Systems Research and Info. Systems
Vol. 10, pp. 89–112
Reprints available directly from the publisher
Photocopying permitted by license only

APPROXIMATING VECTORS FOR SIMILARITY SEARCHING

B. J. BRIEDIS[a,*] and T. D. GEDEON[b,†]

[a]*School of Computer Science and Engineering, The University of New South Wales, NSW, Australia;* [b]*School of Information Technology, Murdoch University, WA, Australia*

(Received June 2000)

Similarity searching in large collections that use long vectors to represent objects is considered. A number of vector indexing techniques explicitly or implicitly involve the use of vectors whose elements approximate those of the vectors to be indexed. This paper presents a measure that, given one assumption, is suitable for judging the quality of approximations of vectors. If it is further assumed that the distributions of elements of each dimension are independent of those of other dimensions and that the distance measure preserves this independence, then it is possible to apply a related measure to the elements of each dimension independently of those in other dimensions. These measures are used in the design of a heuristic algorithm that creates approximations of vectors. The VA-file is modified to use approximations of vectors created using this algorithm, and its performance is evaluated.

Keywords: Similarity searching; *k*-nearest neighbour; High-dimensional vector retrieval

1. INTRODUCTION

This paper discusses a method for improving the construction of approximations of vectors for use in conjunction with a number of indexing techniques used for similarity searching. The vectors under consideration are dense and long, typically with between about ten and a few hundred dimensions.

*Corresponding author. e-mail: benbriedis@hotmail.com
†e-mail: tom@it.murdoch.edu.au

It is assumed that there is a large collection of objects and that searches are frequently conducted on this collection in order to find some number of objects that are similar to the given query objects. It is further assumed that the objects and the queries are represented using vectors, and that these vectors are compared using a similarity measure. This form of similarity searching is often called *k-nearest neighbour searching*. In this case, however, objects are thought of as being represented as points, and a distance measure is used to compare points. The terminology adopted here will be one of vectors and similarity, rather than one of points and distances.

Similarity searching has a large and diverse set of applications. Of these only a subset use long dense vectors to represent objects. In text retrieval and text classification long sparse vectors used have traditionally been used [35]. More recently, however, dimension reduction techniques have been employed in some systems to improve the quality of retrieval [15] and classification [44]. The vectors in these collections have between about 50 and 200 elements each, and are dense. Another range of applications result from the ability to represent shapes using vectors [36, 28]. The retrieval of shapes is useful in areas such as computer-aided design (CAD), geographical information systems (GIS), and robotic vision and movement [26, 6]. More complex one- and three-dimensional information may also by represented by long vectors through the use of histograms, Fourier transforms and various other methods. This leads to several applications in multimedia [19, 21, 31, 39], including the indexing of photographs, video, images of faces [40], medical images [30], and astronomical spectograms [13]. Similarity searching based on long dense vectors may also be performed on molecular sequences (*e.g.*, DNA) [2, 16, 43]. Other applications are knowledge discovery, data mining [18] and time series prediction and analysis (*e.g.*, for financial market movements) [17, 1]. Furthermore, the k-nearest neighbour algorithm has long been used for classification [14] in areas such as optical character recognition (OCR) and speech recognition [5]. Finding vectors quickly is also useful for multivariate density estimation [38] (both when using the *k*-nearest neighbour density estimation technique and when using kernel density estimation), and in vector quantisation for data, which is mostly used for the compression of images and sound [23].

After presenting a number of existing indexing systems in Section 2, Section 3 suggests that for some indexing systems it is possible to

divide the act of approximating vectors from the rest of the indexing system and to analyse the approximation process separately. Section 4 presents a measure suitable for judging the quality of the approximations of vectors, and Section 5 describes the relationship of this measure with two other common measures. In order to create vector approximations it is simplest to deal with just one dimension at a time. Theory is developed in Section 6 that allows this to be done and this theory is applied in Section 7 in the development of an optimisation routine. This work is then applied to the VA-file (Section 8) and performance results are presented in Section 9.

2. VECTOR INDEXING SYSTEMS

Many indexing techniques exist which are suitable for use with nearest neighbour searching. Some of these are the k-d tree [20] and modifications of it [25, 45, 4, 3], the R-tree [27], the TV-tree [32], the SS-tree [42], the SS+-tree [31], and the X-tree [10]. Most of these structures may also be used for other query types as well. A good summary of pre-1995 methods may be found in [22].

It has generally been found that the available techniques have been of little benefit when the dimensionality of the vectors to be indexed is high. Some authors have reported performance gains using tree structures for data of up to about 100 dimensions on some data sets [12, 31, 10], but unless the data is highly clustered or unless some of dimensions are strongly dominant then performance gains are unlikely as partitioning techniques suffer from inherent problems in high-dimensional space [41]. If the Chebyshev metric (l_∞) is used to measure dissimilarity then an efficient indexing structure, the Pyramid Technique [9], does exist. Unfortunately the properties of this measure are quite different from those of the other measures commonly used to test similarity.

One possible solution is to search in parallel [7]. A number of other possibilities, however, also exist. One is to compare the query to approximations of the vectors, rather than to the vectors themselves. The approximations can be stored in a compact file, known as a VA-file [41], that can be scanned faster than a file that contains the original vectors. Vector approximations of this sort may also be incorporated into other structures, such as trees [8]. Another possibility is to project

the high-dimensional space into a lower dimensional space a number of times, then to use indexing techniques that work in the lower dimensional space to perform quick searches on each of the projections, and then finally to exhaustively search the union of the results. This technique has been used in sub-vector indexing (SVI) [11] and in [37]. Neural networks such as Kanerva's sparse distributed memory (SDM) [29] and Greene's CHAM network [24] are also possibly options.

3. USING VECTOR APPROXIMATIONS

Assume that it is possible to divide the indexing process into two stages, the approximation of the vectors and the rest:

$$\text{Vectors} \longrightarrow \text{Vector Approximations} \longrightarrow \text{Index}$$

If it is possible to represent an indexing system in this way then it should be possible to consider the first transformation, from the original vectors to their approximations, largely or entirely in isolation from the rest of the indexing system.

