PERGAMON

Pattern Recognition 33 (2000) 1497–1510

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Fast nearest-neighbor search algorithms based on approximation-elimination search

V. Ramasubramanian[a], Kuldip K. Paliwal[b,]*

[a]*Computer Systems and Communications Group, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay 400 005, India*
[b]*School of Microelectronic Engineering, Griffith University, Brisbane, QLD 4111, Australia*

Received 28 July 1997; received in revised form 5 January 1999; accepted 5 January 1999

Abstract

In this paper, we provide an overview of fast nearest-neighbor search algorithms based on an 'approximation–elimination' framework under a class of elimination rules, namely, partial distance elimination, hypercube elimination and absolute-error-inequality elimination derived from approximations of Euclidean distance. Previous algorithms based on these elimination rules are reviewed in the context of approximation–elimination search. The main emphasis in this paper is a comparative study of these elimination constraints with reference to their approximation–elimination efficiency set within different approximation schemes. © 2000 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Fast nearest-neighbor search; Approximation–elimination search; Partial-distance search; L_1, L_∞ constraints; Fast vector quantization encoding

1. Introduction

Nearest-neighbor search consists in finding the closest point to a query point among N points in K-dimensional space. The search is widely used in several areas such as pattern classification, nonparametric estimation, information retrieval from multi-key data bases and in image and speech data compression. Reducing the computational complexity of nearest-neighbor search is of considerable interest in these areas. In this paper, we discuss fast nearest-neighbor search in the context of vector quantization encoding. Vector quantization encoding is a powerful data compression technique used in speech coding, image coding and speech recognition [1–7]. Vector quantization encoding is the minimum-distortion quantization of a vector $\mathbf{x} = (x_1, x_2, \ldots, x_K)$, (referred to as the *test vector*) using a given set of N K-dimensional *codevectors* called the *codebook* $\mathbf{C} = \{\mathbf{c}_i\}_{i=1,\ldots,N}$, of size N, under a given distance measure $d(\mathbf{x}, \mathbf{y})$. This involves finding the nearest-neighbor of $\mathbf{x}$ in $\mathbf{C}$, given by $\mathbf{q}(\mathbf{x}) = \mathbf{c}_j$: $d(\mathbf{x}, \mathbf{c}_j) \leqslant d(\mathbf{x}, \mathbf{c}_i)$, $i = 1, \ldots, N$ which requires N distance computations $d(\mathbf{x}, \mathbf{c}_i)$ using the exhaustive full search.

1.1. Elimination-based fast nearest-neighbor search

The basic structure of the sequential full-search which obtains the nearest-neighbor codevector $\mathbf{c}_j$ is as follows:

```
$d_{cnn} = \infty$ (a very large number)
For $i = 1, \ldots, N$
    $d_i = d(\mathbf{x}, \mathbf{c}_i)$
    if $d_i < d_{cnn}$ then $j = i$; $d_{cnn} = d_i$
next $i$
```

Here, at any given stage in the search, $\mathbf{c}_j$ is the current-nearest-neighbor and the current-nearest-neighbor ball $b(\mathbf{x}, d_{cnn})$ is the surface defined by $\mathbf{x} : d(\mathbf{x}, \mathbf{c}_j) = d_{cnn}$. For the Euclidean distance between $\mathbf{c}_j = (c_{jk}, k = 1, \ldots, K)$ and $\mathbf{x} = (x_k, k = 1, \ldots, K)$ given by $d(\mathbf{x}, \mathbf{c}_j) = [\sum_{k=1}^{K} (x_k - c_{jk})^2]^{1/2}$, the current-nearest-neighbor ball

* Corresponding author. Tel.: + 61-7-3875-6536; fax: + 61-7-3875-5198.
E-mail address: k.paliwal@me.gu.edu.au (K.K. Paliwal).

is a hypersphere of radius d_{cnn} with center at $\mathbf{x}$. The sequential full-search finds the nearest-neighbor by progressively updating the current-nearest-neighbor $\mathbf{c}_j$ when a codevector is found closer to the test vector than the current-nearest-neighbor, with each update shrinking the current-nearest-neighbor ball radii to the actual nearest-neighbor distance. The final nearest-neighbor is one of the codevectors inside the current-nearest-neighbor ball at any stage of the search and consequently, the current-nearest-neighbor ball assumes importance as a geometric object in defining the primary search space of interest. In the case of sequential full-search, the location of a codevector is determined with respect to the current-nearest-neighbor ball by computing the distance of each successive codevector to the test vector for comparison with the current-nearest-neighbor distance. As a result, the distances of all N codevectors to the test vector are computed with the search complexity being N distance computations per test vector.

A direct approach in reducing the complexity of the above sequential 'exhaustive full-search' is to reject a codevector by a 'quick-elimination'—without computing its actual distance to the test vector by using computationally less expensive rules to determine which codevectors cannot be nearer to the test vector than the current-nearest-neighbor. This is equivalent to approximating the current-nearest-neighbor ball by geometric constraints which allow easy elimination of codevectors which do not lie inside the current-nearest-neighbor ball.

Given the current-nearest-neighbor ball $b(\mathbf{x}, d_{cnn})$ let $E(\mathbf{x}, d_{cnn})$ represent the spatial approximation of the ball by some geometric constraint such that $b(\mathbf{x}, d_{cnn}) \subseteq E(\mathbf{x}, d_{cnn})$. Then, codevectors which do not belong to $E(\mathbf{x}, d_{cnn})$ cannot be inside $b(\mathbf{x}, d_{cnn})$ and elimination consists of rejecting these codevectors. The efficiency of elimination depends on two intrinsic factors of the elimination constraint: (i) the relative cost of the elimination computation with respect to the distance computation and (ii) the number of codevectors retained after elimination in comparison to the actual number of codevectors inside the current-nearest-neighbor ball. This is essentially governed by the computational cost in determining whether a codevector does not belong to $E(\mathbf{x}, d_{cnn})$ and the extent to which $E(\mathbf{x}, d_{cnn})$ approximates $b(\mathbf{x}, d_{cnn})$ spatially in terms of volume. Fig. 1 illustrates this with a simple geometric constraint used in one of the earliest elimination-based fast search [8] and which forms the basis of hypercube-based elimination search reported subsequently [9–12]. Here, the geometric constraint is simply the projection of the current-nearest-neighbor ball $b(\mathbf{x}, d_{cnn})$ on to a coordinate axis k, given by $E(\mathbf{x}, d_{cnn}) = \{\mathbf{y}: |y_k - x_k| < d_{cnn}\}$, i.e., $E(\mathbf{x}, d_{cnn})$ is the region between the hyperplanes $H' = \{\mathbf{y}: y_k = x_k - d_{cnn}\}$ and $H'' = \{\mathbf{y}: y_k = x_k + d_{cnn}\}$. A direct elimination is achieved by rejecting the codevectors $\{\mathbf{c}_i : c_{ik} > x_k + d_{cnn}$ or $c_{ik} < x_k - d_{cnn}\}$, incurring atmost two scalar

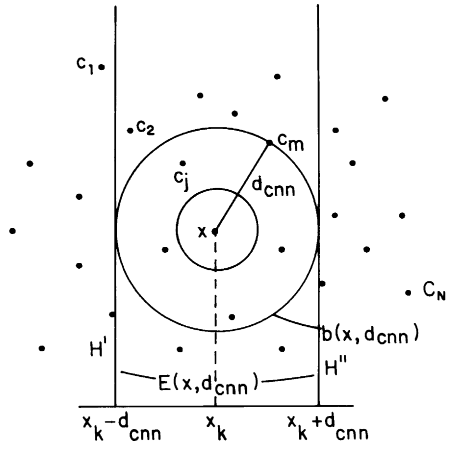

Fig. 1. Example of geometric constraint for elimination.

comparisons to reject a codevector as against a vector distance computation. (This elimination is carried out more efficiently as in Ref. [8], using a binary search on a list of ordered coordinates of the codevectors to eliminate codevectors whose coordinates fall outside the projections of the current-nearest-neighbor ball $b(\mathbf{x}, d_{cnn})$.)

Thus, in the sequential full-search, given a current-nearest-neighbor ball, the code-book is reduced by applying the elimination rule $\mathbf{C} = \mathbf{C} - \{\mathbf{c}_i : \mathbf{c}_i \notin E(\mathbf{x}, d_{cnn})\}$. Since the current-nearest-neighbor ball shrinks for every update of the current-nearest-neighbor, i.e., when a codevector closer to the current-nearest-neighbor has bean found, the above elimination need to be done only for every update of the current-nearest-neighbor using the $E(\mathbf{x}, d_{cnn})$ corresponding to the updated d_{cnn}. The codebook size thus reduces progressively with the aid of the elimination step and results in a reduction in the number of distances computed if each elimination step rejects some codevectors.

1.2. Approximation-elimination search

In addition to the intrinsic nature of the geometric constraint in approximating any given current-nearest-neighbor ball volume, another important aspect influencing the elimination efficiency of the constraint is the size of the current-nearest-neighbor ball. Clearly, for a given constraint, the elimination efficiency is better for a smaller current-nearest-neighbor ball. Hence the relative closeness of the current-nearest-neighbor codevector to the test vector and the sequence in which the test vectors are selected for distance computation influences the rate at which the codebook size is reduced in the elimination based search.

In general, in the sequential search with the elimination step, selecting a codevector close to the test vector in the initial stages of the search will help in the elimination of a large number of codevectors in the early steps of the search thereby avoiding the distance computations of these codevectors. For instance, considering the codevectors in the order of decreasing distance to the test vector represents a worst-case condition, since no geometric constraint of the current-nearest-neighbor ball can eliminate any codevector at each update. In contrast, selecting the codevectors in the order of increasing distance represents the best possible situation, since the search then starts with the actual nearest-neighbor and subsequent search merely amounts to verifying that no other codevector lies within the given elimination-constraint approximating the current-nearest-neighbor ball. In this case, the elimination step rejects the maximum number of codevectors in the first step itself and subsequent search is performed on a reduced set of codevectors. The resultant overall reduction in search complexity is then maximum for a given elimination-constraint.

