Hierarchical Ordering for Approximate Similarity Ranking

J J Chua
P E Tischer

TECHNICAL REPORT NO: 2003/141

April 2003

School of Computer Science and Software Engineering
Monash University
Victoria 3800
Australia

Hierarchical Ordering for Approximate Similarity Ranking

Joselíto J. Chua Peter E. Tischer

School of Computer Science and Software Engineering
Monash University, Victoria 3800, Australia
Tel. +61-3-99055200 Fax. +61-3-99055146
{jjchua, pet}@mail.csse.monash.edu.au

Abstract

We propose a partial ordering that approximates a ranking of the items in a database according to their similarity to a query item. The partial ordering uses a single-link hierarchical clustering of the data items to rank them with respect to the query's closest match. The technique avoids the $O(kn)$ cost of calculating the similarity measure between the query and every item in the database. It requires only $O(n)$ space for pre-computed information. The technique can also provide a criterion for determining which items may not need to be included in the ranking. The results of our experiments suggest that the partial ordering provides a good approximation to the similarity ranking.

1 Introduction

We consider the information retrieval problem of ranking the items in a database according to their similarity to a query item. A large part of the cost is due to the calculation of the similarity measure. In order to rank the data items according to their similarity to the query, the similarity measure needs to be calculated between the query and every item in the database. For example, if the data items are k-dimensional vectors, and similarity is measured by the Euclidean distance, then ranking the items in a database of size n will cost $O(kn)$ computations. A large portion of those calculations can be avoided if we can determine which items are so far away from the query, that they need not be included in the ranking, without having to compute the similarity measure between them and the query.

One solution is to avoid the calculation of the similarity measure between the query and every item in the database by ranking the items according to their similarity to the query item's *closest match* instead. The idea is that items which are similar to the closest match are also likely to be similar to the query item. There are a number of fast nearest-neighbour (NN) search techniques that can be used for finding a query item's closest match in the database [1]. An $O(n^2)$ *proximity table* that lists the value of the similarity measure between data items can be pre-computed. The data items can be ranked with respect to the closest matching item by looking up the table, and sorting the data items according to their similarities to the closest match. The problem with this approach is that the $O(n^2)$ space required by the proximity table can be prohibitive in applications involving large databases.

The solution we propose in this paper is to cluster the data items hierarchically using a *minimal cost spanning tree (MCST)*, and to use the resulting partial ordering to rank the data items relative to the closest match. The MCST represents a *single-link hierarchical clustering* of the data [2], [3]. The clustering is also invariant under monotonic transformations of the distance function. We shall discuss the technique in Section 2. If the similarity measure is a metric, then the triangle inequality provides a criterion for determining sections of the MCST which contain items that may be considered too far away to be included in the ranking. We shall discuss this criterion in Section 3. Section 4 shows the performance of these techniques in our experiments.

2 Hierarchical Ordering

Consider an MCST where each vertex is a data item, and the weight of an edge between two vertices is the value of the (dis)similarity measure between the data items connected by that edge. There are a number of efficient algorithms for constructing and updating an MCST over the database (for example: [4], [5], and [6]). An MCST can be stored in $O(n)$ space. An important well-known property of an MCST is as follows:

Property 1 *Every edge in an MCST partitions the vertices into two clusters, such that no other pair of vertices between those clusters are closer to each other than the two vertices connected by the partitioning edge.*

Suppose the database consists of items $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, and $\mathbf{x}$ is the query item. Property 1 implies that every $\mathbf{y}_i$ is connected to the MCST through one of its closest matching items. Thus, every $\mathbf{y}_i$ is linked to a "chain" of nearest-neighbouring items in the MCST. Each chain terminates at an edge that connects two *mutual nearest neighbours*—that is, the two items are nearest-neighbours to each other. Disjoint chains are connected in the MCST by the shortest path from one of the vertices in one chain to a vertex in another chain.

