EVALUATION OF FAST ALGORITHMS FOR FINDING THE NEAREST NEIGHBOR

Stephane Lubiarz, Philip Lockwood
Matra Communication, Speech Processing Department
rue J.P. Timbaud - BP 26, 78392 Bois d'Arcy Cedex, France
Email stephane.lubiarz@matra-com.fr, philip.lockwood@matra-com.fr

Abstract

In speech recognition systems as well as in speech coders using vector quantization, the search for the nearest neighbor is a computationally intensive task. In this paper, we adress the problem of fast nearest neighbor search. State of the art solutions tend to approach logarithmic access time. The problem is that such performance is generally achieved at the expense of a significant increase in storage requirements. In this contribution, we compare several known approaches and propose new extensions. These new contributions allows for a significant reduction in memory requirements without impacting the performance in terms of number of distances computed and optimality of the search.

I. Introduction

The now widespread HMM approach used in speech recognisers having discrete or continuous representations of the state likelihoods, or the celp-based speech coders also widely present in standards, make extensive use of distance computation and search for nearest neighbor. The question of the implementation of these algorithms for real-time applications can become a critical issue.

Bocchieri [1] has shown that computing only k likelihoods (the k nearest neighbors in fact) does not deteriorate performance of a speech recognition task. Similarly, Lockwood [3] showed that fast search can reduce the computational load of an isolated word recognition task by a significant amount. There is an extensive litterature on fast nearest neighbor search. Three factors have to be considered to characterise the methods: the search efficiency, the storage cost overhead and the complexity of the training phase. Fukunaga & al [2] propose a Branch And Bound algorithm with stopping rules exploiting the triangular inequality properties of the Euclidean distance. This technique has been improved by Lockwood [3] by incorporating a new stopping rule, allowing a reduction in memory requirement. Sethi [4] proposed a (sub-optimal) technique based on the classification of space according to the distance to some reference points. Bakamidis et al. [5] incorporate a condition to this algorithm for suppressing the distortion between the computed Nearest Neighbor and the real Nearest Neighbor. Kim and Park [6] propose a (sub-optimal) Branch And Bound algorithm with stopping rule based on the minimal distance between a point and a cube. Cheng and Gerscho [7] propose an algorithm based on an adaptive cut of the space by a cube and a stopping rule.
This paper presents adaptations of the algorithms proposed by Lockwood [3] and Kim and Park [6], with the goal of reducing the memory requirement without losing optimality. We will compare these algorithms with the « standard » implementations on one hand, and other shemes on the other hand: Sehti algorithm revisited by Bakimidis [5] and with a fast search based on the use of Voronoi diagrams.

II. Fukunaga revisited algorithm

The Fukunaga algorithm creates a data structure on the initial set of vectors. The approach consists in decomposing the vector Euclidean space into a set of embedded hypervolumes. A tree structure is created with at each node an hypersphere. Each hypersphere is decomposed further into smaller hyperspheres. This decomposition process is pursued until a hypersphere contains a minimal number of vector points. Each node of the tree is characterised by the center of the sphere, its radius, the number of sons. Each leaf of the tree contains one or several vectors belonging to the initial codebook.
The center of the hypersphere is obtained by a k-means algorithm: the centroid of the cluster is used as representative. The stopping rules of the Branch And Bound algorithm are given herafter [2], [3]:

Let B be the distance to the current Nearest Neighbor, rp is the radius of the current sphere and d(x,Mp) is the distance between the unknown point x and the center of the current hypersphere.
Rule 1 .If $B+rp<d(x,Mp)$, then the points contained in the hypersphere cannot be Nearest Neighbor candidates.
Rule 2 .On a leaf node, If $B+d(x_i,Mp)<d(x,Mp)$, then x_i cannot be the nearest neighbor of x.
In fact these rules indicate whether there is an intersection between the hypersphere centered on x, having B as radius, and the current Hypersphere.