Some indexing systems lend themselves readily to such treatment. In the case of the VA-file the final index is trivial—it is merely a file consisting of a sequence of vector approximations which, during a search, are read sequentially. It is also natural to treat the creation of the approximations separately for SVI and SDM. In both cases, before the index structure is created it is necessary to construct binary transformations of the original vectors (although it should be noted that it *may* be possible to generalise SDM to use the original vectors). Less obviously, it may also be beneficial to consider those indexing systems that employ spatial or data partitioning as implicitly using approximate vectors. This possibility is further considered in Section 6.

Generally it will be desirable to minimise the error that results from approximating the vectors. In order to determine how best to perform the approximations it is necessary to choose some measure of error that compares the true similarity of a pair of vectors to the similarity achieved when using approximate versions of the vector pairs.

It is possible to devise measures that directly compare the values of the elements in the original vectors with their approximations.

The approach taken here, however, is to compare the similarity scores that result when comparing approximations with the similarity scores that result when comparing the original vectors. Let $\vec{X}$ be a vector random variable that represents an item in the collection and let $\vec{Y}$ be a vector random variable that represents a query. Let $S = s(\vec{X}, \vec{Y})$ be the similarity between $\vec{X}$ and $\vec{Y}$. Let $\vec{X}'$ and $\vec{Y}'$ be approximations of $\vec{X}$ and $\vec{Y}$ and let $T = t(\vec{X}', \vec{Y}')$ be the *approximate similarity* of $\vec{X}$ and $\vec{Y}$. s and t are functions that return a similarity and they are probably the same function. Note that each value of S has an associated approximate similarity T. If a sample of readings is taken, a set is obtained of the form $\{\langle s_i, t_i\rangle | 1 \leq i \leq n\}$ where n is the sample size. The measure proposed here to determine the quality of the approximations operates on this set of pairs. Measures that compare the similarity/approximate similarity pairs have the virtue of not needing to explicitly refer to the similarity measure used or to the distribution of the data.

Various measures exist for comparing sets of pairs of elements, the two most obvious perhaps being the total sum of squares (TSS) and the correlation. The measure that appears to be the most appropriate to use, however, is $\mathrm{Var}(S-T)$. Justification for use of $\mathrm{Var}(S-T)$ as a measure is largely provided by the Theorem 1, presented below.

4. THE VAR($S-T$) MEASURE

First define the *completeness* (Θ) *of a search to be:*

$$\theta = \frac{|\mathcal{C} \cap \mathcal{R}|}{|\mathcal{C}|} \tag{1}$$

where $\mathcal{C}$ and $\mathcal{R}$ are the sets of *close* items and *retrieved* items respectively. Also let $\bar{\mathcal{C}}$ and $\bar{\mathcal{R}}$ be their complements. It is assumed that $|\mathcal{R}| = |\mathcal{C}|$.

THEOREM 1 *Let S, T and D be random variables with S representing the similarities between items in the collection and all possible queries, and T representing the approximations of these similarities. Let $T = S+D$, where D is independent of S. Minimising* $\mathrm{Var}(S-T)$ *maximises the expected completeness of retrieval.*

Proof Let $D=\mu+\sigma N$, where N is a standardised random variable. Note that $\mathrm{Var}(S-T)=\sigma^2$, and that minimising σ is equivalent to minimising $\mathrm{Var}(S-T)$.

Various values of σ will be considered. Let the unprimed forms of the variables (*e.g.*, $\mathcal{R}$) relate to σ, the primed forms (*e.g.*, $\mathcal{R}'$) relate to σ', and the double primed forms (*e.g.*, $\mathcal{R}''$) relate to σ''. Let variables for which $\sigma=0$ be denoted using a* (*e.g.*, $\mathcal{R}^*$).

For a particular query consider two values taken by T, t_a and t_b. All approximate similarities may be calculated as:

$$t_i = s_i + \sigma n_i + \mu \tag{2}$$

where i is an item to be retrieved and s_i and n_i are values taken by S and N respectively. From this it is evident that t_a and t_b are both linear with respect to σ. Consequently the lines given by (σ, t_a) and (σ, t_b) cross at most once.

In the case where $t_a > t_b$ and $t'_b > t'_a$ given $\sigma' > \sigma \geq 0$ the lines must cross in the interval $(\sigma..\sigma')$ as there is at most one crossover, $t''_b > t''_a$ for all $\sigma'' \geq \sigma'$.

Consider two points $a \in \mathcal{C} \cap \bar{\mathcal{R}}'$ and $b \in \bar{\mathcal{C}} \cap \mathcal{R}'$. $a \in \mathcal{R}^*$ as $a \in \mathcal{C}$ and $b \in \bar{\mathcal{R}}^*$ as $b \in \bar{\mathcal{C}}$. All close items have similarities greater than those of non-close items so $s_a > s_b$ or, in an alternative notation, $t_a > t_b$ where $\sigma=0$. As $a \in \bar{\mathcal{R}}'$ and $b \in \mathcal{R}'$, and as all items that are retrieved have approximate similarities greater than those that are not retrieved, it can be seen that $t'_b > t'_a$. The lines given by (σ, t_a) and (σ, t_b) therefore must cross in the interval $(0..\sigma')$. Therefore

$$t''_b > t''_a \quad \text{for all } \sigma'' > \sigma' \quad \text{when} \quad a \in \mathcal{C} \cap \bar{\mathcal{R}}',\ b \in \bar{\mathcal{C}} \cap \mathcal{R}' \tag{3}$$

Consider Eq. (1). As $|\mathcal{C}|$ is fixed, an increase in θ implies an increase in $|\mathcal{C} \cap \mathcal{R}|$. An increase in $|\mathcal{C} \cap \mathcal{R}|$ will only occur if an element of in $\mathcal{C} \cap \bar{\mathcal{R}}$ replaces an element in $\bar{\mathcal{C}} \cap \mathcal{R}$. Increasing σ cannot result in such a replacement by Eq. (3). Therefore θ is non-increasing with respect to σ. Note that θ is a function of $\mathcal{R}$ (Eq. (1)), which is a function of the values of T, which are functions of σ (Eq. (2)), so a non-increasing functional relationship ($\theta = f(\sigma)$) exists between θ and σ.