Thus, the distances $d(\mathbf{x}, \mathbf{c}_i), i = 1, \ldots, N$ used in the search provides the exact information for the best ordering in which the codevectors should be considered for search under the elimination based sequential search. However, getting this exact ordering information amounts to a full-search. Therefore, in order to gain any advantage in selecting codevectors (for distance computation) such that the distance to codevectors close to the test vector are computed in the early stages of search, it is necessary to characterize the closeness of a codevector to the test vector using some approximation of the distance measure; such an explicit 'codevector selection' step is referred to as an 'approximation-step' [13]. In order to gain any advantage from the approximation step in reducing the complexity of the search, the approximation criterion has to be computationally less expensive than the actual distance measure being used in the search, and should characterize the spatial organization of the codevectors with respect to the test vector so as to reflect their relative closeness to the test vector.

The approximation–elimination procedure as described above can be given as a fast-search generalization over the sequential full-search as follows:

```
$d_{cnn} = \infty$ (a very large number)
Do until codebook $\mathbf{C}$ is empty
    $\mathbf{c}_i = \arg\min_{\mathbf{c}_l \in \mathbf{C}} \mathrm{Approx}(\mathbf{x}, \mathbf{c}_l)$   [Approximation]
    $d = d(\mathbf{x}, \mathbf{c}_i)$   [Distance computation]
    if $d < d_{cnn}$ then
        $j = i; d_{cnn} = d$
        $\mathbf{C} = \mathbf{C} - \{\mathbf{c}_m : \mathbf{c}_m \notin E(\mathbf{x}, d_{cnn})\}$   [Elimination]
    endif
enddo
```

In the above, the approximation and elimination costs incurred in the computation of arg $\min_{c_l \in \mathbf{C}}$ *Approx*$(\mathbf{x}, \mathbf{c}_l)$ and $\{\mathbf{c}_m : \mathbf{c}_m \notin E(\mathbf{x}, d_{cnn})\}$ in comparison to the number of distance computations saved by these operations crucially determine the overall reduction in the search complexity. The important issue in achieving fast search under the approximation–elimination framework is in finding efficient approximation and elimination criteria which are computationally less expensive than the distance computations but at the same time provide efficient approximation of the distance measure and the current-nearest-neighbor ball volume.

1.3. Previous work

The main elimination rules which have been employed for fast nearest-neighbor search are the partial distance elimination, hypercube elimination, elimination based on the absolute-error-inequality approximation of the Euclidean distance and the triangle-inequality-based elimination. Some of the earlier papers which are based on these elimination rules and which have addressed the issue of determining a 'good' search sequence for a given test vector to improve the efficiency of elimination schemes are [8–26].

1.3.1. Partial-distance search

A simple but efficient elimination-based search which offers improvement over the sequential exhaustive full-search is the 'partial distance search' method [10,27]. This is applicable to cases where the partial-vector distance attains the full-vector distance in a monotonically nondecreasing manner with the addition of the vector component distances. This allows a codevector to be rejected on the basis of its partial or accumulated distance, i.e., without completing the total distance computation — the elimination being carried out during the distance computation itself. In an earlier paper [28], we have addressed the role of the approximation step in the partial distance-based elimination. Here, we had proposed an ordering of codevectors according to the sizes of their corresponding clusters, i.e., sequencing the codevectors in the order of decreasing probability of being the nearest-neighbor to a given test vector, to improve the savings in computation achieved by the partial distance elimination. The ordering is done in a preprocessing stage and serves as an implicit approximation step in the actual search by providing a favorable sequence of codevectors for the partial distance based elimination. In addition, it was noted that the codebooks obtained at the end of the training process using the clustering algorithms such as the Linde–Buzo–Gray algorithm [29] have arbitrary orderings and are not guaranteed to be arranged in the favorable order. In addition, the implications of dimensionality on the additional complexity reduction due to the proposed ordering was brought out using the asymptotic equipartition property of block coding and it was shown that the proposed favorable ordering would be most useful for codebooks designed on low entropy distributions such as when the dimensionality is low.

1.3.2. Triangle-inequality-based fast search

Another important elimination rule which has been used extensively for fast nearest-neighbor search is the triangle inequality-based elimination, applicable when the distance measure is a metric [13–52]. This elimination rule corresponds to a hyperannulus constraint of the search space, where codevectors lying outside the hyperannulus region formed between two concentric hyperspheres centered at an 'anchor point' are eliminated. The efficiency of the triangle-inequality (or hyperannulus) elimination increases with the number of distinct anchor points used, as the intersection volume of the hyperannulus corresponding to the multiple anchor points becomes more localized, consequently retaining less codevectors after elimination.

The different applications in which triangle-inequality-based elimination has found use and motivated development of fast algorithms are information retrieval from data bases [31], document retrieval [32], non-parametric classification [33], fast isolated word recognition [13,38,40,50,51], and fast vector quantization of speech and image data [20,41,43]. The search based on the triangle inequality elimination in most algorithms is based on the interpoint distances computed during a preprocessing phase and set in a branch and bound framework where the search is organized using a hierarchical decomposition of clustered data into a tree structure [13,31–33,36–39,42,49].

The main algorithmic aspects which have received attention under this framework have been the determination of the clusters and the anchor points within each cluster [31,32], efficient hierarchical decomposition and tree structuring of the given points [33,36,39], improvements in providing additional constraints for elimination [37], the issue of optimum number and location of the fixed anchor points [34], [41], [44–46,48,52], and optimum use of a given set of precomputed interpoint distances by using procedures which approximate the missing interpoint distances [42].

1.4. Organization of the paper

In this paper, we are concerned mainly with the class of elimination rules based on the approximations of Euclidean distance, namely, hypercube elimination and elimination based on the absolute-error-inequality approximation of the Euclidean distance. The main emphasis of this paper is to provide an overview of fast search using these elimination rules under an explicit 'approximation–elimination' framework. In addition, the paper is particularly oriented towards providing additional geometric insight to the main approximation-elimination schemes in an unified framework and empirical studies characterizing the approximation-elimination efficiency of the different approximation–elimination search in detail.

In Section 2, algorithms based on the L_∞, L_1 approximation of the L_2 norm are considered with reference to earlier work related to the hypercube constraint of the search space [9,11], the L_∞-based 'minmax' algorithm [10] and its subsequent improvement using the absolute-error-inequality-based elimination [12]. These algorithms are viewed under the approximation–elimination framework and geometric details, not brought out originally in these algorithms, are provided for additional insight. The absolute-error-inequality is noted to be the general equivalence of the L_1 and L_2 norms. The L_∞, L_1-based elimination criterion are geometrically seen as based on the constrained minimization and maximization of the L_∞, L_1 norms given L_2 norm. Approximation–elimination based on the L_1 constraint is shown to be more efficient than the L_∞ constraint based on geometric interpretation of the norm equivalence relationship and the volume ratios of the L_1 surface and the L_∞ (hypercube) surface for a given L_2 ball. The relative efficiencies of these approximation–elimination schemes are characterized and studied with respect to dimensionality.

2. Search based on L_∞, L_1 approximation

One of the earliest elimination-based fast search [8] is based on the projections of the codevectors on a coordinate axis and forms the basis of the L_∞ (hypercube) search. This algorithm represents a classic example of search under the approximation–elimination framework. The basic principle behind this algorithm, as illustrated in Fig. 1, is the use of the projection of the current-nearest-neighbor ball $b(\mathbf{x}, d_{cnn})$ on to a coordinate axis k, as a simple geometric constraint for elimination. This is given by $E(\mathbf{x}, d_{cnn}) = \{\mathbf{y} : |y_k - x_k| < d_{cnn}\}$, i.e., $E(\mathbf{x}, d_{cnn})$ is the region between the hyperplanes $H' = \{\mathbf{y} : y_k = x_k - d_{cnn}\}$ and $H'' = \{\mathbf{y} : y_k = x_k + d_{cnn}\}$.

The algorithm given in Ref. [8] can be described under an approximation–elimination framework as follows: In a preprocessing step, the codevector coordinates are ordered on some coordinate axis k and the corresponding indices stored. During the actual search, for a given test vector $\mathbf{x}$, the codevectors are examined in the order of their projected distance from $\mathbf{x}$ on axis k, i.e., the approximation criterion is simply arg $\min_{\mathbf{c}_i \in \mathbf{C}} |x_k - c_{ik}|$. Elimination is done by using two binary searches to truncate the ordered list of codevectors so as to retain only those codevectors whose coordinates fall within the projections of the current-nearest-neighbor ball $b(\mathbf{x}, d_{cnn})$.

This algorithm can be notionally represented in the approximation–elimination form as follows:

```
$d_{cnn} = \infty$ (a very large number)
Do until codebook $\mathbf{C}$ is empty
    $\mathbf{c}_i = \arg\min_{\mathbf{c}_l \in \mathbf{C}} |x_k - c_{il}|$; $\mathbf{C} = \mathbf{C} - \mathbf{c}_i$
    $d = d(\mathbf{x}, \mathbf{c}_i)$
    **if** $d < d_{cnn}$ **then**
        $j = i$; $d_{cnn} = d$
        $\mathbf{C} = \mathbf{C} - \{\mathbf{c}_m : |c_{mk} - x_k| \geqslant d_{cnn}\}$
    **endif**
enddo
```

Friedman et al. [8] analyze the performance of this algorithm for uniform distribution and Euclidean distance and obtain an upper bound for the average number of codevectors examined for finding k nearest-neighbors as $\pi^{-1/2}[kKT(K/2)]^{1/K}(2N)^{1-1/K}$. This performance efficiency decreases rapidly with increase in dimension K; for instance, the average number of distances computed for $K = 8$ and $N = 1000$ being about 600 as reported in the simulation results of Ref. [8]. This is primarily due to the fact that use of only one axis for approximation and elimination provides a very poor constraint for higher dimensions.