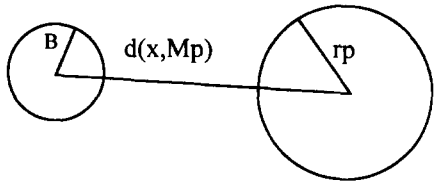


Figure 1: when rule 1 is satisfied, the algorithm forgets the node.

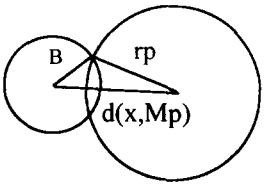


Figure 2: when rule 1 is not satisfied, the algorithm searches into the son node.

The last rule is a stopping rule as described in [3]:
let D be the distance between the current Nearest Neighbor and its Nearest Neighbor in the codebook.
Rule 3 : If 2B<D, then the current nearest neighbor is the « true » nearest neighbor.

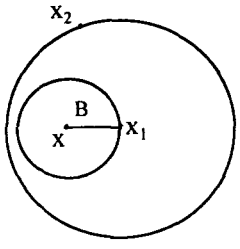


Figure 3: Illustration of rule 3: X_1 is the current nearest neighbor, X_2 is the nearest neighbor of X_1. The condition is true, X_1 is the « true » nearest neighbor.

Lockwood [3] proposes to replace the hypersphere representative (centroid) by vector belonging to the original space. The criterion used for the selection is the min-max condition: the center is the nearest neighbor to the cluster centroid and is called the min-max representation. By using this approach, we obtain a degradation in terms of search efficiency, but the memory overhead requirement has been kept low. This is particularly attractive as the dimension of the vector space increases.

The degradation of the performance is due to the fact that the overlapping between the new spheres is greater than before. As a consequence, the first stopping rule is less efficient and the third stopping rule is generally found not sufficient for compensating for the degradation.

In order to overcome this problem we introduce a new procedure for rearranging the tree. The algorithmic structure is given below:

. For every pair of hyperspheres, draw the line passing through the two centroids obtained by the k-means algorithm.

. For each hypersphere of the pair, choose the orthogonal projection, for each vector included in the sphere, which gives the greatest distance with the initial minmax of the other hypersphere. Take as center of the hypersphere the Nearest Neighbor in the dictionary of this orthogonal projection.

. If the new center is nearer to the center of one of the other hyperspheres, keep the old center. Else the nearest neighbor of the orthogonal projection is the new center of the hypersphere. By this procedure, we move the exterior hyperspheres away on the peripheral of the base. The radius of these hyperspheres tend to be greater. So the separation between hyperspheres tends to be a hyperhyperplane, which minimises the overlapping between the hyperspheres.

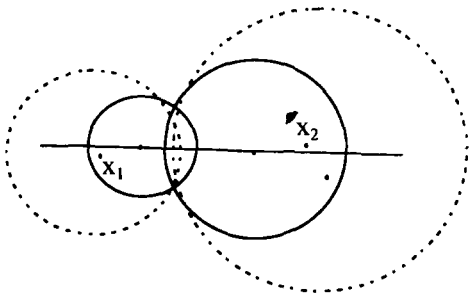


Figure 4: In bold are the spheres generated by the initial representation (C_1, C_2). In dotted lines, the volums obtained by the new representation (X_1, X_2).

III. Kim & Park revisited

This algorithm is based on the distance between a point and a cube. Imagine that the space is decomposed into cubes, as in this figure for p=2:

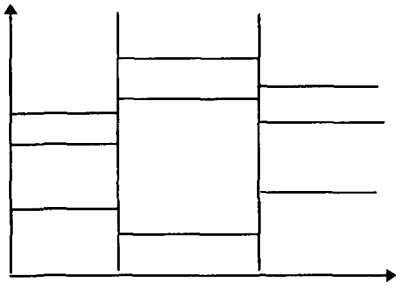
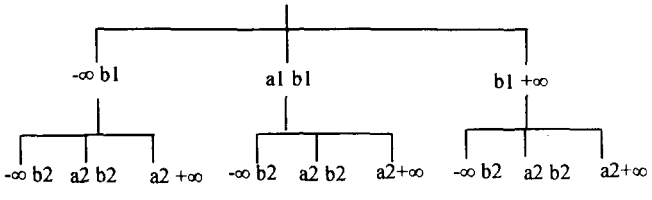


Figure 5: Decomposition of the space

We decompose the space in order to have the same number of points in each cube, and finally represent the codebook with a tree: each level of the tree corresponds to a dimension of the space, each node contains the boundaries of each cube for the dimension considered and each leaf contains the various points of the codebook.



Kim & Park use the iterative distance computation:

$$D_i^2 = \begin{cases} D_{i-1}^2 + \min\left[(x_i - a_i)^2, (x_i - b_i)^2\right] \text{for } i \in [1, p-1] \\ 0 \text{ if } i = 0 \\ D_{i-1}^2 + (x_p - a_p)^2 \text{ if } i = p \end{cases}$$

During the search in the tree, we compute the iteration of the distance for the dimension considered. In a node, if D_i>B, with B the distance of the current nearest neighbor, then none of the points present in the cube can be nearest neighbor. This is due to the fact that D_i is the minimum distance between the unknown point and the cube in dimension i. So,

if $D_i > B$, all the distances between the unknown point and the points of the codebook in the cube will be greater than B.
In fact, when we use this method, we observe that the nearest neighbor found is not all the time the « true » nearest neighbor. For example, on a codebook composed of 256 vectors in dimension 3, we obtain 18,3% of errors.

We propose the following modification for distance computation:

$$D_i^2 = \begin{cases} D_{i-1}^2 + \begin{cases} \min\left[(x_i - a_i)^2, (x_i - b_i)^2\right] \text{ if } x_i \notin [a_i, b_i] & \text{for } i \in [1, p-1] \\ 0 \text{ otherwise} \end{cases} \\ 0 \text{ if } i = 0 \\ D_{i-1}^2 + (x_p - a_p)^2 \text{ if } i = p \end{cases}$$

By using this modified distance computation, we are sure that the point found is the « true » Nearest Neighbor.
The number of hyperhyperplanes can be made variable according to the variance on each axis. If we call Nb the number of points in the codebook, Nb_i the number of hyperhyperplanes for the axis i and V_i the variance of the axis i, we pose that

$$Nb_i = \alpha P_i \text{ with } P_i = \frac{V_i}{\sum_{i=1}^{P} V_i}$$

At the same time, we want ideally only 1 point per cube:

$$\prod_{i=1}^{P} Nb_i = Nb \quad \text{so: } Nb_i = P_i \frac{\sqrt[P]{Nb}}{\left(\prod_{i=1}^{P} Pi\right)^{1/P}}$$

In fact, because the Nb_i are integers, we round off Nb_i to the lower integer, and add 1 to all Nb_i from the greatest to the least until $\prod_{i=1}^{P} Nb_i > Nb$. By this procedure, we are sure to have at least 1 point in each cube, but sometimes 2.

IV. On the use of the Voronoi diagrams

Representing the points of a codebook in a Voronoi Diagram is something leading to optimal performance. Nevertheless, such approach is difficult to implement, especially as the dimension of a vector increases (>6). Thus the Voronoi diagram principle is approached heuristically. For example. Gersho & al [7] propose a tree-like structure and derive separation functions using Voronoi hyperhyperplanes, and construct the tree by choosing which Voronoi polygon is on the right or on the left of the current hyperhyperplane. In this algorithm, we use the same representation as for Kim & Park: the space is cut by hyperhyperplanes orthogonal with each dimension. We use this representation on the Voronoi diagram, for filtering this space (fig 6): each point whose polygon is in the cube, will belong to it.