We obtain a partial ordering of the data items by following the nearest-neighbour chains, starting from the query's closest match. We illustrate this partial ordering in Fig. 1 with a sample MCST from [2]. Suppose item $\mathbf{y}_6$ is the query's closest match. We follow a nearest-neighbour (NN) chain from $\mathbf{y}_6$ as follows: $\mathbf{y}_4$ is a nearest neighbour of $\mathbf{y}_6$, and $\mathbf{y}_5$ is a nearest neighbour of $\mathbf{y}_4$ (Fig. 1-a and 1-b). That chain terminates at $\mathbf{y}_5$ because $\mathbf{y}_5$ and $\mathbf{y}_4$ are mutual nearest neighbours. Since vertex $\mathbf{y}_7$ is closest to the chain $\mathbf{y}_6\mathbf{y}_4\mathbf{y}_5$, we follow the NN chain from $\mathbf{y}_7$, which is $\mathbf{y}_7\mathbf{y}_5\mathbf{y}_4$, but without going through the items $\mathbf{y}_5$ and $\mathbf{y}_4$ again (Fig. 1-c). Since the two chains are now linked, we look for another vertex closest to either chains. In this example, the closest vertex is $\mathbf{y}_2$, and so we follow the chain $\mathbf{y}_2\mathbf{y}_3$ (Fig. 1-d and 1-e), and so on. As a result, the items are ordered as follows:

$$\mathbf{y}_6 \rightarrow \mathbf{y}_4 \rightarrow \mathbf{y}_5 \rightarrow \mathbf{y}_7 \rightarrow \mathbf{y}_2 \rightarrow \mathbf{y}_3 \rightarrow \mathbf{y}_1 \rightarrow \mathbf{y}_8$$

We use the single-link hierarchical clustering represented in the MCST. The single-link clustering algorithm proposed by Gower and Ross [2] consists of an increasing series of predefined edge weight thresholds $(\delta_1, \delta_2, \ldots)$. The clusters at level δ_l are constructed by grouping