2.1. Hypercube-based elimination

A direct generalization of the above single-axis-based search is to use all the axis $k = 1, \dots, K$ to provide an improved geometric constraint around the current-nearest-neighbor ball. This results in the hypercube-based elimination search as used in subsequent fast search algorithms [9–12]. The geometric constraint in the hypercube-based elimination is $E(\mathbf{x}, d_{cnn}) = \{\mathbf{y} : |y_k - x_k| < d_{cnn}, \forall k = 1, \dots, K\}$, i.e., $E(\mathbf{x}, d_{cnn})$ is the smallest hypercube formed by the $2K$ hyperplanes given by $H'_k = \{\mathbf{y} : y_k = x_k - d_{cnn}\}$ and $H''_k = \{\mathbf{y} : y_k = x_k + d_{cnn}\}$ for $k = 1, \dots, K$, containing the current-nearest-neighbor ball. Elimination by the hypercube constraint can be realized within a full-search structure as a part of the squared-error distance computation, by directly checking if each codevector is within the hypercube. By this, a codevector $\mathbf{c}_i$ can be rejected if $c_{ik} > x_k - d_{cnn}$ or $c_{ik} > x_k + d_{cnn}$ for any $k = 1, \dots, K$, thus requiring 1 to $2K$ scalar comparisons for the rejection of a codevector. Alternately, a codevector $\mathbf{c}_i$ can be rejected, as in Ref. [11], by checking if each of the component distances $|c_{ik} - x_k|$ is greater than the current-nearest-neighbor distance d_{cnn}. By this, a codevector can be rejected with a cost of 1 to K scalar subtractions and comparisons prior to a full distance computation. Any codevector passing all the K tests is inside the hypercube inscribing the current-nearest-neighbor ball and is therefore tested by the full distance to determine if it is inside the current-nearest-neighbor ball.

However, a computationally more elegant and simpler scheme for hypercube-based elimination was proposed much earlier by Yunck [9]. This is based on the observation that the hypercube is defined by $HC(\mathbf{x}, d_{cnn}) = \prod_{k=1}^{k}(x_k - d_{cnn}, x_k + d_{cnn})$ and the codevectors contained within the hypercube can be obtained as $\bigcap_{k=1}^{K} S_k$ where $S_k = \{\mathbf{c}_i : x_k - d_{cnn} \leqslant c_{ik} \leqslant x_k + d_{cnn}\}$. For a specified hypercube, the sets S_k, $k = 1, \dots, K$ are determined using two binary searches on ordered codevector indices on each of the axis and the subset of codevectors within the hypercube given by their intersection is determined by a simple and efficient multiset-intersection method in Ref. [9]. The basic algorithm in Ref. [9] based on this procedure is mainly directed towards finding the nearest-neighbor under the L_∞ metric (or Minkowski 'max-metric') and can also be easily applied for search under other general L_p metric (Minkowski p-metrics).

The L_p metric is given by $L_p(\mathbf{x}, \mathbf{y}) = (\sum_{k=1}^{K} |x_k - y_k|^p)^{1/p}$, and the L_∞ metric, obtained as $\lim_{p\to\infty} L_p(\mathbf{x}, \mathbf{y})$, is given by $L_\infty(\mathbf{x}, \mathbf{y}) = \max_{k=1,\dots,K} |x_k - y_k|$. The isometric surface determined by the L_∞ metric is an hypercube of side $2L_\infty(\mathbf{x}, \mathbf{y})$ centered at $\mathbf{x}$ and with the vector $\mathbf{y}$ lying on one of its sides. The hypercube based search constraint for a current-nearest-neighbor ball of radius d_{cnn} centered at $\mathbf{x}$, is thus the isometric L_∞ surface $\{\mathbf{y} : L_\infty(\mathbf{x}, \mathbf{y}) = d_{cnn}\}$ inscribing the current-nearest- neighbor ball. This constraint eliminates all codevectors $\mathbf{c}_i$ whose $L_\infty(\mathbf{x}, \mathbf{c}_i)$ distance is greater than d_{cnn}. This is essentially a part of the general equivalence[1] between L_2 and L_∞ norms given by

$$L_2(\mathbf{x}, \mathbf{y})/\sqrt{K} \leqslant L_\infty(\mathbf{x}, \mathbf{y}) \leqslant L_2(\mathbf{x}, \mathbf{y}).$$

This is illustrated in Fig. 2. Fig. 2(a) shows the bounds on $L_\infty(\mathbf{x}, \mathbf{y})$ given $L_2(\mathbf{x}, \mathbf{y}) = d$. The lower bound $L'_\infty(\mathbf{x}, \mathbf{y}) = d/\sqrt{K}$ and the upper $L''_\infty(\mathbf{x}, \mathbf{y}) = d$ correspond to $\mathbf{y}$ being at A and B, respectively, for a given $\mathbf{x}$. This can also be seen geometrically as in Fig. 2(b) which shows the bounds on $L_2(\mathbf{x}, \mathbf{y})$ given $L''_\infty(\mathbf{x}, \mathbf{y}) = d$. The lower bound $L'_2(\mathbf{x}, \mathbf{y}) = d$ and the upper bound $L''_2(\mathbf{x}, \mathbf{y}) = d\sqrt{K}$ correspond to $\mathbf{y}$ being at A and B, respectively, for a given $\mathbf{x}$. The above inequalities describing the norm equivalence correspond to constrained minimization, maximization of the L_∞ norm given $\mathbf{x}$ and $\{\mathbf{y} : L_2(\mathbf{x}, \mathbf{y}) = d\}$ or the constrained minimization, maximization of the L_2 norm given $\mathbf{x}$ and $\{\mathbf{y} : L_\infty(\mathbf{x}, \mathbf{y}) = d\}$.

[1] Two metrics d_1 and d_2 on a space $\mathbf{X}$ are equivalent if there exist constants $0 < c_1 < c_2 < \infty$ such that $c_1 d_1(\mathbf{x}, \mathbf{y}) \leqslant d_2(\mathbf{x}, \mathbf{y}) \leqslant c_2 d_1(\mathbf{x}, \mathbf{y}) \forall \mathbf{x}, \mathbf{y} \in X$. Alternatively, the inequality can also be given as: there exist constants $0 < e_1 < e_2 < \infty$ such that $e_1 d_2(\mathbf{x}, \mathbf{y}) \leqslant d_1(\mathbf{x}, \mathbf{y}) \leqslant e_2 d_2(\mathbf{x}, y) \forall \mathbf{x}, \mathbf{y} \in \mathbf{X}$. The L_p norms satisfy this equivalence in general. The equivalence between L_2 and L_∞ norms forming the hypercube constraint and L_2 and L_1 norms forming the absolute-error-inequality constraint (Section 2.3) directly follow from this.

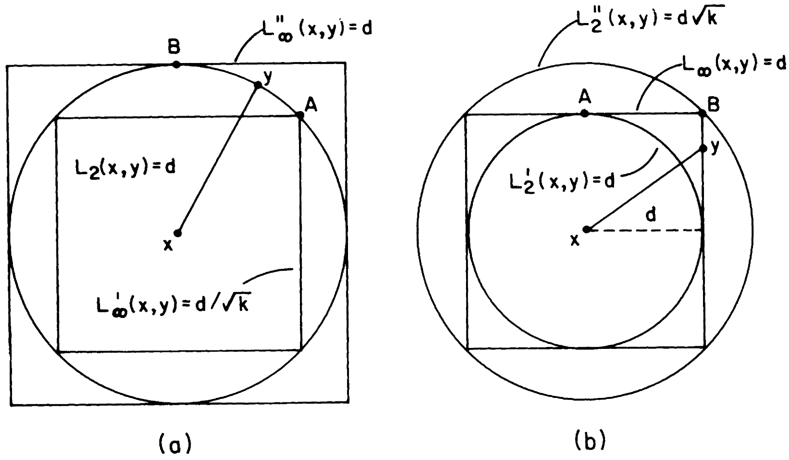

Fig. 2. Equivalence of relationship between L_∞ and L_2 norms: (a) lower (L'_∞) and upper (L''_∞) bounds of $L_\infty(\mathbf{x}, \mathbf{y})$ given $\mathbf{x}$ and $L_2(\mathbf{x}, \mathbf{y}) = d$ and, (b) lower (L'_2) and upper (L''_2) bounds of $L_2(\mathbf{x}, \mathbf{y})$ given $\mathbf{x}$ and $L_\infty(\mathbf{x}, \mathbf{y}) = d$.

Finding the nearest neighbor under the L_∞ metric thus basically involves finding the smallest hypercube with no codevectors inside it, but with one codevector on one of its sides. In Ref. [9], this is achieved by means of an expanding hypercube procedure, where the search starts with a small hypercube and expands the hypercube until it contains exactly one codevector on its side and no codevectors inside. For search with the general L_p metric, the search consists of essentially carrying out a full-search within the small subset of codevectors falling within an initially small, expanding hypercube until the current-nearest-neighbor ball is within the hypercube. This is illustrated for search with the L_2 norm in Fig. 3. Here, the codevector $\mathbf{c}_5$ is the nearest neighbor of the test vector under L_∞ norm. However, the L_2 nearest neighbor is codevector $\mathbf{c}_4$ which is closer to $\mathbf{x}$ than $\mathbf{c}_5$. (This can be understood based on the equivalence relation shown in Fig. 2.) The L_2 nearest neighbor is determined in a second step by using the hypercube corresponding to $L_2(\mathbf{x}, \mathbf{c}_5)$. The main step in this algorithm by Yunck [9] is the determination of the subset of codevectors within a specified hypercube using the multi-set intersection procedure described before. An important factor determining the efficiency of the search is the size of the initial hypercube; if no codevectors are found inside the cube the search incurs additional overheads in appropriately expanding the cube. The main complexity of the search is in this 'cube-finding' step which is estimated to be proportional to KN in terms of shift-operations as implemented in Ref. [9].