The *expected completeness* is the expected value of θ ($\mathrm{E}\theta$) over all searches. Now $\mathrm{E}\theta = 1/|\mathcal{Q}| \sum_{q \in \mathcal{Q}} f_q(\sigma)$, where $\mathcal{Q}$ is the set of queries. As each f_q is non-increasing with respect to σ, $\mathrm{E}\theta$ is non-increasing

with respect to σ. Thus minimising σ maximises $E\theta$, and as minimising σ is equivalent to minimising $\mathrm{Var}(S-T)$, minimising $\mathrm{Var}(S-T)$ maximises the expected completeness. Note, however, that not every reduction in $\mathrm{Var}(S-T)$ need lead to an increase in the expected completeness. ■

Theorem 1 makes one assumption: that the true score and the error that is introduced by creating an approximation of it are independent of one another. This assumption is unlikely to ever hold entirely. For a 1-dimensional vector the dependency will be high. However it is believed that as dimensions are added the effects due to any one dimension are likely to be obscured. This belief is tested in Section 9. Furthermore is should be noted that if the distribution of the error only shifts gradually, particularly for scores around the score used as a cutoff for retrieval, then the measure should remain fairly effective.

5. RELATED MEASURES

$\mathrm{Var}(S-T)$ is closely related to the total sum of squares (TSS) and to the correlation. First consider the relationship with the TSS. It can be seen that:

$$\mathrm{Var}(S-T) = \mathrm{E}[(S-T-\mathrm{E}S+\mathrm{E}T)^2] \tag{4}$$

$$\approx \frac{1}{n}\sum_{i=1}^{n}(s_i - t_i - \mu_s + \mu_t)^2 \tag{5}$$

where $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ are n samples of S and T respectively, and μ_S and μ_T are the sample means. Thus minimising the total sum of squares $\Sigma_{i=1}^{n}(s_i - t_i - \mu_s + \mu_t)^2$ is equivalent to minimising $\mathrm{Var}(S-T)$ for a given sample. In general it is permissible in similarity searching to replace the similarity measure used with a transformation that preserves the rank ordering of retrieved items. It is thus possible to replace S with S' such that $S' = S - \mathrm{E}S$, and to minimise $\sum_{i=1}^{n}(s_i' - t_i + \mu_t)^2$. If S is replaced by S'' such that $S'' = S - \mathrm{E}S + \mathrm{E}T$, then it is possible to minimise the TSS of $S'' - T$. However, any optimisation routine that uses the TSS must update all of the values taken by S'' each time a value taken by is T changed. It is therefore

doubtful whether minimising the TSS is any simpler than minimising $\mathrm{Var}(S-T)$.

It is also interesting to note the relationship of $\mathrm{Var}(S-T)$ to the correlation of S and T:

$$\mathrm{Var}(S-T) = \mathrm{Var}\,S + \mathrm{Var}\,T - 2\sigma_S\sigma_T\mathrm{Cor}(S,T) \tag{6}$$

$$= \mathrm{Var}\,S - 2\sigma_S\sigma_T\left[\mathrm{Cor}(S,T) - \frac{1}{2}\frac{\sigma_T}{\sigma_S}\right] \tag{7}$$

where $\sigma_S = \sqrt{\mathrm{Var}\,S}$, $\sigma_T = \sqrt{\mathrm{Var}\,T}$ and $\mathrm{Cor}(S, T)$ is the correlation of S and T. σ_s is a constant in the process of optimising T so it can be seen that minimising $\mathrm{Var}(S-T)$ is equivalent to maximising

$$\sigma_T\left[\mathrm{Cor}(S,T) - \frac{1}{2}\frac{\sigma_T}{\sigma_S}\right] \tag{8}$$

While there is a clear relationship between minimising $\mathrm{Var}(S-T)$ and maximising $\mathrm{Cor}(S, T)$, the two are not equivalent.

If, however, $T' = aT$ where a is free, then maximising $\mathrm{Cor}(S, T)$ is equivalent to minimising $\mathrm{Var}(S-T')$. It is possible to find the optimal value of a by considering the partial derivate of $\mathrm{Var}(S-aT)$ (call this V) with respect to a:

$$V = \mathrm{Var}(S - aT) \tag{9}$$

$$= \mathrm{Var}\,S + a^2\mathrm{Var}\,T - 2a\mathrm{Cov}(S,T) \tag{10}$$

$$\frac{\partial V}{\partial a} = 2a\mathrm{Var}\,T - 2\mathrm{Cov}(S,T) \tag{11}$$

where $\mathrm{Cov}(S, T)$ is the covariance of S and T. Making $(\partial V/\partial a)|_{a=a'} = 0$:

$$a' = \frac{\mathrm{Cov}(S,T)}{\mathrm{Var}\,T} \tag{12}$$

where a' is the optimal value of a. Substituting a' into Eq. (10) results in:

$$V' = \mathrm{Var}\,S[1 - \mathrm{Cor}(S,T)^2] \tag{13}$$

where $V = V'$ when $a = a'$. As for when the TSS is used as a measure, a' is dependent on all the values of T, so if the expression is used in an optimisation routine it is necessary to update a' as T changes. It will probably not be possible to replace T with T' for all indexing systems. In any case it is usually fairly complicated to calculate the correlation, so it is unlikely to be used as a measure in an optimisation routine.

6. MINIMISING ERROR IN ONE DIMENSION

It is desirable to be able to deal with each dimension independently of the others when deciding where to place the partition boundary points as this simplifies calculations and reduces computation time. To aid this, two assumptions are adopted. Firstly, it is assumed that the distributions of the elements in each dimension are independent of the elements in the other dimensions for both the item vectors and the query vectors. This assumption is likely to come closer to holding for data sets that have undergone dimension reduction, as dimension reduction tends to remove correlations between dimensions.

The second assumption is that the similarity function used may be represented as a sum of functions of the individual dimensions, *i.e.*,

$$s(\vec{x}, \vec{y}) = \sum_{i=1}^{d} s_i(x_i, y_i), \tag{14}$$

where $\vec{x} = (x_1, x_2, \ldots, x_d)$ and $\vec{y} = (y_1, y_2, \ldots, y_d)$ are two vectors. In many cases. $s_i = s_j$ for all $1 \leq i, j \leq d$. Let the value $s_i(x_i, y_i)$ be referred to as a *part-similarity*.