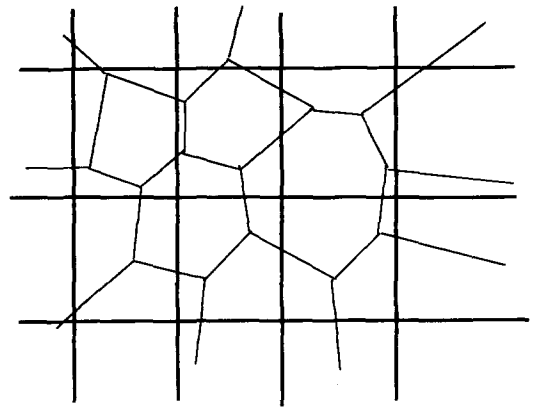


Figure 6: filtering of the Voronoi diagram

During the search, the cube in which the unknown point lies is found straightforwardly. We do a full search on the points present in the leaf node represented in the selected cube.
The critical point in this representation is the cutting of the space by hyperhyperplanes. If the cutting is not done very well, one could have a totally unbalanced tree and the method would loose its efficiency. A complete study about this topic could be found in [8].

V. Experimental results

We have compared these various algorithms on different codebooks used in speech recognition or in speech coders. We give here the results obtained with codebooks composed of LSP coefficients [9].

Number of points / Number of hyperhyperplans	128	256	516
10	4.31	7.34	10.95
20	3.1	4.31	5.6
30	2.78	3.45	4.36
40	2.61	3.21	3.8
50	2.54	2.96	3.55
60	2.49	2.81	3.39
100	2.38	2.61	3.05

table 1: number of distances computed when using Voronoi diagram with p=3.

	codebook 1 Nb=256 p=3		codebook 2 Nb=512 p=3		codebook 3 Nb=256 p=4	
Sehti	16.23	8.32	19.01	13.24	17.14	8.49
Fukunaga	58.75	3.55	74.97	4.9	56.63	3.53
Fukunaga revisited	50.12	3.2	67	3.99	51	3.47
KimPa.rev.	53.13	3.18	89.84	4.32	42.54	4.52

table 2: The first column is the Number of computed distances, the second the duration of the search.

	O	P	Q
Sehti	$o(N\log_2 N)$	2.ref.N	$o(\log_2 N)$
Sehti revisited	$o(N^2)$	(2+ref).N	$o(\log_2 N)$
Fukunaga	$o(N\log_2 N)$	C.(P+1) +N	$o(\log_2 N)$
Fukunaga revisited	$o(N^2)$	C+N	$o(\log_2 N)$
Kim&Park revisited	$o(N\log_2 N)$	$\prod_{i=1}^{P} Cpe_i + P$	$o(\log_2 N)$
Voronoi	$o(p\frac{N^2(N+1)}{2})$	$\prod_{i=1}^{P} Cpe_i(1+Moy)$	o(Moy)

table3: Comparison between the algorithms tested in terms of learning stage complexity, search complexity or memory requirements. O is the learning stage complexity, P the memory requirement and Q the search complexity. We note N the number of points in the codebook, P the space dimension, ref the number of reference points (for Sehti algorithm) C the number of nodes of the tree, Cpe_i the number of hyperhyperplanes (for Voronoi or Kim&Park algorithm) and Moy the average number of points in each cube (for Voronoi algorithm).

By analysing the results, we could see that the algorithm based on the Voronoi diagram seems to be the best in terms of number of distances computed. In theory, it is possible to generate a tree leading to the computation of only one distance. But this is at the expense of the memory requirement. That is why this algorithm is not comparable directly with the other methods. If the memory space is sufficient, this algorithm will yield the best results. But otherwise the revisited Fukunaga algorithm performs best. This algorithm offers the best compromise among the algorithms tested in terms of number of distances computed, complexity of the training and memory requirements.