The hypercube method in Ref. [10] is an identical search procedure, with the additional feature being the analysis pertaining to finding the optimal initial hypercube size for a uniform distribution. The optimality is with respect to finding an effective trade off between the

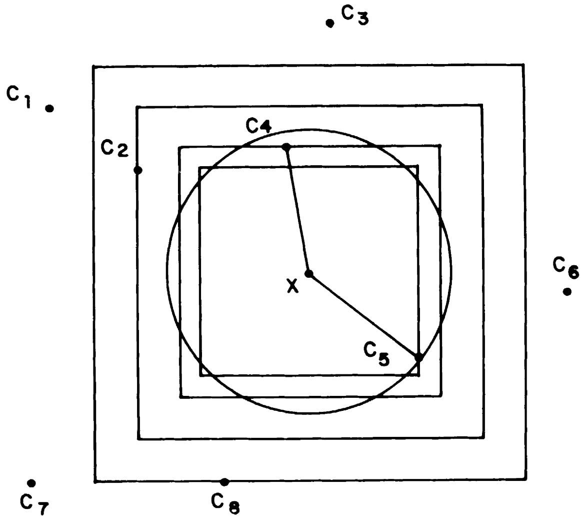

Fig. 3. Approximation by $L_\infty(\mathbf{x}, \mathbf{c}_i)$; $\mathbf{c}_5 = \mathrm{argmin}_{\mathbf{c}_i \in \mathbf{C}} L_\infty(\mathbf{x}, \mathbf{c}_i)$; $\mathbf{c}_4 = \mathrm{argmin}_{\mathbf{c}_i \in \mathbf{C}} L_2(\mathbf{x}, \mathbf{c}_i)$.

two complexities: (i) the overhead complexity in expanding the hypercube until it contains a codevector and (ii) the average number of codevectors contained in the initial hypercube which determines the main complexity of number of distances calculated in performing a full-search in the subset of codevectors within the hypercube. Here, it is shown that the average number of codevectors for the optimal initial hypercube thus determined is approximately equal to 2^K, the average complexity of the search thus increasing exponentially with dimension. In terms of determining the subset of codevectors within a specified hypercube, Ref. [10] uses a multi-set-intersection method similar to that proposed earlier in Ref. [9].

2.2. Minmax method

In the 'minmax' method [10], the L_∞ metric is used explicitly as an approximation criterion in selecting the initial codevector. Subsequent elimination is carried out using the L_∞ values computed between the test vector and the codevectors for approximation. This algorithm can be given in the approximation-elimination form as follows:

```
$d_{cnn} = \infty$ (a very large number)
Do until codebook $\mathbf{C}$ is empty
    $\mathbf{c}_i = \arg\min_{\mathbf{c}_l \in \mathbf{C}} L_\infty(\mathbf{x}, \mathbf{c}_l)$; $\mathbf{C} = \mathbf{C} - \mathbf{c}_i$
    $d = d(\mathbf{x}, \mathbf{c}_i)$
    if $d < d_{cnn}$ then
        $j = i$; $d_{cnn} = d$
        $\mathbf{C} = \mathbf{C} - \{\mathbf{c}_m : L_\infty(\mathbf{x}, \mathbf{c}_m) \geqslant d_{cnn}\}$
    endif
enddo
```

The above search is carried out by computing $L_\infty(\mathbf{x}, \mathbf{c}_i)$ for $i = 1, \ldots, N$ in a first step. After the first elimination using the L_∞ values, the remaining codevectors are ordered according to their L_∞ distance to the test vector such that subsequently, approximation consists in examining the codevectors sequentially in the order of increasing L_∞ distances to the codevectors. This can be viewed as a direct generalization over the single-axis projection-based selection procedure of Ref. [8].

The main shortcoming in the minmax method is the high overhead complexity of the L_∞-based approximation. The initial L_∞-based approximation is essentially a full-search using the L_∞ distance and has a cost of $[0, NK, (N-1) + N(K-1)]$ macs (multiplications, additions, comparisons) per vector. This initial approximation step of the search itself has an additional overhead cost of $N(K-1)$ comparisons over the L_2-distance-based full-search and the search complexity reduction here is mainly with respect to reducing the NK multiplications per test vector to a low value.

2.3. Absolute error inequality

Soleymani and Morgera [12] proposed an algorithm as an improvement over the minmax method of Ref. [10] with two additional features: (i) it reduces the overhead cost of determining the initial codevector by L_∞ approximation by using a procedure which is essentially a partial-distance realization of the L_∞-based full-search, and (ii) it uses an additional elimination rule termed the 'absolute error inequality' to eliminate codevectors during the distance computation alongwith the hypercube elimination. The absolute error inequality is based on the observation that

$$\text{if} \sum_{k=1}^{K} (x_k - c_{ik})^2 < d^2 \quad \text{then} \sum_{k=1}^{K} |x_k - c_{ik}| < d\sqrt{K}. \qquad (1)$$

A codevector $\mathbf{c}_i$ which does not satisfy $\sum_{k=1}^{K} |x_k - c_{ik}| < d\sqrt{K}$, i.e., whose absolute error is greater than $d\sqrt{K}$ will therefore have its squared error distance greater than d^2 and can be rejected on the basis of its absolute error without having to compute the more expensive squared error distance. This algorithm can be given in terms of the following steps:

- Select initial codevector $\mathbf{c}_i$ as L_∞ nearest neighbor in a full-search with partial distance realization:
- Distance computation: $d_{cnn} = d(\mathbf{x}, \mathbf{c}_i)$
- Perform full-search of codebook $\mathbf{C}$ using (i) hypercube and (ii) absolute error inequality elimination rules during distance computation with each codevector $\mathbf{c}_m$ in $\mathbf{C}$. Its distance to the test vector $d(\mathbf{x}, \mathbf{c}_m)$ is computed and compared with the current-nearest-neighbor distance d_{cnn} only if the codevector passes these two tests. The two elimination rules are carried out as follows for each codevector $\mathbf{c}_m \in \mathbf{C}$:

- Reject $\mathbf{c}_m$ if $|x_k - c_{mk}| > d_{cnn}$, $k = 1, \ldots, K$.
- Reject $\mathbf{c}_m$ by partial distance method in the computation of the absolute error $\sum_{k=1}^{K} |x_k - c_{mk}|$ with the reference absolute error distance being $d_{cnn}\sqrt{K}$.

2.4. L_1-approximation-based search

In this section, we highlight the 'absolute error inequality'-based elimination in the above algorithm and interpret it geometrically as a space constraint provided by the smallest L_1 surface inscribing the current-nearest-neighbor ball, i.e., the given L_2 norm. We consider this notion with additional details and compare the L_∞ and L_1-based approximation–elimination using a search procedure in the minmax form. The proposed search procedure using the L_1 values for approximation–elimination is a more efficient realization of L_1-based elimination than that of Ref. [12] and is shown to offer lower complexity than the minmax procedure [10] which uses the L_∞ values for approximation–elimination.

The absolute error $\sum_{k=1}^{K} |x_k - c_{ik}|$ in the inequality used in Ref. [12] is the L_1 norm between $\mathbf{x}$ and $\mathbf{c}_i$ and inequality (1) is basically the equivalence between L_1 and L_2 norms given by

$$L_2(\mathbf{x}, \mathbf{y}) \leqslant L_1(\mathbf{x}, \mathbf{y}) \leqslant L_2(\mathbf{x}, \mathbf{y})\sqrt{K}. \qquad (2)$$

This is illustrated in Fig. 4. Fig. 4(a) shows the bounds on $L_1(\mathbf{x}, \mathbf{y})$ given $L_2(\mathbf{x}, \mathbf{y}) = d$. The lower bound $L_1'(\mathbf{x}, \mathbf{y}) = d$ and the upper bound $L_1''(\mathbf{x}, \mathbf{y}) = d\sqrt{K}$ correspond to $\mathbf{y}$ being at A and B, respectively, for a given $\mathbf{x}$. This can also be seen geometrically as in Fig. 4(b) which shows the bounds on $L_2(\mathbf{x}, \mathbf{y})$ given $L_1(\mathbf{x}, \mathbf{y}) = d$. The lower bound $L_2'(\mathbf{x}, \mathbf{y}) = d/\sqrt{K}$ and the upper $L_2''(\mathbf{x}, \mathbf{y}) = d$ correspond to $\mathbf{y}$ being at A and B respectively for a given $\mathbf{x}$.

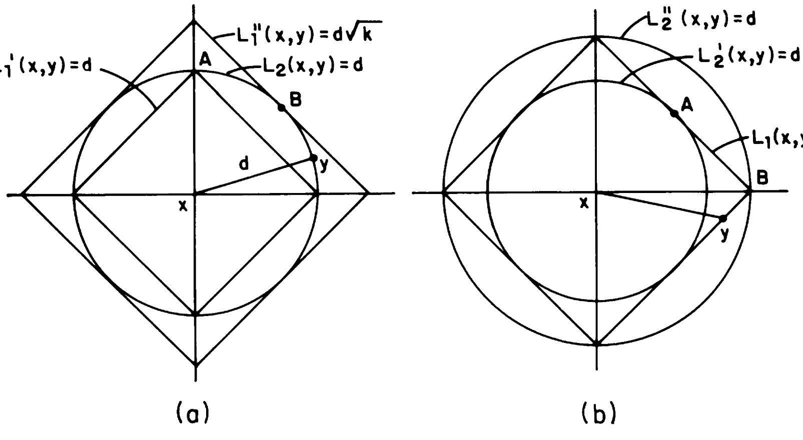

Fig. 4. Equivalence of relationship between L_1 and L_2 norms: (a) lower (L'_1) and upper (L''_1) bounds of $L_1(\mathbf{x}, \mathbf{y})$ given $\mathbf{x}$ and $L_2(\mathbf{x}, \mathbf{y}) = d$ and, (b) lower (L'_2) and upper (L''_2) bounds of $L_2(\mathbf{x}, \mathbf{y})$ given $\mathbf{x}$ and $L_1(\mathbf{x}, \mathbf{y}) = d$.