The dot product, a common measure, can clearly be expressed as a sum of part-similarities:

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{d} x_i y_i \tag{15}$$

While the l_p metrics cannot be expressed in this form, l_p^p can be, and as l_p^p is an order preserving transformation of l_p for $p > 0$, l_p may be replaced by l_p^p. Another common measure, the cosine measure (the dot product of two normalised vectors) [34] cannot be represented exactly

in this form as the act of normalisation introduces dependencies between the dimensions. The dependencies, however, are fairly small for long vectors and can probably be disregarded.

In order to approximate the elements of one dimension it is desirable to have a separate measure of error for each dimension. As has been seen, the similarity score may be expressed as a summation of part-similarities. T may similarly be considered as a summation of *part-approximate similarities*:

$$T = \sum_{i=1}^{d} T_i \tag{16}$$

So choosing $\mathrm{Var}(S-T)$ as the measure of error:

$$\mathrm{Var}(S-T) = \mathrm{Var}\left(\sum_{i=1}^{d} S_i - \sum_{i=1}^{d} T_i\right) \quad \text{(by Eqs. (14) and (16))} \tag{17}$$

$$= \sum_{i=1}^{d} \mathrm{Var}(S_i - T_i) \tag{18}$$

Equation (18) uses the assumption that the dimensions are independent of one another. This independence implies that the terms $S_i - T_i$ are independent of one another. Note that they are random variables and that the variances of independent random variables sum. Thus minimising $\mathrm{Var}(S_i - T_i)$ for each dimension i individually minimises $\mathrm{Var}(S-T)$ overall.

Let X be a random variable representing the elements from one dimension in a set of vectors. In order to approximate X, let the domain of X be partitioned and associate with each partition an *approximation value*. Let X' be the approximation of X (see Fig. 1).

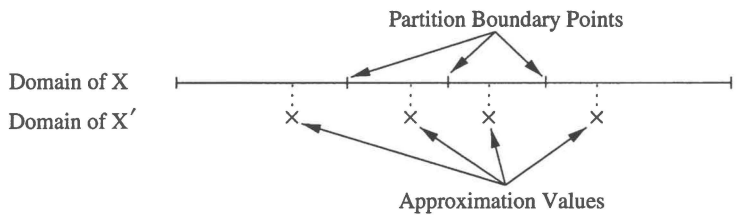

FIGURE 1 Approximating the elements in a dimension.

Although the values of X' are depicted as being drawn from the corresponding partitions, this need not necessarily be the case.

In some cases, such as in SVI and SDM, the vectors in the collection need to be transformed into binary vectors. If a transformation to a binary vector is required then the values taken by X' may be binary strings. If binary unreflected Gray codes are used to represent the partitions, then the Hamming distance between two binary vectors (*i.e.*, the number of bits which are different) is equal to the city-block distance (l_1) when assigning integers sequentially to the partitions (see Fig. 2). Note that as the values of X' in this case are restricted to being binary strings (or equivalently, integers), the minimum value that may be achieved by $\mathrm{Var}(S_i - T_i)$ is likely to be greater than that for when there is an unrestricted choice of real values. Some extra flexibility may be introduced by using a scaling factor c_i and minimising $\mathrm{Var}(S_i - c_i T_i)$ instead of $\mathrm{Var}(S_i - T_i)$. c_i may be optimised in a fashion similar to Eq. (9) to give the optimal value:

$$c_i' = \frac{\mathrm{Cov}(S_i, T_i)}{\mathrm{Var} T_i} \tag{19}$$

As for the full-vector case, $\mathrm{Cor}(S_i, T_i)$ may be maximised as an alternative to minimising $\mathrm{Var}(S_i - c_i T_i)$.

Note that many indexing techniques, such as the k-d tree, partition the search space on a dimension-by-dimension basis. One method of doing this is to divide the domain into intervals of equal size. An alternative method, which is usually superior, is to partition a dimension so that there is an equal number of elements in each partition. These methods are respectively examples of *spatial partitioning* and *data partitioning*. Neither of these methods are ideal: spatial

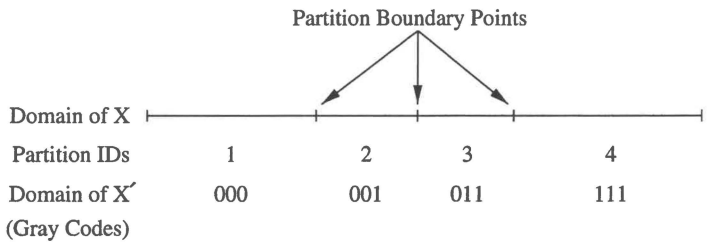

FIGURE 2 The use of Gray codes to represent partitions of an element's domain.

partitioning ignores the density of the data and data partitioning ignores the distances between the points. It is to be noted that the measure $\mathrm{Var}(S_i - T_i)$ suffers from neither of these defects. While most indexing techniques do not explicitly construct approximate vectors, it is possible to interpret their behaviour in terms of vector approximations. The act, for instance, of partitioning a data set into two parts along one dimension can be equated to approximating the values in that dimension by a 1 or a 2. It is possible, therefore, to modify many indexing systems so that they construct partitions so as to minimise $\mathrm{Var}(S_i - T_i)$ or $\mathrm{Var}(S_i - c_i T_i)$. Doing so could lead to improvements in performance, but this is yet to be tested.

7. OPTIMISING PARTITION BOUNDARY POINTS BY DIMENSION

It is possible to optimise the approximations for a dimension i by adjusting the partition boundary points and the values taken by X' so as to minimise $\mathrm{Var}(S_i - T_i)$. If possible it would be best to set these parameters so that $\mathrm{Var}(S_i - T_i)$ achieves its global minimum. No technique for finding this minimum is known, so instead a heuristic optimisation algorithm has been used.