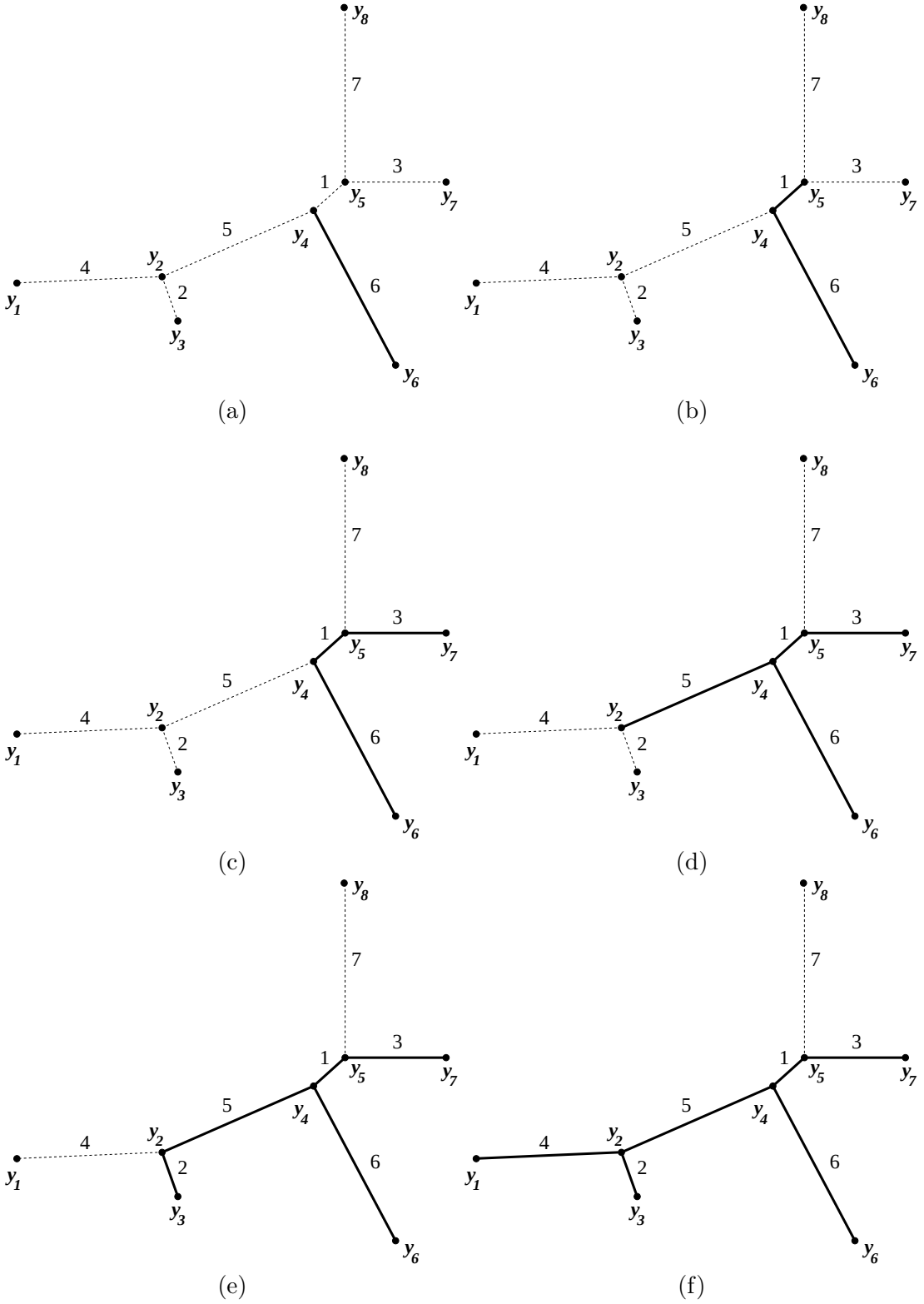

Figure 1: This figure illustrates an example of how the chains in the MCST can be followed starting from $\mathbf{y}_6$. The resulting partial ordering is given by $\mathbf{y}_6\mathbf{y}_4\mathbf{y}_5\mathbf{y}_7\mathbf{y}_2\mathbf{y}_3\mathbf{y}_1\mathbf{y}_8$.

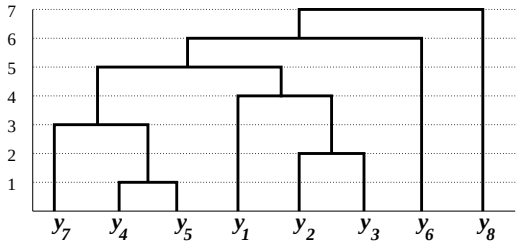

Figure 2: The *dendogram* representing the single-link hierarchical clustering for the MCST in Fig. 1. The thresholds are determined by the sorted weights of all the edges in the MCST.

together the vertices which are joined in the MCST by an edge whose weight is δ_l or less. At level δ_{l+1}, two clusters will combine into a bigger cluster only if they are joined by an edge whose weight is greater than δ_l and less than or equal to δ_{l+1}. The thresholds can be spaced uniformly within the range of the edge weights. In our implementation, we use the sorted weights of all the edges in the MCST for the thresholds. The resulting clustering information is represented by a *dendogram*. Fig. 2 shows an example of the dendogram that corresponds to the example in Fig. 1.

We implement the dendogram as a binary tree where each child node has a pointer to its parent node, as illustrated in Fig. 3. Each non-leaf node represents the MCST edge connecting two mutually exclusive subtrees of the MCST. Each subtree represents a cluster, and the leaves are the items in the cluster represented by that subtree.

The partial ordering is obtained by traversing the dendogram recursively according to the links provided by the nodes. Fig. 3 demonstrates the traversal for the example in Fig. 1. The resulting ordering is *hierarchical* in the sense that the clusters are traversed recursively from the lowest to the highest level of the hierarchy. Clusters closest to the smallest cluster containing the starting leaf are traversed first. The dendogram makes it easy to find another NN chain that can link to the current NN chain, because the link is given by the node with which the current cluster is connected to the rest of the dendogram. We obtain an approximate similarity ranking of the items by traversing the dendogram from the leaf that corresponds to the query's closest match.

