A Survey of Recent Advances in Hierarchical Clustering Algorithms

Article *in* The Computer Journal · November 1983

DOI: 10.1093/comjnl/26.4.354 · Source: DBLP

1 author:

Fionn Murtagh
University of Huddersfield
560 PUBLICATIONS **10,389** CITATIONS

A Survey of Recent Advances in Hierarchical Clustering Algorithms

F. Murtagh
Department of Computer Science, University College Dublin, Dublin 4, Ireland

It has often been asserted that since hierarchical clustering algorithms require pairwise interobject proximities, the complexity of these clustering procedures is at least $O(N^2)$. Recent work has disproved this by incorporating efficient nearest neighbour searching algorithms into the clustering algorithms. A general framework for hierarchical, agglomerative clustering algorithms is discussed here, which opens up the prospect of much improvement on current, widely-used algorithms. This 'progress report' details new algorithmic approaches in this area, and reviews recent results.

1. INTRODUCTION

The problems of automatically classifying data arise in many areas, and hierarchical classification offers a flexible, non-parametric approach. Substantial reviews of the clustering problematic can be obtained in Refs 1–3. Hierarchical clustering algorithms have tended to be somewhat more prominent than others, perhaps because they presuppose very little in the way of data characteristics or of *a priori* knowledge on the part of the analyst.

Progress on algorithms for the minimal spanning tree, and its associated single linkage hierarchical clustering method, has been spectacular in recent years (see Refs 4–7 and—less directly relevant here—Refs 8 and 9). Other clustering algorithms would, if efficient, often be of greater practical use, since the single linkage method has certain well-known practical disadvantages. These include the 'chaining' effect, and the lack of any immediate definition of cluster centre or representative (both of which are serious handicaps when clustering is used in such areas as information retrieval: for its use here, see Ref. 10).

Current, efficient hierarchical clustering algorithms include that of Sibson,[11] who gave an algorithm for constructing the single linkage hierarchy, and Defays,[12] who extends Sibson's approach to give a (not necessarily unique) complete link hierarchy. These approaches cannot be further improved, since they require all pairwise dissimilarities to be considered.

The algorithms to be discussed in this article focus instead on exactly what is required in order to carry out an agglomeration at any stage of the clustering: this is usually little more than the nearest neighbour points of specified points. Sections 2 and 3 will present clustering algorithms at a high level, removed from detailed implementation considerations. Complexity aspects of the algorithms will be discussed, but it will be assumed until Section 4 that brute-force nearest neighbour searching is used (i.e. $N-1$ comparisons to find the nearest neighbour of a given point among N). This is because experience with the incorporation of nearest-neighbour-finding enhancements on clustering algorithms is relatively limited in experimental scale and in the range of algorithms used, and because it will be clearer to take hierarchical clustering algorithms as they are currently employed and to show how and where the nearest neighbour subproblem can be demarcated.

Regarding terminology, we will be concerned with a set of objects with associated description vectors, i.e. a set of *points* in multidimensional space. A *dissimilarity* measure is defined on these, which is a positive semi-definite symmetric mapping of pairs of objects and/or clusters of objects onto the reals (i.e. $d(i,j) \geq 0$ and $d(i,j) = d(j,i)$ for objects or clusters i and j). Often the stronger *distance* is used, where in addition the triangular inequality is satisfied (i.e. $d(i,j) \leq d(i,k) + d(j,k)$). This is satisfied, for example, by the widely-used Euclidean distance. It is possible that the initially-given data might be in dissimilarity form, but the bulk of this survey will be focused on the more usual case where N objects each have a descriptor vector in real M-space.

2. BACKGROUND

2.1 General hierarchical clustering algorithm using dissimilarity-update formula

Very different algorithms can be given for the same hierarchical clustering method (see Ref. 13 for a review of many algorithms for the single linkage method). However, a general agglomerative algorithm for hierarchical clustering may be described informally as follows:

Algorithm 1

Step 1. Determine all interobject dissimilarities.
Step 2. Form cluster from two closest objects or clusters.
Step 3. Redefine dissimilarities between new cluster and other objects or clusters (all other interpoint dissimilarities remaining unchanged).
Step 4. Return to Step 2 until all objects are in the one cluster.

