighbors centrality measures and local intrinsic dimension, in: International Conference on Similarity Search and Applications, Springer, 2020, pp. 270–278.
[29] O. Pedreira, N.R. Brisaboa, Spatial selection of sparse pivots for similarity search in metric spaces, in: International Conference on Current Trends in Theory and Practice of Computer Science, Springer, 2007, pp. 434–445.
[30] B. Bustos, G. Navarro, E. Chávez, Pivot selection techniques for proximity searching in metric spaces, Pattern Recognit. Lett. 24 (14) (2003) 2357–2366.
[31] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517, http://dx.doi.org/10.1145/361002.361007, URL http://doi.acm.org/10.1145/361002.361007.
[32] D.G. Lowe, Distinctive image features from scale-invariant keypoints, Int. J. Comput. Vis. 60 (2) (2004) 91–110.
[33] M.J. Huiskes, M.S. Lew, The mir flickr retrieval evaluation, in: MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval, ACM, New York, NY, USA, 2008, pp. 39–43.
[34] R. Connor, F.A. Cardillo, Quantifying the specificity of near-duplicate image classification functions, in: 11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2016.
[35] P.N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, in: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, in: SODA '93, Society for Industrial and Applied Mathematics, USA, 1993, pp. 311–321.
[36] E. Chávez, G. Navarro, A compact space decomposition for effective metric indexing, Pattern Recognit. Lett. 26 (9) (2005) 1363–1376, http://dx.doi.org/10.1016/j.patrec.2004.11.014.