3 Terminating Criterion

If the similarity measure, $d(\cdot, \cdot)$, is a metric, then the triangle inequality provides a *criterion* for pruning the ranked list of items. The triangle inequality guarantees that if $h = d(\mathbf{x}, \mathbf{y}_{\text{curr}})$ and

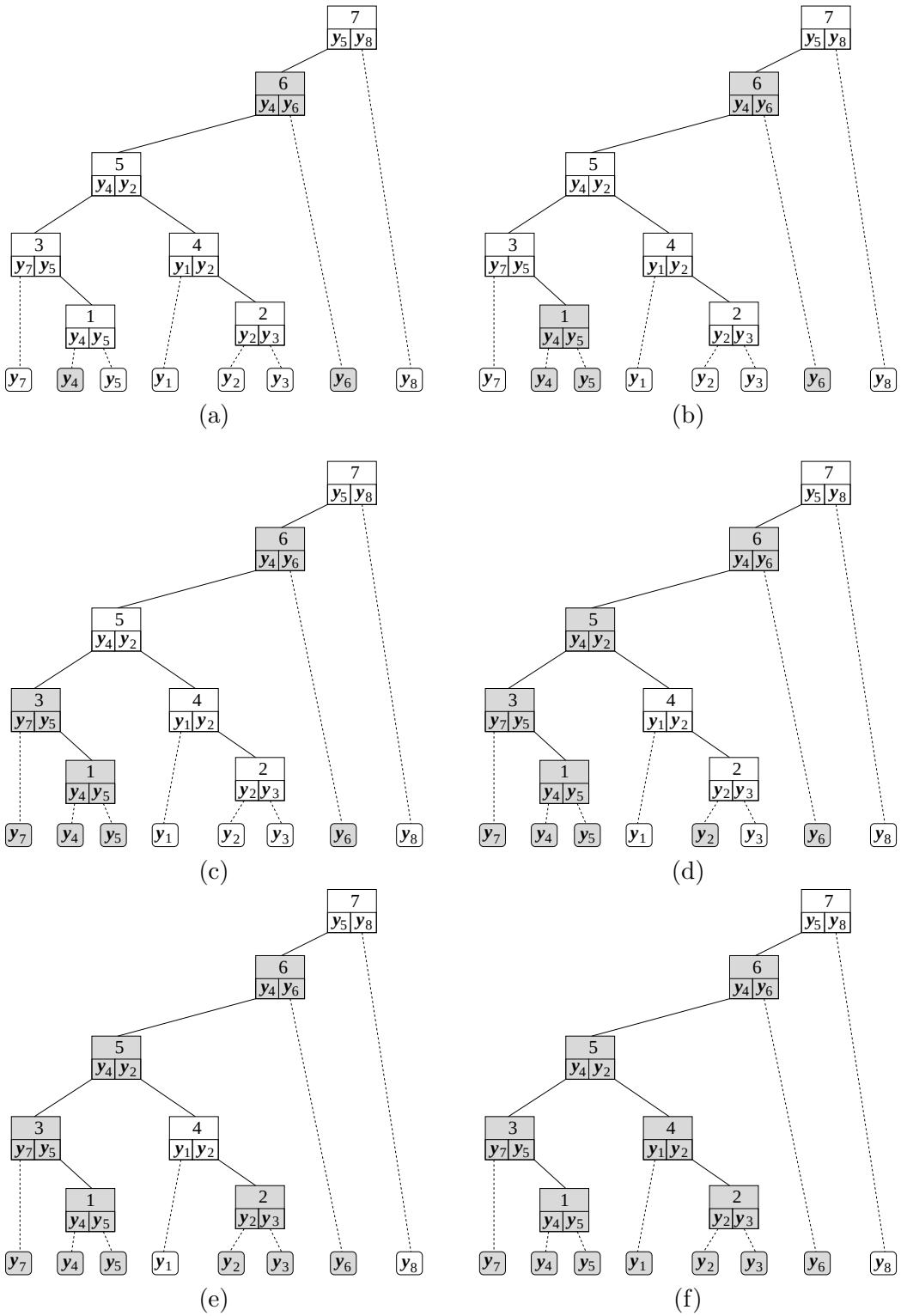

Figure 3: This figure illustrates how to traverse the binary-tree representation of the dendogram in Fig. 2. The hierarchical clusters are traversed recursively from the lowest to the highest level of the hierarchy. Starting from $\mathbf{y}_6$, the same partial ordering as that of Fig. 1 is obtained.