Ordinarily Step 1 will require $O(N^2)$ calculations, i.e. $N(N-1)/2$ interobject dissimilarities. The number of variables obviously effects the calculation time required, but they are usually considered constant for any particular set of data. In Steps 2 and 3, it might be worth while to consider keeping a sorted list of all dissimilarities under consideration (taking $O(N^2 \log N)$ time for the initial

CCC-0010-4620/83/0026-0354$03.00

sorting, plus update time thereafter). Otherwise $O(N^2)$ time will be required for any execution of Step 2. The hierarchy involves the construction of at most $N-1$ clusters, and so $N-1$ iterations (Steps 2, 3, and 4) are required. Step 3 can be carried out in $O(N)$ time and involves the use of the Lance–Williams combinatorial formula. If the objects or clusters just merged are indexed by i and j, and if k is any other object or cluster, then this formula is:[14]

$$d(i+j, k) = a(i)d(i, k) + a(j)d(j,k) + bd(i, j) + c|d(i, k) - d(j, k)|$$

where the values of a, b and c are dependent on the clustering strategy used, and common values are listed in Table 1. Values for a number of other methods are given by Jambu.[15]

The above recurrence formula not only unifies what might at first appear to be a number of very different procedures, but also aids in the analysis of subordinate questions, such as the conditions under which inversions (or reversals) take place, i.e.

$$d(i+j, k) \ngeq d(i, j) \quad \text{for some } i, j, k$$

(see Refs 16–18). However Algorithm 1 has not facilitated computational improvement, and for that we must consider an alternative algorithm.

2.2 Hierarchical clustering algorithms using original data and cluster data

Algorithm 2

Step 1. Examine all interpoint dissimilarities, and form cluster from two closest points.

Step 2. Replace two points clustered by representative point or by cluster fragment.

Step 3. Return to Step 1, treating clusters as well as remaining objects, until all objects are in one cluster.

This algorithm is justified by storage considerations, when the $O(N^2)$ storage required for a dissimilarity matrix is replaced by the $O(N)$ storage required for N initial objects and $O(N)$ storage for the $N-1$ (at most) clusters. Whether a cluster is represented by a centre point or alternatively by a subgraph or 'fragment' of interconnected points clearly divides the well-known clustering algorithms into geometric methods (the centroid method, the median method, and Ward's minimum variance method) and graph methods (the single, complete and average linkage methods). The formulae of the cluster centres for the former are given in Table 1. The term 'fragment' refers to a connected component in the case of the single link method and to a clique or complete subgraph in the case of the complete link method.

Apart from lessened storage requirements, the above algorithm can also be justified in that it can be made computationally very efficient by searching for two closest clusters, in Step 1, in a restricted, local region only. Before reviewing techniques for this, more details of Algorithm 2 are discussed in the next section. These are based on the nearest neighbours among the points or cluster centres, and more particularly on reciprocal nearest neighbours. The latter will be seen to be a basic 'atom' or 'building block' of most hierarchical clustering algorithms. With little loss of generality, and in accord with most current programs, we will assume that the hierarchies are strictly binary (i.e. that there are precisely

Table 1. Properties of six hierarchical clustering methods

Hierarchical clustering methods (and aliases)	Lance and Williams dissimilarity update formula	Co-ordinates of centre of cluster, which agglomerates clusters i and j	Dissimilarity between cluster centres g_i and g_j
Single link (nearest neighbour)	$a(i) = 0.5$ $b = 0$ $c = -0.5$ (More simply: min $\{d(i, k), d(j, k)\}$)	—	—
Complete link (diameter)	$a(i) = 0.5$ $b = 0$ $c = 0.5$ (More simply: max $\{d(i, k), d(j, k)\}$)	—	—
Group average (average link, UPGMA)	$a(i) = \lvert i\rvert/(\lvert i\rvert + \lvert j\rvert)$ $b = 0$ $c = 0$	—	—
Median (Gower's method, WPGMC)	$a(i) = 0.5$ $b = -0.25$ $c = 0$	$g = (g_i + g_j)/2$	$\lVert g_i - g_j\rVert^2$
Centroid (UPGMC)	$a(i) = \lvert i\rvert/(\lvert i\rvert + \lvert j\rvert)$ $b = -\lvert i\rvert\lvert j\rvert/(\lvert i\rvert + \lvert j\rvert)^2$ $c = 0$	$g = (\lvert i\rvert g_i + \lvert j\rvert g_j)/(\lvert i\rvert + \lvert j\rvert)$	$\lVert g_i - g_j\rVert^2$
Ward's method (minimum variance, error sum of squares)	$a(i) = (\lvert i\rvert + \lvert k\rvert)/(\lvert i\rvert + \lvert j\rvert + \lvert k\rvert)$ $b = -\lvert k\rvert/(\lvert i\rvert + \lvert j\rvert + \lvert k\rvert)$ $c = 0$	$g = (\lvert i\rvert g_i + \lvert j\rvert g_j)/(\lvert i\rvert + \lvert j\rvert)$	$(\lvert i\rvert\lvert j\rvert/(\lvert i\rvert + \lvert j\rvert))\lVert g_i - g_j\rVert^2$