In order to estimate $\mathrm{Var}(S_i - T_i)$, a number of vectors (n) representing items from the collection are drawn at random and paired with randomly selected vectors representing queries. The elements of dimension i from this sample are then used to calculate S_i and T_i. $\mathrm{Var}(S_i - T_i)$ is then approximated by

$$\left[\frac{1}{n}\sum_{i=1}^{n}(s_i - t_i)^2\right] - \left[\frac{1}{n}\sum_{i=1}^{n}(s_i - t_i)\right]^2 \tag{20}$$

If n is large then this is clearly a slow calculation to make. In order to reduce the time spent calculating $\mathrm{Var}(S_i - T_i)$ a simple random descent algorithm was used (see Algorithm 1). Only one partition boundary point was moved at a time, and each time only by a small step. The value of $\mathrm{Var}(S_i - T_i)$. then only requires a small adjustment after each move and does not need to be fully calculated. For l_2 and a number of other measures it is also possible to optimally set an approximation

value in near-constant time if the remaining parameters are fixed. Note that as all the parameters are interdependent the approximation value is optimal only with respect to the other current values of the other parameters. It does not produce one of the parameters of the global minimum of $\mathrm{Var}(S_i - T_i)$. Each time a partition boundary point or approximation value is changed, several statistics used for calculating $\mathrm{Var}(S_i - T_i)$ and the approximation values need to be updated. The details of these updates are quite involved and have been omitted. A number of other heuristic algorithms were tested, but all of these required $\mathrm{Var}(S_i - T_i)$ to be fully calculated after each adjustment of the parameters. The algorithm presented here clearly performed better both in terms of speed and quality of results. To start with the algorithm tries to move the partition boundary points by moderately-sized steps, and when no improvements can be found the size of the steps is reduced. The approximation values are updated frequently to take advantage of the fact that this update is a quick operation.

ALGORITHM 1 Optimisation algorithm for setting partition boundary points and approximations using incremental update of statistics

Set all partition boundary points using data partitioning
Set all approximations to be the mid-way values between boundary points

Calculate $\mathrm{Var}(S-T)$ and the statistics required for setting approximation values.

for all partitions in random order **do**
  Set an approximation value; update statistics
end for
CommonRadius = 1000 {For a sample size of 100000.}
while *commonRadius* > 0 **do**
  radius = *commonRadius*
  for all boundary points and both directions in random order **do**
    while no improvement in $\mathrm{Var}(S-T)$ and *radius* > 0 **do**
      Consider moving the boundary point in the given direction by *radius*.

      Calculate $\mathrm{Var}(S-T)$ for the prospective change.
      if an improvement **then**

```
            Set the new partition boundary point; update statistics
            for all partitions in random order do
              Set approximation value, update statistics
            end for
          end if
          radius = radius/2
        end while
      end for
      if number of moved boundary partition points ≤ 2 (or 1 for 4
partitions or fewer)
      then
        commonRadius = commonRadius/2
      end if
    end while
```

8. APPLICATION TO THE VA-FILE

The VA-file [41] is a file that contains one approximation for each of the vectors in the collection. As the approximations require less memory to store than the full vectors the VA-file is smaller than a file that contains the full vectors, and is consequently quicker to search. In the original version of the VA-file technique the scan of the VA-file was the first of two stages. The first stage identified candidate vectors (by considering the boundary points of each partition) and the second stage searched through these candidates to find the closest vectors to the query. This resulted in a complete search. The second stage does, however, significantly increase the search time. The version of the technique considered here only performs the first stage of the search, and instead of a set of candidates, a set of approximate similarity scores is produced. The resulting search is incomplete but faster than the two-stage search.

In order to create the VA-file, it is first necessary to partition the domain of each of the dimensions. Weber [41] forms partitions that contain equal numbers of elements. If approximation scores are to be calculated then it is necessary that each partition have an associated approximate value. Various values can be used, for example the median value or the value midway between the partition boundaries.

Weber uses the lower bound of the partitions (assuming a distance measure rather than a similarity measure) for ordering the candidates for subsequent search.

An alternative to this approach is to set the partitions and approximation value so as to minimise $\mathrm{Var}(S_i - T_i)$ for each dimension i. Algorithm 1 may be used for this purpose. It is desirable also to allow the dimensions to have different numbers of partitions, as this allows for the more efficient use of memory. If the overall number of bytes for a vector approximation is fixed then the number of partitions may be assigned so as to minimise $\mathrm{Var}(S-T)$ overall. For simplicity it is assumed that the number of partitions for each dimension is a power of 2. This allows each element to be approximated by a few bits that are kept independent of the other dimensions and thus simplifies the encoding and decoding routines considerably. Algorithm 2 is used to allocate the available bits between the dimensions. The allocation process involves adding and removing bits from each dimension to determine the relative effects of each on the measure $\mathrm{Var}(S-T)$. It is necessary to call the program implementing Algorithm 1 many times during this process, so consequently the process is quite slow. It was, for instance, necessary to limit each dimension to a maximum of 8 bits as it took too long to set the parameters if more bits were allocated. It would be desirable to be able to predict the number of bits required for each dimension by analysing the data distribution ahead of time. No adequate method, however, has yet been found.

ALGORITHM 2 Allocate Bits

Assign bits evenly amongst elements
while finding improvements **do**
  for $i = 1$ to d **do**
    b = current number of bits assigned to dimension i
    $increase_i = \mathrm{Var}_b(S_i - T_i) - \mathrm{Var}_{b+1}(S_i - T_i)$
    {Var_b indicates the variance given b bits. *increase* and *decrease* are lists.}
    if $b > 0$ **then**
      $decrease_i = \mathrm{Var}_{b-1}(S_i - T_i)\ \mathrm{Var}_b(S_i - T_i)$
    end if
  end for
  Sort *increase* into descending order

Sort *decrease* into descending order
for = 1 to *d* **do**
if $increase_i > decrease_i$ **then**
decrement the number of bits assigned to the dimension of the decrease

increment the number of bits assigned to the dimension of the increase

end if
end for
end while

9. RESULTS

9.1. Testing Error Dependency

In order to test the extent of the dependency of the error D on the similarity score S (see Section 4), four simple experiments of the following form were conducted. A set of tuples, both of whose elements are vectors, was generated. The Euclidean distance (S) between each pair of vectors was calculated. The domain of each of the dimensions of the vectors was partitioned, and each partition assigned an approximate value. The distance between the approximate vectors was calculated (T). The error between each pair of vectors was then $D = T - S$. The probability density function (PDF) of D was calculated for those vector pairs whose values of S lay within the largest 10%, middle 10% and smallest 10% of values. The PDFs for vectors of dimensionality 1 and 10 are shown in Figure 3. This procedure was followed for four data sets: for 1-dimensional uniformly distributed vectors, for 10-dimensional uniformly distributed vectors, for 1-dimensional normally distributed vectors, and for 10-dimensional normally distributed vectors. The standard normal distribution was used to generate the elements in each of the dimensions in the normal data sets. In the case of the uniformly distributed data the partition boundary points were placed at 0.125, 0.25, 0.375, 0.5, 0.625, 0.75 and 0.875, with the approximation values of each of the partitions being placed at the midway points between the partition boundary points (where 0 and 1 are treated as the endpoints of the end-most partitions). In the case of