The above inequalities describing the norm equivalence can be obtained as a constrained minimization, maximization of the L_1 norm given $\mathbf{x}$ and $\{\mathbf{y} : L_2(\mathbf{x}, \mathbf{y}) = d\}$ or the constrained minimization, maximization of the L_2 norm given $\mathbf{x}$ and $\{\mathbf{y} : L_1(\mathbf{x}, \mathbf{y}) = d\}$. (Soleymani and Morgera [12] obtain the absolute error inequality (2) by induction. Though the inequality is finally shown to be satisfied for general K, we note here that, an intermediate step in Ref. [12] is incorrect: this involves proving an alternate inequality $\sqrt{(K-1)(d^2 - z_K^2)} + z_K < d\sqrt{K}$. In showing this, Ref. [12] incorrectly obtains the maximum of the function $\sqrt{(K-1)(d^2 - z_K^2)} + z_K$ to be $(2K-1)d/\sqrt{4K-3}$ at $z_K = d/\sqrt{4K-3}$; the correct maximum of the function is $d\sqrt{K}$ at $z_K = d/\sqrt{K}$.)

The use of absolute error inequality in elimination can be seen in Fig. 4(a) as it implies that points lying outside the surface $L''_1(\mathbf{x}, \mathbf{y}) = d\sqrt{K}$ cannot lie within the $L_2(\mathbf{x}, \mathbf{y}) = d$ surface and hence cannot be nearer to $\mathbf{x}$ than $\mathbf{y}$. The geometric constraint provided by the absolute error inequality around the current-nearest-neighbor ball is thus the isometric surface $L''_1(\mathbf{x}, \mathbf{y}) = d\sqrt{K}$. This is essentially a simplex polytope, which can be viewed as formed by 2^K hyperplanes each given by $\{\mathbf{y}: \sum_{k=1}^{K} |y_k| = d\sqrt{K}\}$ with intercepts at $d\sqrt{K}$ on the coordinate axes in each quadrant of the K-dimensional space with the origin translated to $\mathbf{x}$.

Note that the above algorithm essentially obtains a subset of candidate codevectors $\mathbf{C}' = \{\mathbf{c}_j : L_1(\mathbf{x}, \mathbf{c}_j) \leqslant d_{cnn}\sqrt{K}$ and $L_\infty(\mathbf{x}, \mathbf{c}_j) \leqslant d_{cnn}\}$ by employing the upper bounds of both L_1, and L_∞ distances corresponding to the initial codevector $\mathbf{c}_i$ obtained by the L_∞ based approximation in Step 1 ($\mathbf{c}_i = \arg\min_{\mathbf{c}_j \in \mathbf{C}} L_\infty(\mathbf{x}, \mathbf{c}_j)$; $d_{cnn} = d(\mathbf{x}, \mathbf{c}_i)$). The codevectors in $\mathbf{C}'$ are potential updates for the current-nearest-neighbor and have their $L_2(\mathbf{x}, \mathbf{c}_j)$ distances bound by $[L_\infty(\mathbf{x}, \mathbf{c}_i), d_{cnn}\sqrt{K}]$ as can be seen from Figs. 2(b) and 4(b), corresponding to the initial codevector $\mathbf{c}_i$ obtained by L_∞ approximation and the $L''_1(\mathbf{x}, \mathbf{c}_i)$ bound for $L_2 = (\mathbf{x}, \mathbf{c}_i) = d_{cnn}$. The algorithm selects candidate codevectors from $\mathbf{C}'$ by L_∞ ordering and performs elimination using both hypercube and absoule error inequality constraints.

2.4.1. Comparison of L_∞ and L_1 elimination constraints

The relative elimination efficiencies of the L_∞ and L_1 surfaces in providing space constraint for search can be seen by comparing their respective volumes for an unit hyperspheres, i.e., corresponding to an unit current-nearest-neighbor ball radius $L_2(\mathbf{x}, \mathbf{y}) = 1$. The volume $V(L_\infty)$ enclosed by the L_∞ surface (hypercube) is 2^K and the volume $V(L_1)$ enclosed by the L_1 surface can be shown to be $2^K(\sqrt{K})^K/K!$. It can be seen in Fig. 5, that use of both L_∞ and L_1 surfaces in providing elimination results in an improved elimination efficiency as the intersection volume (IJKLMNOP) approximates the inscribed hypersphere more closely. The volume $V(L_\infty \cap L_1)$ enclosed by the intersection of the L_∞ and L_1 surfaces can be shown to be $2^K((\sqrt{K})^K - K(\sqrt{K}-1)^K)/K!$. Fig. 6(a), shows the volumes $V(L_\infty)$, $V(L_1)$ and $V(L_\infty \cap L_1)$ for $K = 1$–10. From this, the L_1 surface can be seen to offer far superior space constraint in comparison to the L_∞ surface. The intersection volume of both L_∞ and L_1 can be seen to be only marginally better than L_1 constraint alone. This can be seen in Fig. 6(b), which shows the volumes $V(L_1)$ and $V(L_\infty \cap L_1)$ normalized to the corresponding $V(L_\infty)$ volume, thereby giving a direct comparison of the relative efficiency of L_1 and $(L_\infty \cap L_1)$ with respect to the L_∞ surface in terms of the volume ratios. This clearly reveals that elimination by L_1 is more efficient than the hypercube L_∞ based elimination, with the efficiency increasing with dimension.

The important consideration in the L_1-based elimination is the computational cost involved in determining if a codevector lies inside a given L_1 surface. As seen earlier, this will amount to computing the L_1 norm between the codevector and the test vector $\beta_i = \sum_{k=1}^{K}|x_k - c_{ik}|$ and checking if $\beta_i > d_{cnn}\sqrt{K}$. While the hypercube geometry permits realization of the L_∞ based elimination using binary search with ordered coordinates and intrinsically simple elimination checks, L_1-based elimination cannot be realized by similar alternate schemes. As a result, one of the direct ways of realizing it is as a part of the distance computation, as done in Ref. [12] with the use of an additional loop which checks the L_1 norm value in a partial distance form to eliminate a codevector before computing its squared error distance.

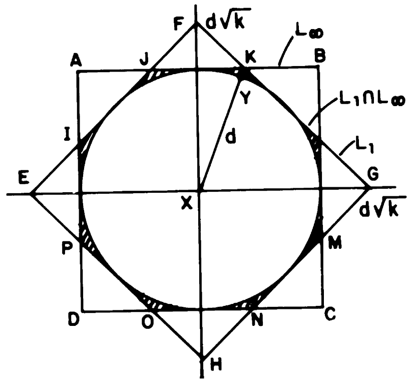

Fig. 5. Geometric constraint for elimination by L_∞ (ABCD), L_1 (EFGH) and $L_\infty \cap L_1$ (IJKLMNOP) approximation surfaces of current-nearest-neighbor ball.

2.4.2. *Approximation–elimination with L_1 constraint*

Alternately, consider the following approximation–elimination search algorithm based on using L_1 values. Given a test vector $\mathbf{x}$, the L_1 approximation values $\beta_i = L_1(\mathbf{x}, \mathbf{c}_i) = \sum_{k=1}^{K}|x_k - c_{ik}|$ for $i = 1, \dots, N$, are computed. Approximation-elimination search using the L_1 approximation can be performed as

```
$d_{cnn} = \infty$ (a very large number)
Do until codebook $\mathbf{C}$ is empty
    $\mathbf{c}_i = \arg\min_{\mathbf{c}_l \in \mathbf{C}} \beta_l$; $\mathbf{C} = \mathbf{C} - \mathbf{c}_i$
    $d = d(\mathbf{x}, \mathbf{c}_i)$
    **if** $d < d_{cnn}$ **then**
        $j = i$; $d_{cnn} = d$
        $\mathbf{C} = \mathbf{C} - \{\mathbf{c}_m : \beta_m \geq d_{cnn}\sqrt{K}\}$
    **endif**
enddo
```

Here, it can be noted that the $\beta_i = \sum_{k=1}^{K}|x_k - c_{ik}|$ values are used for approximation instead of the L_∞ values as in Refs. [10,12]. This approximation by L_1 norm is illustrated in Fig. 7. Here, the codevector $\mathbf{c}_4$ is the nearest neighbor of the test vector under L_1 norm. The L_2 nearest neighbor is codevector $\mathbf{c}_3$ which is closer to $\mathbf{x}$ than $\mathbf{c}_4$. (This can be understood based on the equivalence relation shown in Fig. 4.) The L_2 nearest neighbor is determined in the subsequent approximation–elimination steps using the L_1 elimination corresponding to $L_2(\mathbf{x}, \mathbf{c}_4)$.