Notes: $|i|$ = number of objects in cluster i.
g_i is a vector in M-space (where M is set of variables); either initial point or cluster centre.
$\|.\|$ is the norm in some metric, usually L_2.

$N-1$ agglomerations or clusters in the hierarchy): in practice, arbitrary resolving of equal dissimilarities might on occasion be necessitated.

3. FAST ALGORITHMS

3.1 Reciprocal nearest neighbours and the reducibility property

Let the nearest neighbour graph (NN-graph) be defined as a set of points, p, whose directed edges $\{(p, \mathrm{NN}(p))\}$ are such that $\mathrm{NN}(p)$ is the nearest neighbour of p. For any point p, three cases can be distinguished (Fig. 1). In case III, where $q = \mathrm{NN}(p)$, and $p = \mathrm{NN}(q)$, points p and q are referred to as mutual or reciprocal nearest neighbours (RNNs).

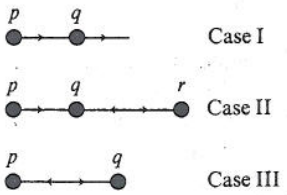

Figure 1. Three situations of interest in the NN-graph.

The NN-graph can be defined at all stages of the construction of the hierarchy for a geometric clustering method—where the vertices consist of remaining, unclustered, initial points and cluster representative points—and with suitable additional operations can also be used for a graph clustering method. Before looking at algorithms for these different areas (Algorithms 3 and 4 for geometric clustering methods, and Algorithms 5 and 6 for single linkage clustering), consider first the initial situation where the set of N multidimensional points is given. If i and j are any two RNNs, we have some ρ such that for any other point, k:

$$d(i, j) < \rho$$
$$d(i, k) > \rho$$
$$d(j, k) > \rho$$

and, using these inequalities and the Lance–Williams recurrence results, it is easily verified that

$$d(i + j, k) > \rho$$

for all clustering methods in Table 1 with the exception of the centroid and median methods. This is known as the reducibility property.[19] If verified by a clustering method, the merging of RNNs i and j into cluster $i + j$ requires the updating of the NN-graph only for those points which had i or j as NN (cf. Case II, Fig. 1). More importantly it means that all RNNs i and j can be simultaneously merged, without effecting the RNN properties of other parts of the NN-graph.

These two corollaries of the reducibility property will be used in the single and multiple cluster algorithms (Algorithms 3 and 4) for geometric clustering methods, to be studied in the next two subsections. Following this, the use of the NN-graph in graph clustering methods will be outlined in a similar perspective but with some noticeable differences.

3.2 Multiple cluster algorithm for geometric clustering methods[20–22]

Algorithm 3

Step 1. Determine all RNNs.
Step 2. Replace all RNNs by cluster centres, thus reducing set of points.
Step 3. Redetermine all NNs and hence RNNs.
Step 4. Goto Step 2 until only one point remains.

A brute-force approach to Step 1 takes $O(N^2)$ time for the dissimilarity calculations, as in Algorithms 1 and 2 above, but some techniques for significantly improving the complexity of this step are discussed in Section 4. Given the NN of each of the N points, the existence of RNNs is straightforwardly checked for in $O(N)$ time. The sum total of RNNs in Step 2, for all iterations (i.e. Steps 2, 3, and 4) is $N - 1$. Therefore Steps 2, 3, and 4, together, are $O(N^2)$ if the number of new NNs to be determined in Step 3 is some constant times the number of agglomerations.