$d(\mathbf{y}_i, \mathbf{y}_{\text{curr}}) > 2h$, then $d(\mathbf{x}, \mathbf{y}_i) > h$ [7]. Suppose item $\mathbf{y}_j$ is followed immediately by item $\mathbf{y}_i$ in an hierarchical ordering which starts from $\mathbf{y}_{\text{curr}}$. Property 1 and the triangle inequality imply that if

$$d(\mathbf{y}_j, \mathbf{y}_i) > 2h \tag{1}$$

then $\mathbf{y}_i$, as well as all the items that follow it in the ordering, cannot be closer than $\mathbf{y}_{\text{curr}}$ to $\mathbf{x}$ [8]. Thus, we can terminate the partial ordering with respect to $\mathbf{y}_{\text{curr}}$ as soon as we find two consecutive items which are connected by an edge of weight greater than $2h$.

Furthermore, the triangle inequality ensures that all the closest matches of $\mathbf{x}$ are included in the pruned list at that stage of the partial ordering [8]. This can be useful in applications where a good match, though not necessarily the closest, can be obtained inexpensively. For example, our experiments in [1] suggest that a variation of the k-d tree proposed in [9] can yield good approximate matches at $O(\log n)$ scalar decisions. The hierarchical ordering can start at the approximate match, and terminate as soon as the terminating criterion is satisfied. All the closest matches of $\mathbf{x}$ are contained in the pruned list.

We note that the shortened list can be extended by relaxing the criterion in Equation 1 by a scale of h. For example, if the user finds that the items in the pruned list are too few, then the hierarchical ordering can continue by replacing the right-hand side of the inequality to $2.5h$, $3h$, $4h$, and so on, until the user's requirements are satisfied. We also note that the criterion does not guarantee that all the items in the pruned list are going to be closer to $\mathbf{x}$ than the discarded items. If the query is close to a cluster, the criterion may ignore other clusters which are farther than the current cluster, even when those other clusters may have items that are closer to the query than some of the items in the current cluster.

4 Results of Experiments

We demonstrate the technique proposed in this paper using data sets from vector quantisation (VQ) coding of images. Each data item is an image block from an 8-bit grayscale image. The database is a VQ codebook trained over an image using the LBG algorithm [10]. We use various image block and codebook sizes in our experiments. We show the results for two test images: `len` ("Lenna") is a 512×512 image of a woman, and `f18` is a 480×640 image of an F-18 jet fighter. The Euclidean distance function is used as the similarity measure.

Tables 1 and 2 show the average results over the total number of queries, `QTotal`. In Table

Image	Block size	Codebk size (n)	Performance:				
			QTotal	NNCost	ShiftsF	RSize	ShiftsR
len 512×512	2×2 (k=4)	1024	65536	11.552	0.380	2.418	0.049
		256	65536	5.898	0.362	2.419	0.050
	4×4 (k=16)	1024	16384	28.082	0.361	7.575	0.094
		256	16384	8.793	0.353	4.139	0.056
	8×8 (k=64)	256	4096	13.748	0.337	7.780	0.075
		128	4096	7.879	0.330	4.223	0.047
	16×16 (k=256)	128	1024	10.845	0.306	7.111	0.082
		64	1024	7.457	0.309	5.066	0.054
f18 480×640	2×2 (k=4)	1024	76800	9.425	0.164	16.344	0.025
		256	76800	3.880	0.243	2.301	0.019
	4×4 (k=16)	1024	19200	71.396	0.224	34.506	0.089
		256	19200	12.278	0.206	10.141	0.107
	8×8 (k=64)	256	4800	23.957	0.168	33.783	0.122
		128	4800	6.337	0.222	6.643	0.094
	16×16 (k=256)	128	1200	13.073	0.157	20.009	0.110
		64	1200	5.671	0.199	6.011	0.094

Table 1: Results for query sets which are the same as the VQ codebook's training data.

1, the query set is the same as the training set used to generate the VQ codebook. Thus, it is likely that a good match can be found for every query. In Table 2, however, the results for image `len_f18` were obtained by encoding `len` using VQ codebooks trained for `f18`, and vice versa. Since the query set is outside the training set, it is possible that some queries may not have a good match.