The above algorithm essentially obtains a subset of candidate codevectors $\mathbf{C}' = \{\mathbf{c}_j : L_1(\mathbf{x}, \mathbf{c}_j) \leq d_{cnn}\sqrt{K}\}$ by employing the upper bound of L_1 distances corresponding

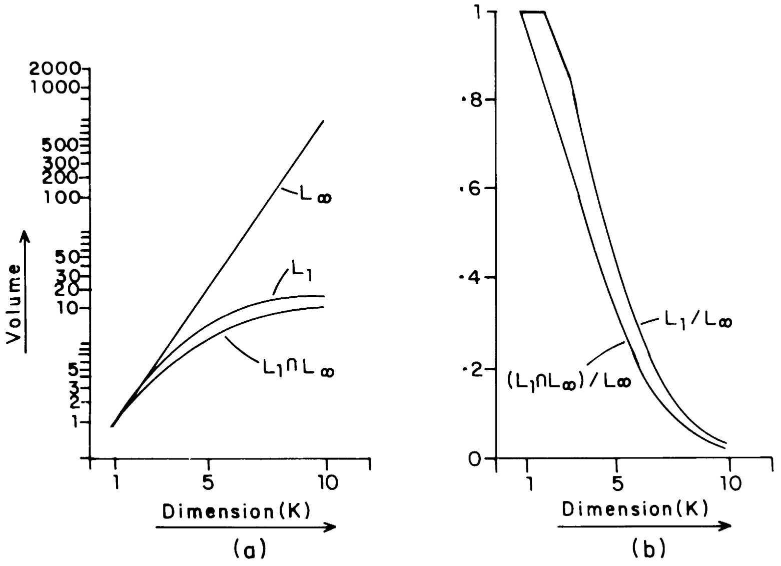

Fig. 6. Volume of L_∞, L_1 and $L_\infty \cap L_1$ surfaces inscribing unit hypersphere ($L_2 = 1$); (a) volume enclosed by L_∞, L_1 and $L_\infty \cap L_1$ surfaces as function of dimension K; (b) volume ratios L_1/L_∞ and $(L_\infty \cap L_1)/L_\infty$ as function of dimension K.

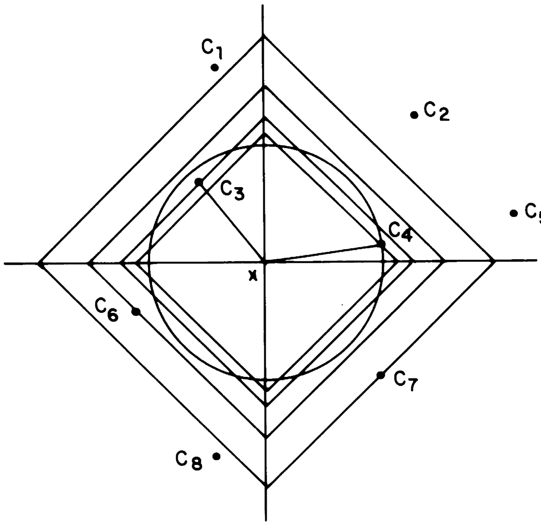

Fig. 7. Approximation by $L_1(\mathbf{x}, \mathbf{c}_i)$; $\mathbf{c}_4 = \text{argmin}_{\mathbf{c}_i \in \mathbf{C}} L_1(\mathbf{x}, \mathbf{c}_i)$; $\mathbf{c}_3 = \text{argmin}_{\mathbf{c}_i \in \mathbf{C}} L_2(\mathbf{x}, \mathbf{c}_i)$.

to the initial codevector $\mathbf{c}_i$ obtained by the L_1 based approximation in Step 1 ($\mathbf{c}_i = \arg\min_{\mathbf{c}_j \in \mathbf{C}}: L_\infty(\mathbf{x}, \mathbf{c}_j)$; $d_{cnn} = d(\mathbf{x}, \mathbf{c}_i)$). The codevectors in $\mathbf{C}'$ are potential updates for the current-nearest-neighbor and have their $L_2(\mathbf{x}, \mathbf{c}_j)$ distances bound by $[L_1(\mathbf{x}, \mathbf{c}_i)/\sqrt{K}, d_{cnn}\sqrt{K}]$ as can be seen from Fig. 4(b), corresponding to the initial codevector $\mathbf{c}_i$ obtained by L_1 approximation and the $L_1''(\mathbf{x}, \mathbf{c}_i)$ bound given $L_2(\mathbf{x}, \mathbf{c}_i) = d_{cnn}$. The algorithm selects candidate codevectors from $\mathbf{C}'$ by L_1 ordering and performs elimination using the $\beta_i = L_1(\mathbf{x}, \mathbf{c}_i)$ ordering.

The precomputed β_i values (used for approximation) are also used for elimination by L_1 in subsequent elimination steps for each current-nearest-neighbor update. This will have an intrinsically higher approximation–elimination efficiency than the elimination carried out during the distance computation step within a full-search as in Ref. [12] for the following reasons: (i) L_1-based approximation efficiency is higher than L_∞-based approximation and the search starts with an initial codevector (by L_1 approximation) closer to the test vector than that obtained by L_∞ approximation, (ii) this in turn results in a faster codebook size reduction during the subsequent elimination steps, (iii) the elimination consists only of scalar comparisons of the form $\beta_i > d_{cnn}\sqrt{K}$, which can be realized by binary search on an ordered list of the precomputed β_i.

It should also be noted that the procedure in Ref. [12] does not use precomputed L_∞ values for repeated approximation–elimination subsequent to the first approximation–elimination and hence does not represent a typical 'approximation–elimination' search as given above. Use of the L_∞ norm in the place of L_1 in the above procedure makes the search similar to the minmax method [10] and gives a direct comparison of the basic L_∞ and L_1 approximation–elimination efficiencies. Thus, the $L_1(L_\infty)$-based search procedures in the above form essentially consists of using $L_1(L_\infty)$ values for approximation and elimination and provide a comparison of the intrinsic $L_1(L_\infty)$ approximation–elimination efficiency.

Table 1
Comparison of L_∞ and L_1 approximation–elimination efficiency

	SNR_{inn}	%CER	$\overline{nc_i}$	$\widehat{nc_i}$	$((nc)_i)_\sigma$	$\overline{nc}$	$\widehat{nc_i}$
L_∞	12.0	17.9	8.8	395	19.5	6.1	159
L_1	12.3	9.1	2.2	117	4.1	1.8	49

Dimension $K = 8$; Codebook size $N = 1024$ Data: 50000 vectors of speech waveform ($\text{SNR}_{fs} = 12.5$).

2.5. Simulation results

In Table 1, we compare the approximation–elimination efficiency of the above L_∞ and L_1-based search in the context of vector quantization of speech waveform for dimension $K = 8$ and codebook size $N = 1024$ using 5000 vectors. If $\mathbf{c}_i$ is the *first* candidate codevector selected by the approximation criterion from the full codebook, the approximation efficiency can be characterized using measures which quantify the closeness of $\mathbf{c}_i$ to the test vector (or the actual nearest neighbor) when obtained over a large test vector sequence. The measures used here are: (i) SNR_{inn}: the SNR in using the 'initial' codevector as the actual nearest-neighbor codevector, (ii) %CER: the percent 'classification error rate' which is the percentage of times the initial codevector is *not* the actual nearest neighbor. The ideal limits of these measures are their values corresponding to the actual nearest-neighbor codevector; these limits are: SNR_{fs} for SNR_{inn}, i.e., the SNR corresponding to the actual nearest neighbor determined by full-search and 0 for %CER. In addition to this, the combined approximation–elimination efficiency is characterized by measuring the (average, maximum) number of codevectors $(\overline{nc_i}, \widehat{nc_i})$, retained after elimination with the current nearest-neighbor ball corresponding to the initial candidate codevector selected by the approximation criterion; $((nc)_i)_\sigma$ is the standard deviation of this measure. $(\overline{nc}, \widehat{nc})$ is the (average, maximum) number of distances computed in the overall search, which is the actual complexity of the search.

From this table, it can be seen that L_1 has a significantly higher approximation efficiency than L_∞ with the SNR_{inn}, and %CER values much closer to their respective actual nearest-neighbor limits. The combined approximation–elimination efficiency of L_1 can be seen to be much higher than L_∞ by the smaller $(\overline{nc_i}, \widehat{nc_i})$ values for L_1, particularly with the maximum number of codevectors $\widehat{nc}_i$ retained after the first elimination being significantly lower than for L_∞. The large difference in the standard deviation $((nc)_i)_\sigma$ also indicates the relatively poorer efficiency of L_∞ in comparison to L_1. The same can be observed with respect to the overall approximation-elimination efficiency given in terms of $(\overline{nc}, \widehat{nc})$, particularly with the worst-case complexity $\widehat{nc}$ being very low for search based on L_1 in comparison to L_∞ approximation-elimination.

In Fig. 8, we show the performance of L_∞-based search for dimensions $K = 2,4,6$ and 8 and codebook size $N = 1024$ in terms of $(\overline{nc_i}, \widehat{nc}_i)$-the (average, maximum) number of codevectors retained after the first elimination step (i.e., with the current nearest-neighbor ball corresponding to the initial candidate codevector) and $(\overline{nc}, \widehat{nc})$—the (average, maximum) number of distances computed in the overall search, which is the actual complexity of the search. $(\overline{nc_i}, \widehat{nc}_i)$ represents the complexity of search if a full-search is carried out after the first elimination step and $(\overline{nc}, \widehat{nc})$ is the complexity under approximation–elimination search with the codebook size reducing progressively with each elimination step. Here, the main points to be noted are: (i) the initial complexity of $(\overline{nc_i}, \widehat{nc}_i)$ is reduced significantly to $(\overline{nc}, \widehat{nc})$ by the approximation–elimination search for both L_∞ and L_1 search consistently for all dimensions, (ii) while the average complexity of both L_∞ and L_1 search is practically the same, L_∞-based search has a significantly higher worst-case complexity than L_1-based search. Moreover, the worst-case complexity of L_∞-based search increases more significantly with increasing dimension than L_1 based search. The increasing difference between L_1 and L_∞ in the worst-case initial codebook size $\widehat{nc}_i$ can be particularly noted. This trend conforms with the general volume relationship shown in Fig. 6.