Unfortunately this cannot be guaranteed in general: the determining of the NNs of new clusters presents no problem, but the NNs of points which have one or other of the RNN points as NN (cf. Case II of Fig. 1) must also be determined. The number of these points is unbounded. Special situations where the number of these points is bounded includes the case of points in the Euclidean plane. In the centroid and median methods, in this case, the distance used between cluster centres is the same as that used between initial points, and it can be easily verified that the maximum possible number of points which can simultaneously have some one point as NN is 6. Note however that even if the Euclidean distance is used for Ward's method, a non-Euclidean intercluster distance is employed (cf. Table 1), and so this result does not apply. Thus a worst case $O(N^2)$ operations must be assumed for Step 3, which coupled with $O(N)$ iterations (Steps 2, 3, 4), gives complexity $O(N^3)$ for Algorithm 3. In order to remedy the $O(N^3)$ worst-case performance of Algorithm 3, an algorithm which constructs a single cluster on each iteration will be discussed next.

Algorithm 3 is exact for all hierarchical clustering methods which satisfy the reducibility property (Section 3.1). As already mentioned, the centroid and median methods do not. For Algorithm 3 to provide exact results for these two methods, the RNNs would need to be examined in order of decreasing closeness in Step 2. The increased computational time required for this might favour instead the good, approximate centroid and median hierarchies produced by an unmodified Algorithm 3.

3.3 Single cluster algorithm for geometric clustering methods[23, 24]

Let an NN-chain be defined as the sequence of points

$$i, j = \mathrm{NN}(i),\ k = \mathrm{NN}(j), \ldots, q = \mathrm{NN}(p), p = \mathrm{NN}(q)$$

where i is an arbitrary point. Among the properties of a NN-chain we have the following:

1. The final two points (p and q, say, where p might be i) constitute an RNN pair.

2. Dissimilarities between adjacent points in an NN-chain are monotonically decreasing (it is assumed that no two dissimilarities are equal).
3. An NN-chain cannot contain a circuit.

Owing to property 2, we may say that the NN-chain is *grown towards increasing density*, since interpoint dissimilarity—hence sparseness—at the start of an NN-chain is greater than that at the end.

By continually growing an NN-chain, we have a single cluster algorithm for geometric clustering methods:

Algorithm 4

Step 1. Starting with arbitrary i, determine $j = \mathrm{NN}(i)$, $k = \mathrm{NN}(j), \ldots$ until $q = \mathrm{NN}(p)$ and $p = \mathrm{NN}(q)$.
Step 2. Agglomerate RNNs p and q, replacing with cluster centre.
Step 3. Continue NN-chain from point in chain prior to p (or if the first two points in the NN-chain gave a pair of RNNs start a NN-chain from any point).
Step 4. Return to Step 4 until only one point remains.

Since the NN-chain is grown, in Step 3, from where it ended in Step 2, it can be deduced that this algorithm requires $O(N^2)$ dissimilarity calculations. As for Algorithm 3, Algorithm 4 provides a good heuristic for the centroid and median methods. Evidently the execution of Algorithm 4 is dependent on the order in which the input data are presented, but the reducibility property ensures that the resulting hierarchy is unique and exact. In particular it may be noted that Algorithm 4 yields an exact minimum variance (Ward's method) hierarchy in $O(N^2)$ time and $O(N)$ space, and is thus optimal in the sense used by Sibson[11] when discussing the single linkage method.

3.4 Constructing the minimal spanning tree by subgraphs[6]

The single and multiple cluster algorithms for geometric clustering methods have not been generalized for all graph clustering methods. In this section, suggested approaches to single link hierarchical clustering will be discussed. These approaches are based on the minimal spanning tree (which is transformed into the hierarchy in $O(N)$ time[25]).

The following proposition has been used in the previous sections:

Proposition 1. Given a point-set, any pair of RNNs is a class or cluster of an agglomerative hierarchic clustering, if the hierarchic clustering method satisfies the reducibility property.

The single link method satisfies this property, but a stronger result is:

Proposition 2. Any NN-chain is a subset of a minimal spanning tree.

Any NN-chain, originating in an arbitrary point and ending in a pair of RNNs, therefore defines in reverse order a sequence of nested clusters in a single linkage hierarchic clustering. Instead of single and multiple cluster algorithms for geometric cluster methods, we have here single and multiple fragment algorithms, where a fragment is 'grown' from an NN-chain. The following is a single fragment algorithm.