The modified Sehti algorithm gives a very small number of distances to be computed. But the duration of the search is anyhow long, because the number of comparisons is large. And anyhow, when the dimension of the space grows, the number of computed distances increases so much that for a codebook of dimension 10, Fukunaga revisited becomes also better in terms of number of computed distances.

The final figures are codebook dependent, but the classification of these different algorithms will not differ from one codebook to another. Two properties of the codebook are important: one is the topology of the points, the other is the ratio between the number of vectors and their dimension. The importance of the second characteristic could be seen in figure 5: the larger the dimension of the vectors, the higher the number of vectors in the codebook needed in order to keep the ratio between the number of computed distances and the number of vectors in the dictionary constant. In fact, it seems that a quadratic relationship between p and Nb must be fulfilled in order to maintain the efficiency of the search.

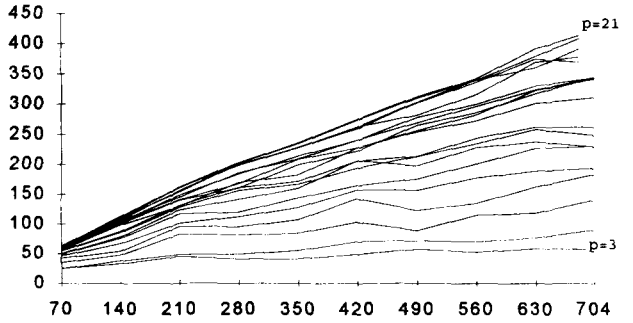


Figure 5: Number of computed distances function of Nb for th Fukunaga revisited algorithm. Each curve represents a different p. Here $3 \leq p \leq 21$.

V. Conclusion

In this paper, we have presented an evaluation of state of the art fast nearest neighbor search algorithms. New evolutions have also been proposed. One of the main conclusions are that we show here that the revisited Fukunaga scheme, offers the best compromise among the important factors characterising fast nearest neighbor search algorithms (memory requirements, number of distances computed, duration of the search). An optimal version of Kim&Park algorithm has also been proposed. Among the possible areas for improvement, one axis would be to combine « the best » among a few carefully selected schemes. The second axis is the developement of algorithms which will overpass the dimensionnality versus codebook size paradigm.

References

[1] E. Bocchieri., *"Vector quantization for the efficient computation of continuous density likelihoods"*, ICASSP 94, II. 692-695.

[2] K. Fukunaga and P.M. Narendra, *"A Branch and Bound algorithm for computing k nearest neighbors"*, IEEE trans. on computers, July 1975.

[3] P. Lockwood, *"A low cost dtw-based discrete utterance recogniser"*, EICPR-86, pp 467-469.

[4] I.K. Sethi, *" A fast algorithm for recognising nearest neighbors"*, IEEE trans. on Systems, Man an Cybernetique, vol. 11, Num 3, March 1981.

[5] S.G. Bakamidis an Y.S. Boutalis, *"A new fast algorithm to identify the nearest neighbor"*, Signal Processing VI, Theories and Applications, Volume I, pp 539-542, Aug 1992.

[6] B.S. Kim and S.B. Park, *"A fast nearest neighbor finding algorithm based on the ordered partition"*, IEEE trans. on patt. anal.,Vol 8 N°6, nov. 1986.

[7] Y. Cheng and A. Gersho, *"A fast codebook search algorithm for nearest-neighbor pattern matching"*, ICASSP 86, pp 6.14.1-6.14.4.

[8] K. Ramasubramanian and Kuldip K. Paliwal *"Fast K-Dimensional Tree for Nearest Neighbor Search with Application to vector Quantization Encoding"* IEEE trans. on sig. pro., Vol 40, N°3, March 1992.

[9] Kuldip K. Paliwal and Bishnu S. Atal *"Efficient Vector Quantization of LPC Parameters at 24 Bits/Frame"* IEEE tr. on speech and audio pro., Vol 1, N°1, January 1993.