2.6. Discussion

The main shortcoming in the above search procedures is the high overhead complexity in the L_∞- or L_1-based approximation. The initial approximation step is essentially a full-search using the L_∞ or L_1 distance. The cost incurred in the initial L_1 value (β_i) computation and approximation is $[0,\ NK + N(K-1), (N-1)]$ macs (multiplications, additions, comparisons) per vector. For the L_∞-based approximation, this initial approximation cost is $[0, NK, (N-1) + N(K-1)]$ macs per vector. In comparison to the L_2 distance-based full-search costs, this initial approximation step itself has an additional overhead cost of $N(K-1)$ comparisons per vector for L_∞-based search and $N(2K-1)$ additions/subtractions per vector for L_1 approximation.

The other overhead costs in the above search consists of the two main parts:

(i) *Initial elimination cost*: The cost incurred in the first elimination step after the computation of the distance with the initial codevector and the test vector. When carried out by direct comparisons in an unordered list, this has a cost of N scalar comparisons for both L_1- and L_∞-based search.
(ii) *Subsequent approximation–elimination costs*: This mainly depend on the size of codebook after the first elimination step. As seen in Table 1 $\widehat{nc}_i$, the average number of codevectors retained after the first

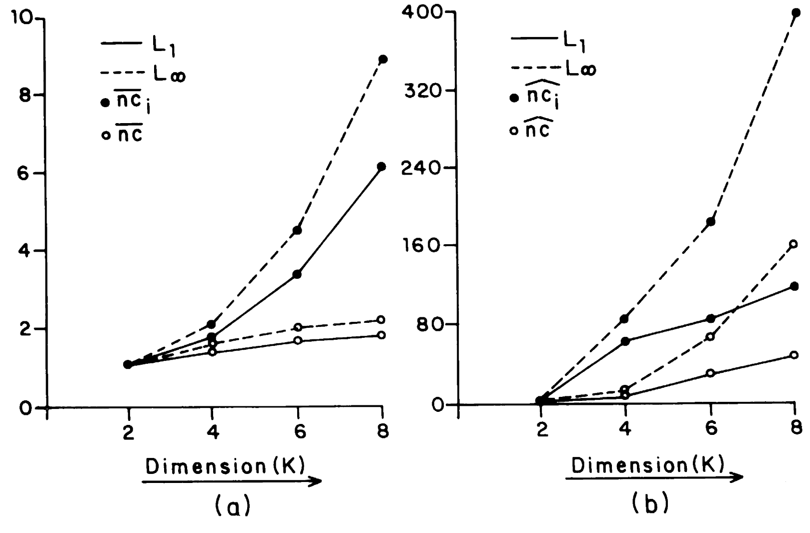

Fig. 8. Performance of L_∞- and L_1-based search for dimensions $K = 2, 4, 6$ and 8 and codebook size $N = 1024$; (a) average complexity, (b) worst-case complexity.

elimination is very small and hence the average overhead costs in subsequent approximation–elimination is practically negligible. The worst-case costs can however be high as this is proportional to $\widehat{nc}_i$. Here, L_1 based search will incur lower overhead cost in subsequent search due to the small $\widehat{nc}_i$ resulting from its higher elimination efficiency.

The main complexity and overhead costs involved in the L_1-, L_∞-based search for dimension $K = 8$ and codebook size $N = 1024$ in terms of number of macs (multiplications, additions, comparisons) per sample is as follows: The main complexity (macs) per sample, corresponding to the number of codevector distances nc is $[nc, nc(2K-1)/K, (nc-1)/K]$. The average and worst-case macs are obtained using $nc = \overline{nc}$ and $nc = \widehat{nc}$. Using these values from Table 1, the (average, worst-case) complexity of L_∞ search is [(6.1, 159), (11, 286), (0.63, 19.7)] and of L_1 search is [(1.8, 49), (8.2, 88), (0.1, 6)]. The overhead complexity obtained using the different overhead costs described above is [0, 1024, 1152] for L_∞ search and [0, 1920, 256] for L_1 search. The full-search macs are [1024, 1920, 127.9]. From this, it can be seen that in terms of main complexity, while the average complexities of L_1- and L_∞-based search are comparable, L_1-based search offers a much lower worst-case complexity; here this can be seen to be about a third of the complexity of L_∞ based search. In terms of the overhead complexity, L_1-based search has higher cost of additions and subtractions than L_∞ search, but, significantly lower comparison cost than L_∞ search. In practical applications where additions, subtractions and comparisons, have same computational cost and are less expensive than multiplications, L_1-based search will offer better complexity reduction than L_∞-based search.

Thus, the search complexity reduction for both L_1- and L_∞-based search here is mainly with respect to reducing the NK multiplications per test vector to a low value. Though multiplication is usually considered the dominant operation in actual search, in hardware realizations where all the three operations (multiplications, additions and comparisons) have equal weightage, this algorithm will not offer any significant savings due to the high overhead complexity alone, despite the excellent reduction in the main complexity of number of distances computed.

The high overhead computational complexity presents the main drawback in the L_1 and L_∞-based approximation–elimination search. This motivates the need to find alternate approximation procedures which will incur lower approximation cost than the procedures considered above. The partial distance realization (as done in Ref. [12]) for the first L_∞ approximation is one such solution which can also be applied for L_1-based approximation. However, this results in the L_1 values of several codevectors being lower than the actual values and will reduce the efficiency of elimination using the L_1 values. For the L_∞ (hypercube)-based search, an alternate and computationally more attractive approximation–elimination procedure has been proposed and studied in detail in Refs. [52,44].

3. Conclusions

In this paper, we have considered algorithms based on elimination rules which use direct properties of the distance measure such as the partial distance elimination, hypercube elimination, elimination based on the absolute-error-inequality approximation of the Euclidean distance and the triangle-inequality-based elimination, in an ‘approximation–elimination’ search framework. Here, in addition to using efficient elimination rules to reject codevectors without distance computation, explicit importance is given to the sequence in which the codevectors are examined in the search as determined approximately by their relative closeness to the test vector. The algorithms proposed and studied in this paper under this framework are based on elimination rules derived from approximations of the Euclidean distance, namely, hypercube elimination, and the elimination based on the absolute-error-inequality approximation. In this paper, previous algorithms based on these elimination rules are reviewed in the context of approximation–elimination search. The main contribution in this paper is a comparative study of different approximation schemes and elimination constraints with reference to their approximation–elimination efficiency. The paper is particularly oriented towards providing additional geometric insight to the main approximation–elimination schemes in an unified framework and empirical studies characterizing the approximation–elimination efficiency of the different approximation–elimination search in detail.

Algorithms based on the L_∞, L_1 approximation of the L_2 norm have been considered with reference to the earlier work based on the hypercube constraint of the search space [9,11], the L_∞-based ‘minmax’ algorithm [10] and its subsequent improvement using the absolute-error-inequality-based elimination [12]. These algorithms are viewed under the approximation–elimination framework and geometric details, not brought out originally in these algorithms, are provided for additional insight. The absolute error inequality is noted to be the general equivalence of the L_1 and L_2 norms and the L_∞-, L_1-based elimination criterion are geometrically seen as based on the constrained minimization and maximization of the L_∞ or L_1 norm given L_2 norm. Elimination based on the L_1 constraint has been shown to be more efficient than the L_∞ constraint based on geometric interpretation of the norm equivalence relationship and the volume ratios of the L_1 surface and the L_∞ (hypercube) surfaces for a given L_2 ball. The relative efficiencies of these eliminations have been characterized and studied with respect to dimensionality in the context of vector quantization of speech waveforms.

4. Summary

In this paper, we provide an overview of fast nearest-neighbor search algorithms based on an 'approximation–elimination' framework under a class of elimination rules derived from approximations of Euclidean distance. Here the search consists of successive approximation to the actual nearest neighbor using repeated application of two steps: (i) approximation: selecting a candidate codevector for testing as a possible successor to the current-nearest-neighbor and, (ii) elimination: rejecting codevectors which cannot be nearer to the given test vector than the current-nearest-neighbor using elimination rules without having to compute the distance from the test vector. The role of an explicit approximation step is to select codevectors which are as close as possible to the test vector using computationally inexpensive criterion and serves as an efficient alternative to choosing codevectors in random or in a prespecified sequence. Elimination involves the approximation of the current-nearest-neighbor ball by geometric constraints which are simple to compute so as to allow easy elimination of codevectors which do not lie inside the current nearest-neighbor ball. Previous algorithms based on elimination rules, namely, partial distance elimination, hypercube elimination and absolute-error-inequality elimination, are reviewed in the context of approximation–elimination search. The main emphasis in this paper is a comparative study of these elimination constraints with reference to their approximation–elimination efficiency set within different approximation schemes.