Algorithm 5

Step 1. Construct an NN-chain; let q be the last point added, and call the NN-chain a fragment.
Step 2. Find r, the nearest point to q which is not in the fragment.
Step 3. If, for some i in the fragment, $d(i, \mathrm{NN}(i)) < d(q, r)$ then see if there is an s not in the fragment such that

$$d(i, \mathrm{NN}(i)) < d(i, s) < d(q, r)$$

If so, find the least such $d(i, s)$, and connect s to i; otherwise connect r to q.
Step 4. Redefine q to be the point whose link to the fragment is of least dissimilarity, and return to Step 2 until all points are in the fragment.

Step 3 is explained as follows. The point r could be connected to q. However it must be checked if a closer point could instead be connected to some other point in the fragment. The edge $(i, \mathrm{NN}(i))$ is in the fragment if i is in the fragment. If $d(i, \mathrm{NN}(i)) > d(q, r)$, and if some s is connected to i then $d(i, s) < d(q, r)$, which together imply: $d(i, s) < d(i, \mathrm{NN}(i))$. But then s is the NN of i, and from this contradiction it is seen that we were justified in connecting r to q.

Step 3 possibly necessitates 'climbing back' some way in the fragment (i.e. it requires the finding of NNs of a number of vertices in the fragment) and is best implemented using a list of vertices in the fragment, ordered by the smallest dissimilarity which connects them to the fragment (see Ref. 1, pp. 258–263, for a worked example of such a priority queue).

The analysis of this algorithm depends on the average number of iterations between Steps 2 and 3, i.e. if constant or $O(N)$, we get overall complexity of $O(N^2)$ or $O(N^3)$, respectively. Bentley and Friedman[6] suggest an approximate minimal spanning tree algorithm, where this number of iterations is held constant, but in general worst-case $O(N)$ must be assumed.

A better, multiple fragment algorithm with two separate stages (Steps 1, 2, and 3; and Steps 4 and 5) is as follows.

Algorithm 6

Step 1. Pick an arbitrary point.
Step 2. Construct an NN-chain from this point.
Step 3. Pick another isolated point, and return to Step 2 until all points are in one of m NN-chains.
Step 4. Connect the closest point in an arbitrary NN-chain (or fragment) to some other NN-chain (or fragment), using Steps 2, 3, and 4 of Algorithm 5.
Step 5. Return to Step 4 until all points are in the one fragment.

Algorithm 6 will work better if the NN-chains are long, i.e. if the points chosen in Steps 1 and 3 are in sparse regions. Following the iterated Steps 2 and 3, there are m NN-chains and hence $m - 1$ edges remaining to be found in the minimal spanning tree.

The principal computational advantage of Algorithms 5 and 6 lies in their ability to incorporate efficient NN-finding techniques. The latter algorithm, for example, is of $O(N \log N)$ complexity, using the approach of Bentley and Friedman.[6] We now review a number of these fast NN-finding techniques.

4. TECHNIQUES FOR NEAREST NEIGHBOUR FINDING

The finding of NNs, on which all the algorithms discussed are based, is 'local' in the sense that the NN of a point can only lie in some intelligently delimited region of the space of points. This justifies an initial, crude clustering of the set of points which then allows NNs, and hence the exact agglomerative hierarchy, to be determined very much more efficiently than by an exhaustive search approach. What follows is a short catalogue of preprocessing strategies for NN-finding, some of which have been used for clustering.

The preliminary hashing of all points onto directly addressable cells takes $O(N)$ time. The search for an NN then proceeds by examining all points which have been mapped onto the same cell, and if the given point could potentially have an NN in a surrounding layer of cells, then these must be explored also (Ref. 1, pp. 251–252; also Refs 7 and 26). For low dimensional data (<4), Rohlf[5] has discussed the construction of the minimal spanning tree and single link hierarchy using this approach; and Murtagh[27] has looked at results obtained for other clustering algorithms using uniformly distributed points in the plane.

As an alternative to the crude non-hierarchical clustering technique just described, a crude, divisive, hierarchical approach is provided by a multidimensional binary search tree—a generalization of a binary search tree. At successive levels of the tree, some co-ordinate is chosen and all points are associated with one or other of two offspring nodes according to whether they take values less than or greater than some cut-off value on this co-ordinate. Continually subdividing the points in this way, we eventually halt with a set of buckets, where each of the N points is associated with some one bucket. This divisive clustering takes $O(N \log N)$ time, on average, as in the case of the one-dimensional binary search tree. Similarly, average search time is $O(\log N)$, although more than one bucket might need to be searched to determine the NN of a point. Further description of this preprocessing method is to be found in Ref. 1, p. 248; results for the NN finding problem are given by Friedman, Bentley and Finkel;[28] and empirical results for minimal spanning tree construction are given by Bentley and Friedman.[6]

The branch and bound approach of Fukunaga and Narendra[29] uses some tree structuring of the points and determines in addition the centre and maximum deviation of points in each cluster. Then, given a current, potential NN, it may be established which clusters can be definitely excluded from the search for the NN.