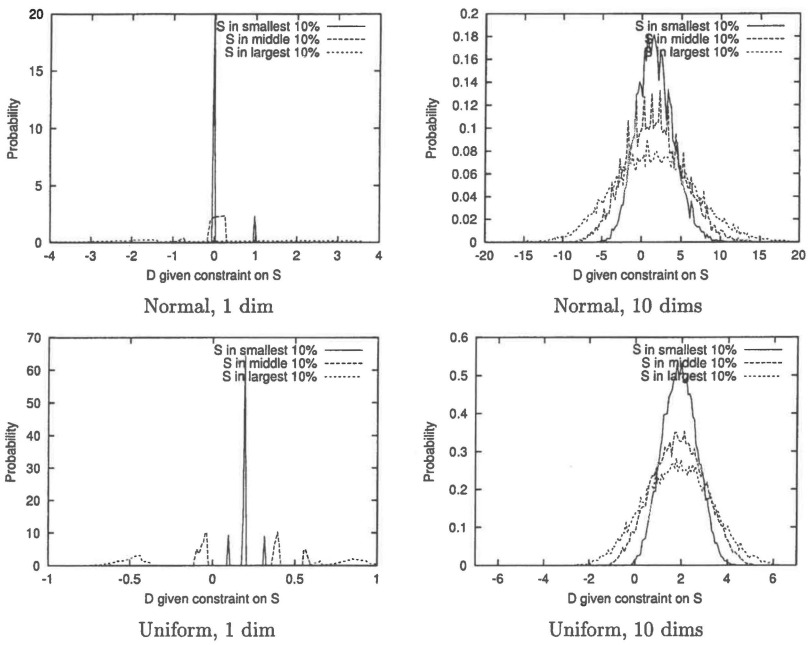

FIGURE 3 The PDFs of the error D given that the similarity S is in the largest, middle or smallest 10% of S values for 1-and 10-dimensional vectors respectively.

the normally distributed data the partition boundary points were placed at by partitioning the domain of each element at the points -3, -2, -1, 0, 1, 2 and 3, with the approximation values being -3.5, -2.5, -1.5, -0.5, 0.5, 1.5, 2.5 and 3.5. The partitioning was somewhat arbitrary in both the uniform and normal cases, but it was designed to cover the data fairly well. These graphs demonstrate that adding dimensions does reduce the dependency of the error on the similarity. When there is only one dimension there is a very strong dependency whereas when there are 10 the PDFs more closely resemble one another. Nonetheless some dependency is clearly present and this will effect the performance of $\mathrm{Var}(S-T)$ as a measure. The persistence of the dependency is probably due to the strength of the dependency in the single dimension case. The similarity measure used, which emphasises differences in individual dimensions would also have an effect. The degree of dependency that is evident for 10 dimensions does indicate that some caution is needed when using $\mathrm{Var}(S-T)$ as a measure. It is possible that even in a data set which has a very large number of dimensions, a

few may dominate in their contribution to the overall similarity scores, and to the error.

9.2. The Modified VA-File

In all of the tests presented here 100 queries were run on collections consisting of 100 000 vectors. In each case the completeness is the average percentage of the 10 items closest to the query that were actually retrieved. Six different collections were used. The *uniform* data set has 50 dimensions, with the data being uniformly distributed in a hypercube. The *normal* data set has 50 dimensions, with the elements in each dimension having a standard normal distribution. The *mixed* data set has 50 dimensions, with each dimension being distributed according to a different type of random distribution, and with the queries having different distributions to the indexed vectors. The *AustLII* data set is a 100-dimensional data set derived from performing dimension reduction on a collection of legal documents. Finally *ColourHistR1* and *ColourHistR5* are two data sets derived using colour histograms from the same collection of images taken from the Internet. The *ColourHistR1* data set contains histograms for one region and has 64 dimensions, whereas the *ColourHistR5* data set contains histograms for 5 regions and has 320 dimensions. The similarity measures used were the cosine measure (cos) and the Euclidean distance (l_2).

Table I compares the average completeness of the searches that result when the dimensions are partitioned into equal parts with that obtained by setting the parameters so as to minimise $\mathrm{Var}(S_i - T_i)$ for

TABLE I The average completeness when using data partitioning and when minimising $\mathrm{Var}(S_i - T_i)$. Each element has 16 partitions

Collection	*Data partitioning*	*Error minimisation*
Uniform (l_2)	82.4%	82.2%
Uniform (cos)	70.6%	82.6%
Normal (l_2)	32.1%	74.1%
Normal (cos)	29.1%	70.0%
Mixed (l_2)	41.5%	74.7%
Mixed (cos)	24.0%	51.2%
AustLII (cos)	31.1%	32.5%
ColourHistR1 (l_2)	33.0%	57.0%
ColourHistR5 (l_2)	42.6%	69.3%

each dimension i. In the first case the values midway between the boundaries of each partition were used as approximation values. The outer boundary of an end-most partition is taken to be the minimum or maximum of the values in the partition. In most cases optimising $\mathrm{Var}(S_i - T_i)$ produced considerably better results. In the case that ran against the trend, *Uniform* (l_2), the difference was small and possibly due to sampling error.

It is to be noted that in most cases in which both l_2 and the cosine measure were tested, performance was superior for l_2. The reason for this is not clear. One possibility, however, is that when l_2 is used items close to the centre of the query distribution are likely to be retrieved regardless of the query. Using the cosine measure is effectively the same as normalising the data, and normalising the data removes the significance of the positioning of items relative to the origin. Thus when the cosine measure is used there may be a larger number of vectors that are close to the query and the task of distinguishing between them is more difficult.