Column `NNCost` shows the average cost of finding the query's closest match. We used the fast NN search algorithm proposed in [1]. The algorithm uses $O(n)$ pre-computed information. The algorithm also uses the MCST to speed up the search. The cost of the search is expressed in the tables as the equivalent number of $O(k)$ distance function evaluations. Compared to the $O(kn)$ arithmetic cost of an actual similarity ranking, the cost of finding a closest match is considerably smaller.

We evaluate the approximate similarity ranking from the hierarchical ordering according to the "sortedness" of the resulting list. We measure this in terms of the amount of work that a sorting algorithm would require to re-arrange the items in the order that an actual similarity ranking would list them. We count the number of *element shifts* that the insertion sort algorithm has to perform to re-arrange the items so that their distances to the query are in increasing

Image	Block size	Codebk size (n)	Performance:				
			QTotal	NNCost	ShiftsF	RSize	ShiftsR
len_f18 512×512	2×2 (k=4)	1024	65536	21.871	0.271	18.495	0.040
		256	65536	4.525	0.318	2.818	0.034
	4×4 (k=16)	1024	16384	62.323	0.336	84.248	0.068
		256	16384	12.022	0.304	10.937	0.070
	8×8 (k=64)	256	4096	30.269	0.267	42.604	0.138
		128	4096	9.839	0.322	8.338	0.093
	16×16 (k=256)	128	1024	23.651	0.250	45.169	0.237
		64	1024	12.472	0.266	19.846	0.190
f18_len 480×640	2×2 (k=4)	1024	76800	15.416	0.328	7.740	0.074
		256	76800	7.323	0.328	4.529	0.097
	4×4 (k=16)	1024	19200	43.603	0.340	37.153	0.158
		256	19200	12.506	0.317	10.030	0.109
	8×8 (k=64)	256	4800	18.916	0.313	19.003	0.079
		128	4800	9.417	0.298	8.736	0.056
	16×16 (k=256)	128	1200	16.051	0.274	14.242	0.099
		64	1200	9.270	0.294	7.941	0.066

Table 2: Results for query sets which are outside the VQ codebook's training data.

order. We chose the insertion sort algorithm because it returns the best result if the distances were already in sorted order, so that no element in the list needs to be shifted. The worst case result occurs when the distances are in reversed (decreasing) order, resulting in $\frac{N(N-1)}{2}$ shifts (where N is the number of items in the list). We scale the results by taking the *ratio* between the actual number of element shifts over the number of shifts in the worst case. The performance value ranges from 0 (best) to 1 (worst).

Column `ShiftsF` shows the performance when all the items are ordered hierarchically. The values range from 0.157 to 0.380. We note from the results that the performance appears to be dependent on the relative distances between the items in the database. In both query sets `len` and `f18`, hierarchical ordering performed better on the VQ codebook which was trained over `f18`, than on the codebook which was trained over `len`. Because hierarchical ordering is based on a single-link hierarchical clustering of the data items, the technique is likely to perform better in cases where the items tend to form small dense clusters which are distant from each other.

Column `RSize` shows the average size of the pruned list when the criterion based on the triangle inequality is used to terminate the partial ordering. The comparatively small size of

Image	Block size	Codebk size (n)	Performance:					
			QTotal	$\mathtt{h}_{\mathrm{best}}$	h	ShiftsF	RSize	ShiftsR
len 512×512	2×2 (k=4)	1024	65536	4.831	7.104	0.380	6.974	0.130
		256	65536	6.495	9.300	0.361	5.012	0.135
	4×4 (k=16)	1024	16384	18.945	28.951	0.361	36.418	0.173
		256	16384	22.518	32.794	0.353	9.758	0.135
	8×8 (k=64)	256	4096	66.401	94.500	0.338	20.782	0.163
		128	4096	71.395	107.156	0.332	11.046	0.136
	16×16 (k=256)	128	1024	190.852	272.514	0.310	17.076	0.159
		64	1024	217.874	320.189	0.316	11.764	0.154

Table 3: Results for hierarchical ordering which starts from the approximate match obtained using a k-d tree. The query set is the same as the VQ codebook's training set.

the pruned list indicates that a large part of the result in Column ShiftsF is probably due to data items that may not need to be included in the ranking because they are too far away from the query. Column ShiftsR indicates the "sortedness" of the items in the pruned list. The comparatively small element shift ratios suggest that the items in the pruned list are often already arranged in nearly the same order in which an actual similarity ranking would list them.