Other heuristic improvements on brute-force NN-searching include the following. Friedman, Baskett and Shustek[30] use the value of points on some co-ordinate axis, in order to exclude from consideration all points with projections which are too distant on this axis. Kittler[31] and Richetin, Rives and Naranjo[32] use metrics other than the Euclidean to give bounds on the latter: the L_1 (city-block) and L_∞ (Chebyshev or max) lower- and upper-bound the L_2 metric, and less computation is required to determine the former two. Finally, Sethi[33] establishes three reference points in the space of points to be searched and uses these to determine a smaller region of the space to be searched.

5. DISCUSSION AND CONCLUSION

We have discussed the design of algorithms for hierarchical clustering which take the nearest neighbour problem as a more primitive task. A number of practical results reported in the literature have been noted. Much work remains to be accomplished in the area of NN searching; in particular, efficient algorithms for high-dimensional spaces are very much needed.

One major area of recent work, of direct relevance to hierarchical clustering algorithms, has not been discussed in this article. This is the construction of the minimal spanning tree, and hence the single link clustering, by first constructing a supergraph of the former. The Delauney triangulation and the Voronoi diagram (Thiessen polygons or Dirichlet tesselation) have been used for this purpose. The former is the dual of the latter, which in turn is the graph of the N given points, and the edges are the perpendicular bisectors of straight lines connecting neighbouring points. This supergraph approach has shown itself particularly powerful for $O(n \log n)$ worst case performance[4] and $O(N)$ expected time[7] algorithms for the minimal spanning tree. To date, however, this approach has been restricted to very low dimensions and has not been used for other clustering methods. The subgraph approaches, on the other hand, are versatile and allow a wide range of NN finding techniques to be incorporated.

We have attempted in this overview of recent work on clustering algorithms to show the centrality of such concepts as mutual nearest neighbours, the NN-chain, and the NN-graph. The former concept was much used by McQuitty from the early nineteen-sixties for particular clustering methods (see, e.g. Ref. 34). It was briefly surveyed by Bock (Ref. 35, pp. 308–312), and by Rohlf.[20] However it has in particular emerged as a basis for general clustering algorithms in a series of articles in the journal *Les Cahiers de l'Analyse des Données* (associated with J. P. Benzécri). From the point of view of general hierarchical clustering algorithms for large data sets, it appears to constitute the first major advance since the well-known combinatorial relationship formulated by Lance and Williams.[14]

Open problems are legion: the generalization of Algorithms 5 and 6 (the single and multiple fragment algorithms for the minimal spanning tree and single link clustering) to other graph clustering methods; the incorporation of other fast NN finding routines into Algorithms 3–6; and, most of all, further efforts in the technology of NN finding.

REFERENCES

1. R. C. T. Lee, Clustering analysis and its applications. In *Advances in Information Systems Science*, Edited by J. T. Tou, Vol. 8, pp. 169–292, Plenum Press, New York (1981).
2. A. D. Gordon, *Classification*, Chapman and Hall (1981).
3. R. Dubes and A. K. Jain, Clustering methodologies in exploratory data analysis. In *Advances in Computers*, Edited by M. C. Yovits, Vol. 19, Academic Press, New York (1980).
4. M. I. Shamos and D. Hoey, Closest-point problems. *Proceedings*