Using different numbers of partitions for each dimension may improve the quality of retrieval. Table II contains the results of tests that allow the number of partitions for each dimension to vary, and these results are compared to the situation in which the number of partitions is the same for each dimension. Algorithm 2 is used to allocate the number of partitions to for each dimension. In both cases the total number of bytes used to store a vector approximation was $\lceil d/2 \rceil$, where d is the number of dimensions. A maximum of 256 partitions was imposed on the dimensions so as to limit the amount of time required to optimise the parameters for each dimension. Only those test sets that do not have identically distributed dimensions were tested. As can be seen, allowing different numbers of bits to be

TABLE II The average completeness when fixing the number of partitions of each dimension to 16 and when allowing the bits to be redistributed

Collection	*Same number of partitions for each dimension*	*Varying number of partitions*
Mixed (l_2)	74.7%	82.7%
Mixed (cos)	51.2%	58.7%
AustLII (cos)	32.5%	35.0%
ColourHistR1 (l_2)	57.0%	82.2%
ColourHistR5 (l_2)	69.3%	91.8%

allocated to different dimensions results in improved performance in all cases. Allocating the bits to each dimension is a slow process, and a the faster technique would be desirable.

The figures for completeness presented in Tables I and II showed the number of average of items amongst the 10 closest to the query that were among the 10 items actually retrieved. In the results presented in Table III the number of items amongst the closest 10 that appear in the top 10, 20, 30, 40 and 50 items retrieved are presented. Each approximation vector was allocated $\lceil d/2 \rceil$ bytes, with dimensions having variable numbers of partitions. It is evident that if more than 10 items are retrieved then the completeness of the search rapidly increases. In most cases nearly all of the 10 items closest to the query are retrieved within the top 50 items.

The effect of using different lengths of approximation vectors was tested, and the results of these tests are presented in Table IV. The number of partitions for each dimension was allowed to vary.

TABLE III The percentage of the 10 items closest to the query, given that the first 10, 20, 30, 40 and 50 items are retrieved

Collection	10/10	10/20	10/30	10/40	10/50
Uniform (l_2)	82.8%	98.4%	100.0%	100.0%	100.0%
Uniform (cos)	82.8%	98.1%	99.7%	100.0%	100.0%
Normal (l_2)	72.7%	91.6%	97.4%	99.2%	99.7%
Normal (cos)	71.9%	92.6%	97.9%	99.3%	99.7%
Mixed (l_2)	82.7%	98%	99.9%	100.0%	100.0%
Mixed (cos)	58.7%	89.1%	98.0%	99.2%	99.9%
AustLII (cos)	35.0%	48.7%	55.1%	60.0%	63.9%
ColourHistR1 (l_2)	82.2%	97.5%	99.6%	99.9%	100.0%
ColourHistR5 (l_2)	91.8%	99.5%	99.8%	100.0%	100.0%

TABLE IV The average completeness given different lengths of approximation vector

	Average bits per element			
Collection	1	2	3	4
Uniform (l_2)	3.5%	34.8%	62.3%	82.8%
Uniform (cos)	5.2%	32.9%	60.2%	82.8%
Normal (l_2)	1.6%	20.0%	46.8%	72.7%
Normal (cos)	2.7%	18.7%	47.1%	71.9%
Mixed (l_2)	8.7%	44.9%	67.2%	82.7%
Mixed (cos)	16.4%	30.6%	42.2%	58.7%
AustLII (cos)	2.5%	15.0%	27.9%	35.0%
ColourHistR1 (l_2)	39.8%	62.3%	75.8%	82.2%
ColourHistR5 (l_2)	51.3%	72.5%	82.6%	91.8%

Approximation vectors that had an average of 1, 2, 3 and 4 bits per dimension were tested. As would be expected, using longer vector approximations increases the completeness in all cases. It appears that in general 4 or more bits per dimension are required. It is to be noted that fewer bits are required for *ColourHistR5*, with 320 dimensions, than for *ColourHistR1*, with 64 dimensions. The two are very similar data sets, so this suggests that longer vectors require fewer bits in order to achieve comparable levels of completeness.

10. CONCLUSION

A measure was presented which is suitable for judging the quality of the approximations of vectors for use in similarity searching. One assumption was made when deriving this measure. It was shown that the assumption comes closer to being met as the dimensionality of the vectors increases, so long as no dimensions dominate in their contribution to the similarity scores. Given two further assumptions regarding the independence of elements in different dimensions, the measure may be used in constructing vector approximations for a number of different indexing systems. The VA-file was selected as an example of one application, and it was seen that when the measure was used to construct the vector approximations, improved performance was the result for a number of data sets.

Acknowledgements

This research was partly funded by an Australian Research Grant. Thanks to Arthur Ramer and Graham Mann for their supervision of the research presented here, and to Roger Weber who supplied the data sets *ColourHistR1* and *ColourHistR5*.

References

[1] Agrawal, R., Lin, K., Sawhney, H. S. and Shim, K., Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: *Proceedings of the 21st International Conference on Very Large Databases*, pp. 490–501, Zurich, Switzerland, Sept., 1995.

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. and Lipman, D. J., Basic local alignment search tool. *Journal of Molecular Biology*, **215**(3), 403–410, Oct., 1990.