References

[1] A. Gersho, V. Cuperman, Vector quantization: a pattern matching technique for speech coding, IEEE Commun. Mag. 21 (1983) 15–21.
[2] R.M. Gray, Vector quantization, IEEE ASSP Mag. 1 (1984) 4–29.
[3] J. Makhoul, S. Roucos, H. Gish, Vector quantization in speech coding, Proc. IEEE 73 (1985) 1555–1588.
[4] H. Abut, Vector Quantization, IEEE, Piscataway, NJ, 1990.
[5] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer, Boston, 1992.
[6] W.B. Kleijn, K.K. Paliwal, Speech Coding and Synthesis, Elsevier, Netherlands, 1995.
[7] M. Barlaud et al., Special issue on vector quantization, IEEE Trans. Image Process. 5 (1996) 197–404.
[8] J.H. Friedman, F. Baskett, L.J. Shustek, An algorithm for finding nearest neighbours, IEEE Trans. Comput. 24 (1975) 1000–1006.
[9] T.P. Yunck, A technique to identify nearest-neighbours, IEEE Trans. System Man Cybernet. SMC-6 (1976) 678–683.
[10] D.Y. Cheng, A. Gersho, B. Ramamurthi, Y. Shoham, Fast search algorithms for vector quantization and pattern matching, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 1, 1984, pp. 9.11.1–9.11.4.
[11] M.R. Soleymani, S.D. Morgera, A high speed search algorithm for vector quantization, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 1987, pp. 1946–1948.
[12] M.R. Soleymani, S.D. Morgera, An efficient nearest-neighbour search method, IEEE Trans. Comput. COM-35 (1987) 677–679.
[13] E. Vidal, An algorithm for finding nearest neighbours in (approximately) constant average time complexity, Inform. Process. Lett. 4 (1986) 145–157.
[14] R.C.T. Lee, Y.H. Chin, S.C. Chang, Application of principal component analysis to multi-key searching, IEEE Trans. Software Engng SE-2 (1976) 185–193.
[15] C.H. Papadimitriou, J.L. Bentley, A worst-case analysis of nearest-neighbor searching by projection, in: Automata Languages and Programming: Lecture Notes in Computer Science, Vol. 85, Springer, Berlin, 1980, pp. 470–482.
[16] K. Weidong, H. Zheng, Fast search algorithms for vector quantization, Proceedings of the International Conference on Pattern Recognition, 1986, pp. 1007–1009.
[17] S. Adlersberg, V. Cuperman, Transform domain vector quantization for speech signals, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 1987, pp. 1938–1941.
[18] J.S. Koh, J.K. Kim, Fast sliding search algorithm for vector quantisation in image coding, Electron. Lett. 24 (1988) 1082–1083.
[19] J. Bryant, A fast classifier for image data, Pattern Recognition 22 (1989) 45–48.
[20] S.H. Huang, S.H. Chen, Fast encoding algorithm for VQ-based image coding, Electron. Lett. 26 (1990) 1618–1619.
[21] L. Guan, M. Kamel, Equal-average hyperplane partitioning method for vector quantization of image data, Pattern Recognition Lett. 13 (1992) 693–699.
[22] S.W. Ra, J.K. Kim, Fast weight-ordered search algorithm for image vector quantization, Electron. Lett. 27 (1991) 2081–2083.
[23] J. Ngwa-Ndifor, T. Ellis, Predictive partial search algorithm for vector quantization, Electron. Lett. 27 (1991) 1722–1723.
[24] A. Nyeck, H. Mokhtari, A. Tosser-Roussey, An improved fast adaptive search algorithm for vector quantization by progressive codebook arrangement, Pattern Recognition 25 (1992) 799–802.
[25] A.P. Wilton, G.F. Carpenter, Fast search methods for vector lookup in vector quantisers, Electron. Lett. 28 (1992) 2311–2312.
[26] G. Poggi, Fast algorithm for full-search VQ encoding, Electron. Lett. 29 (1993) 1141–1142.
[27] C.D. Bei, R.M. Gray, An improvement of the minimum distortion encoding algorithm for vector quantization, IEEE Trans. Commun. COM-33 (1985) 1132–1133.
[28] K.K. Paliwal, V. Ramasubramanian, Effect of ordering the codebook on the efficiency of the partial distance search algorithm for vector quantization, IEEE Trans. Commun. COM-37 (1989) 538–540.

[29] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantization design, IEEE Trans. Commun. COM-28 (1980) 84–95.

[30] F.P. Fischer, E.A. Patrick, A preprocessing algorithm for nearest neighbour decision rules, Proceedings of the National Electronics Conference, Vol. 26, 1970, pp. 481–485.

[31] W.A. Burkhard, R.M. Keller, Some approaches to best-match file searching, Commun. ACM 16 (1973) 230–236.

[32] C.J. Van Rijsbergen, The best-match problem in document retrieval, Commun. ACM 17 (1974) 648–649.

[33] K. Fukanaga, P.M. Narendra, A branch and bound algorithm for computing *k*-nearest neighbours, IEEE Trans. Comput. 24 (1975) 750–753.

[34] B.A. Shapiro, The choice of reference points in best-match file searching, Commun. ACM 20 (1977) 339–343.

[35] I.K. Sethi, A fast algorithm for recognizing nearest neighbours, IEEE Trans. System Man Cybernet. SMC-11 (1981) 245–249.

[36] I. Kalantari, G. McDonald, A data structure and an algorithm for the nearest point problem, IEEE Trans. Software Engng. SE-9 (1983) 631–634.

[37] B. Kamgar-Paris, L.N. Kanal, An improved branch and bound algorithm for computing *k*-nearest neighbours, Pattern Recognition Lett. 3 (1985) 7–12.

[38] E. Vidal, H.M. Rulot, F. Casacuberta, J.M. Benedi, On the use of a metric-space search algorithm (AESA) for fast DTW-based recognition of isolated words, IEEE Trans. Acoustics, Speech Signal Process. 36 (1988) 651–660.

[39] H. Niemann, R. Goppert, An efficient branch-and-bound nearest-neighbour classifier, Pattern Recognition Lett. 7 (1988) 67–72.

[40] S.H. Chen, J.S. Pan, Fast search algorithm for VQ-based recognition of isolated words, IEE Proceedings, Vol. 136, 1989, pp. 391–396.

[41] V. Ramasubramanian, K.K. Paliwal, An efficient approximation-elimination algorithm for fast nearest-neighbour search based on a spherical distance coordinate formulation, in: L. Torres-Urgell, M.A. Lagunas-Hernandez (Eds.), Signal Processing V: Theories and Applications (Proceedings EUSIPCO-1990, Barcelona, Spain), North-Holland, Amsterdam, 1990, pp. 1323–1326.

[42] D. Shasha, Tsong-Li Wang, New techniques for best match retrieval, ACM Trans. Inform. Systems 8 (1990) 140–158.

[43] M.T. Orchard, A fast nearest-neighbor search algorithm, Proceedings of the IEEE International Conference Acoustics, Speech, and Signal Processing, Toronto, Canada, 1991, pp. 2297–2300.

[44] V. Ramasubramanian, K.K. Paliwal, An efficient approximation-elimination algorithm for fast nearest-neighbor search, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, San Francisco, California, 1992, pp. I-89–I92.

[45] M.L. Mico, J. Oncina, E. Vidal, An algorithm for finding nearest neighbors in constant average time with linear space complexity, Proceedings International Conference on Pattern Recognition, Hague, 1992, pp. 557–560.

[46] M.L. Mico, J. Oncina, E. Vidal, A new version and improvements of the nearest-neighbor approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements, Pattern Recognition Lett. 15 (1994) 9–18.

[47] E. Vidal, New formulation and improvements of the nearest neighbor approximating and eliminating search algorithm, Pattern Recognition Lett. 15 (1994) 1–8.

[48] M.L. Mico, Nearest-neighbor search algorithms in metric spaces, Ph.D. Thesis, Universidad Politecnica de Valencia, 1996.

[49] M.L. Mico, J. Oncina, E. Vidal, A fast branch and bound nearest neighbor classifier in metric spaces, Pattern Recognition Lett. 17 (1996).

[50] E. Vidal, F. Casacuberta, H. Rulot, Is the DTW 'distance' really a metric?—an algorithm reducing the number of DTW comparisons in isolated word recognition, Speech Commun. 4 (1985) 333–344.

[51] E. Vidal, F. Casacuberta, J.M. Benedi, M.J. Lloret, H. Rulot, On the verification of the triangle inequality by DTW dissimilarity measures, Speech Commun. 7 (1988) 67–79.

[52] V. Ramasubramanian, Fast algorithms for nearest-neighbour search and application to vector quantization encoding, Ph.D. Thesis, Computer Systems and Communications Group, TIFR, Bombay, India, 1991.

About the Author—V. RAMASUBRAMANIAN received his B.Sc. degree (in Applied Sciences) in 1981, B.E. degree (in Electronics and Communications) from Indian Institute of Science, Bangalore, India, in 1984. He joined the Computer Systems and Communications Group, Tata Institute of Fundamental Research, Bombay, India in 1984 as Research Scholar and obtained his Ph.D. degree (in Computer science) from University of Bombay in 1992. He was a visiting scientist at the Department of Informatic Systems Computation, University of Valencia, Spain from 1991–1992. Since 1993, he is a Fellow in Computer Systems and Communications (CSC) Group, TIFR. Presently he is an Invited Researcher at ATR Interpreting Telecommunications Research Laboratories, Kyoto, Japan for the period 1996–1997. His main research interests are in speech coding, vector quantization-based coding algorithms, complexity reduction of vector quantization encoding, fast nearest-neighbor search algorithms, self-organizing neural networks, pattern recognition in metric space, and speech recognition.

About the Author—K.K. PALIWAL received the B.S. degree from Agra University, India, in 1969, the M.S. degree from Aligarh University, India, in 1971 and the Ph.D. degree from Bombay University, India, in 1978. Since 1993, he has been a Professor (Chair, Communication/Information Engineering) at the Griffith University, Brisbane, Australia. He has worked at a number of organizations including Tata Institute of Fundamental Research, Bombay, India, Norwegian Institute of Technology, Trondheim, Norway, University of Keele, U.K., AT&T Bell Laboratories, Murray Hill, NJ, USA, and Advanced Telecommunication Research (ATR) Laboratories, Kyoto, Japan. He has co-edited two books: *Speech Coding and Synthesis* (Elsevier, 1995) and *Speech and Speaker Recognition*: *Advanced Topics* (Kluwer, 1996). He has published more than 100 papers in international journals. He is a recipient of the 1995 IEEE Signal Processing Society Senior Award. He is as Associate Editor of the IEEE Transactions on Speech and Audio Processing and IEEE Signal Processing Letters. His current research interests include speech processing, image coding and neural networks.