of the 16th IEEE Symposium on the Foundations of Computer Science 151–162 (1975).
5. F. J. Rohlf, A probabilistic minimum spanning tree algorithm, *Information Processing Letters* **7**, 44–48 (1978).
6. J.L. Bentley and J. H. Friedman, Fast algorithms for constructing minimal spanning trees in coordinate spaces. *IEEE Transactions on Computers* **C-27**, 97–105 (1978).
7. J. L. Bentley, B. W. Weide and A. C. Yao, Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software* **6**, 563–580 (1980).
8. A. C. Yao, An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees. *Information Processing Letters* **4**, 21–23 (1975).
9. D. Cheriton and R. E. Tarjan, Finding minimum spanning trees. *SIAM Journal of Computing* **5**, 724–742 (1976).
10. N. Jardine and C. J. van Rijsbergen, The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval* **7**, 217–240 (1971).
11. R. Sibson, SLINK: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal* **16**, 30–34 (1973).
12. D. Defays, An efficient algorithm for a complete link method. *The Computer Journal* **20**, 364–366 (1977).
13. F. J. Rohlf, Single link clustering algorithms, *IBM Research Report RC 8569*, 33 pp. (1980).
14. G. N. Lance and W. T. Williams, A general theory of classificatory sorting strategies. I. Hierarchical systems. *The Computer Journal* **9**, 373–380 (1967).
15. M. Jambu, *Classification Automatique pour l'Analyse des Données*, Dunod, Paris (1978). (*Cluster Analysis and Data Analysis*, North-Holland, Amsterdam (1983).
16. G. W. Milligan, Ultrametric hierarchical clustering algorithms. *Psychometrika* **44**, 343–346 (1979).
17. V. Batagelj, Note on ultrametric hierarchical clustering algorithms. *Psychometrika* **46**, 351–352 (1981).
18. E. Diday, Crossings, orders and ultrametrics. In *COMPSTAT 1982 Part I*, Edited by H. Caussinus *et al.*, pp. 186–191, Physica-Verlag, Wien (1982).
19. M. Bruynooghe, Classification ascendante hiérarchique, des grands ensembles de données: un algorithme rapide fondé sur la construction des voisinages réductibles. *Les Cahiers de l'Analyse des Données* **III**, 7–33 (1978).
20. F. J. Rohlf, Computational efficiency of agglomerative clustering algorithms, *IBM Research Report RC 6831*, 36 pp. (1977).
21. C. de Rham, La classification hiérarchique ascendante selon la méthode des voisins réciproques. *Les Cahiers de l'Analyse des Données* **V**, 135–144 (1980).
22. J. Juan, Le programme HIVOR de classification ascendante hiérarchique selon les voisins réciproques et le critère de la variance. *Les Cahiers de l'Analyse des Données* **VII**, 173–184 (1982).
23. J. P. Benzécri, Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques. *Les Cahiers de l'Analyse des Données* **VII**, 209–219 (1982).
24. J. Juan, Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques, *Les Cahiers de l'Analyse des Données* **VII**, 219–225 (1982).
25. F. J. Rohlf, Algorithm 76: hierarchical clustering using the minimum spanning tree. *The Computer Journal* **16**, 93–95 (1973).
26. C. Delannoy, Un algorithme rapide de recherche de plus proches voisins. *RAIRO Informatique/Computer Science* **14**, 275–286 (1980).
27. F. Murtagh, Expected-time complexity results for hierarchic clustering algorithms which use cluster centres, *Information Processing Letters* **16**, 237–241 (1983).
28. J. H. Friedman, J. L. Bentley and R. A. Finkel, An algorithm for finding best matches in logarithmic time. *ACM Transactions on Mathematical Software* **3**, 209–226 (1977).
29. K. Fukunaga and P. M. Narendra, A branch and bound algorithm for computing k-nearest neighbours. *IEEE Transactions on Computers* **C-24**, 750–753 (1975).
30. J. H. Friedman, F. Baskett and L. J. Shustek, An algorithm for finding nearest neighbours. *IEEE Transactions on Computers* **C-24**, 1000–1006 (1975).
31. J. Kittler, A method for determining k-nearest neighbours. *Kybernetes* **7**, 313–315 (1978).
32. M. Richetin, G. Rives and M. Naranjo, Algorithme rapide pour la détermination des k plus proches voisins. *RAIRO Informatique/Computer Science* **14**, 369–378 (1980).
33. I. K. Sethi, A fast algorithm for recognizing nearest neighbours, *IEEE Transactions on Systems, Man and Cybernetics* **SMC-11**, 245–248 (1981).
34. L. L. McQuitty, A mutual development of some typological theories and pattern-analytic methods. *Educational and Psychological Measurement* **27**, 21–46 (1967).
35. H. H. Bock, *Automatische Klassifikation.* Vandenhoeck und Rupprecht, Göttingen (1974).

Received January 1983