[3] Arya, S., *Nearest Neighbour Searching and Applications. Ph. D. thesis*, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, Maryland 20742–3275, June, 1995.
[4] Beis, J. S. and Lowe, D. G., Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: *Conference on Computer Vision and Pattern Recognition*, pp. 1000–1006, Puerto Rico, June, 1997.
[5] Bentley, J. L., Multidimensional binary search trees used for associative searching. *Communications of the ACM*, **18**(9), 509–517, Sept., 1975.
[6] Berchtold, S. and Kriegel, H.-P., *S3: Similarity search in CAD database systems*. In: Peckham [33], pp. 564–567.
[7] Berchtold, S., Böhm, C., Braunmüller, B., Keim, D. A. and Kriegel, H.-P., *Fast parallel similarity search in multimedia databases*. In: Peckham [33], pp. 1–12.
[8] Berchtold, S., Böhm, C., Jagadish, H. V., Kriegel, H.-P. and Sander, J., Independent quantization: An index compression technique for high-dimensional data spaces. *16th International Conference on Data Engineering (IDCE)*, 2000.
[9] Berchtold, S., Böhm, C. and Kriegel, H.-P. (1998). The pyramid-technique: Towards breaking the curse of dimensionality. In: Haas, L. M. and Tiwary, A. Editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, **27**(2) of *SIGMOD Record*, 142–153. ACM Press.
[10] Berchtold, S., Keim, D. and Kriegel, H.-P., The X-tree: An index structure for high-dimensional data. In: *22nd Conference on Very Large Databases*, pp. 28–39, Bombay, India, 1996.
[11] Briedis, B. J. and Gedeon, T. D., A new approach to indexing in high-dimensional space. *Australian Computer Science Communications*, **21**(2), 1–12, Jan., 1999. Database Systems 99. *Proceedings of the 10th Australiasian Database Conference, ADC'99*.
[12] Chakrabarti, K. and Mehrotra, S., The hybrid tree: An index structure for high dimensional feature spaces. In: Kitsuregawa, M., Maciaszek, L., Papazoglou, M. and Pu, C. Editors, *Proceedings Fifteenth International Conference on Data Engineering*, pp. 440–447, Sydney, Australia, Mar., 1999. IEEE Computer Society.
[13] Csillaghy, A. and Benz, A. O., Interactive image retrieval in large astronomical archives: the ASPECT system. *Solar Physics*, **188**(1), 203–216, Aug., 1999.
[14] Dasarathy, B. V., *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, California, 1991.
[15] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. and Harshman, R., Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, **41**(6), 391–407, Sept., 1990.
[16] Downs, G. M., Similarity searching and clustering of chemical-structure databases using molecular property data. *Journal of Chemical Information and Computer Sciences*, **34**(5), 1094–1102, Sept./Oct., 1994.
[17] Faloutsos, C., Ranganathan, M. and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In: Snodgrass, R. T. and Winslett, M. Editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, **23**(2) of SIGMOD Record, 419–429. ACM Press.
[18] Fayad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. Editors (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI Press/The MIT Press, Boston.
[19] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D. and Yanker, P. (1995). Query by image and query content: The QBIC system. *Computer*, **28**(9), 23–32.
[20] Friedman, J. H., Bently, J. L. and Finkel, R. A., An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, **3**(3), 209–226, Sept., 1977.
[21] Furht, B., Smoliar, S. W. and Zhang, H., *Video and Image Processing in Multimedia Systems*. Kluwer Academic Publishers, Boston, 1995.

[22] Gaede, V. and Günther, O., Multidimensional access methods. *Technical report*, Institute of Information Systems, Humboldt-Universität zu Berlin, Berlin, 1995.
[23] Gersho, A. and Gray, R. M., *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, 1992.
[24] Greene, R. L. (1994). Efficient retrieval from sparse associative memory. *Artificial Intelligence*, **66**, 395–410.
[25] Grother, P. J., Candela, G. T. and Blue, J. L., Fast implementations of nearest neighbor classifiers. *Pattern Recognition*, **30**(3), 459–465, Mar., 1997.
[26] Guenther, O. and Buchmann, A., Research issues in spatial databases. *SIGMOD Record*, **19**(4), 61–68, Dec., 1990.
[27] Guttman, A. (1984). R–trees: A dynamic index structure for spatial searching. In: Yormack, B. Editor, *SIGMOD'84, Proceedings of Annual Meeting*, **14**(2) of *SIGMOD Record*, 47–57. ACM Press.
[28] Jagadish, H. V. (1991). A retrieval technique for similar shapes. In: Clifford, J. and King, R. Editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, **20**(2) of *SIGMOD Record*, 208–217. ACM Press.
[29] Kanerva, P., *Sparse Distributed Memory*. The MIT Press, Cambridge, Massachusetts, 1988.
[30] Korn, F., Sidiropolos, N., Faloutsos, C., Siegel, E. and Protopapas, Z., Fast nearest neighbor search in medical image databases. In: Vijayaraman, T. M., Buchmann, A. P., Mohan, C. and Sarda, N. L. Editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3–6, 1996, Mumbai (Bombay), India*, pp. 215–226. Morgan Kaufmann, 1996.
[31] Kurniawati, R., Jin, J. S. and Shepherd, J. A., The SS+-tree: An improved index structure for similarity searches in a high-dimensional feature space. In: *SPIE Storage and Retrieval for Image and Video Databases V*, San Jose CA, Feb., 1997.
[32] Lin, K.-I., Jagadish, H. V. and Faloutsos, C., The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, **3**(4), 517–549, Oct., 1994.
[33] Peckham, J. Editor. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, **26**(2) of *SIGMOD Record*. ACM Press, 1997. 13–15 May, Tucson, Arizona.
[34] Salton, G., *Automatic Text Processing*: *The Transformation Analysis and Retrieval of Information by Computer*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.
[35] Salton, G., Wong, A. and Yang, C. S., A vector space model for automatic indexing. *Communications of the ACM*, **18**(11), 613–620, Nov., 1975.
[36] Samet, H., *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1990.
[37] Shepherd, J., Zhu, X. and Megiddo, N., A fast indexing method for multidimensional nearest neighbour search. In: *SPIE Conference on Storage and Retrieval for Image and Video Databases VI*, Jan., 1999. San Jose, California.
[38] Silverman, B. W., *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, Boca Raton, Florida, 1992.
[39] Subrahmanian, V. S. and Jajodia, S., Editors. *Multimedia Database Systems*: *Issues and Research Directions*. Springer, Berlin, 1996.
[40] Turk, M. and Pentland, A., Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, **3**(1), 71–86, Winter, 1991.
[41] Weber, R., Schek, H.-J. and Blott, S., A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: *Proceedings of the 24th VLDB Conference*, New York, USA, 1998.
[42] White, D. A. and Jain, R., Similarity indexing with the SS-tree. In: *Proc. 12th IEEE International Conference on Data Engineering*, New Orleans, LA, Feb., 1996.

[43] Wu, C., Berry, M., Shivakumar, S. and McLarty, J. (1995). Neural networks for full-scale protein sequence classification: Sequence encoding with singular value decomposition. *Machine Learning*, **21**, 177–193.
[44] Yang, Y. and Chute, C. G., An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems*, **12**(3), 252–277, July, 1994.
[45] Zakarauskas, P. and Ozard, J. M., Complexity analysis for partitioning nearest neighbor searching algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(6), 663–668, June, 1996.