Table 3 shows the results when a variation of the k-d tree is used to obtain the approximate match [1]. The k-d tree can be pre-computed, and then stored in $O(n)$ space. It takes just $O(\log n)$ scalar decisions to find an approximate match. Column h shows the average Euclidean distance between $\mathbf{x}$ and the approximate match. Note that the distances are relatively close compared to Column $\mathtt{h}_{\mathrm{best}}$, which shows the average Euclidean distance between $\mathbf{x}$ and the closest match. Column ShiftsF shows the performance when all the times are ordered hierachically starting from the approximate match. Column RSize shows the size of the pruned list when the terminating criterion is applied. As expected, the pruned list contains more items compared to the corresponding pruned list in Table 1, where the closest match is used. Column ShiftsR shows the element shift ratios in sorting the pruned list.

5 Conclusion

The results of our experiments suggest that hierarchical ordering is able to provide a good approximate similarity ranking at a fraction of the cost of an actual similarity ranking. The technique is efficient in terms of both arithmetic cost and storage space requirements. The cost of finding the closest match for the hierarchical ordering is only a fraction of the $O(kn)$

cost of an actual similarity ranking. Hierarchical ordering itself requires only scalar decisions to traverse the dendogram. It also requires only $O(n)$ space for the dendogram, compared to the $O(n^2)$ space required by the proximity table. If the similarity measure is a metric, then the triangle inequality provides a criterion for determining a partial list of ranked items.

The efficiency and performance of hierarchical ordering can allow researchers to consider sophisticated similarity measures despite the arithmetic complexity of those measures [11], [12]. Of interest to the authors are data mining and case-based reasoning applications, where large databases have been accumulated over the years in an *ad hoc* manner. Given an appropriate similarity measure, a single-link hierarchical clustering can provide the needed structure for such databases.

References

[1] Joselíto J. Chua. Fast full-search equivalent nearest-neighbour search algorithms. Master of Computing, School of Computer Science and Software Engineering, Faculty of Information Technology, Monash University, 1999.

[2] J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 10:54–64, 1969.

[3] F. James Rohlf. Single-link clustering algorithms. In P.R. Krishnaiah and L.N. Kanal, editors, *Handbook of Statistics, vol. 2*, pages 267–284. Elsevier Science Publishers, Amsterdam, The Netherlands, 1982.

[4] Xiaofeng Han, Pierre Kelsen, Vijaya Ramachandran, and Robert Tarjan. Computing minimal spanning subgraphs in linear time. *SIAM Journal on Computing*, 24(6):1332–1358, December 1995.

[5] Jon Louis Bentley and Jerome H. Friedman. Fast algorithms for constructing minimal spanning trees in coordinate spaces. *IEEE Transactions on Computers*, C-27(2):97–105, February 1978.

[6] D.B. Johnson and P. Metaxas. Optimal algorithms for the single and multiple vertex updating problems of a minimum spanning tree. *Algorithmica*, 16:633–648, 1996.

[7] C.-M. Huang, Q. Bi, G. S. Stiles, and R. W. Harris. Fast full search equivalent encoding algorithms for image compression using vector quantization. *IEEE Transactions on Image Processing*, 1(3):413–416, July 1992.

[8] Joselíto Chua and Peter Tischer. Minimal cost spanning trees for nearest-neighbour matching. In Masoud Mohammadian, editor, *Computational Intelligence for Modelling, Control and Automation: Intelligent Image Processing, Data Analysis and Information Retrieval*, pages 7–12. IOS Press, 1999.

[9] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(2):209–226, June 1977.

[10] Yoseph Linde, Andrés Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28(1):84–95, January 1980.

[11] Simone Santini and Ramesh Jain. Similarity measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):871–883, September 1999.

[12] Nicu Sebe, Michael S. Lew, and Dionysius P. Huijsmans. Toward improved ranking metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1132–1143, October 2000.