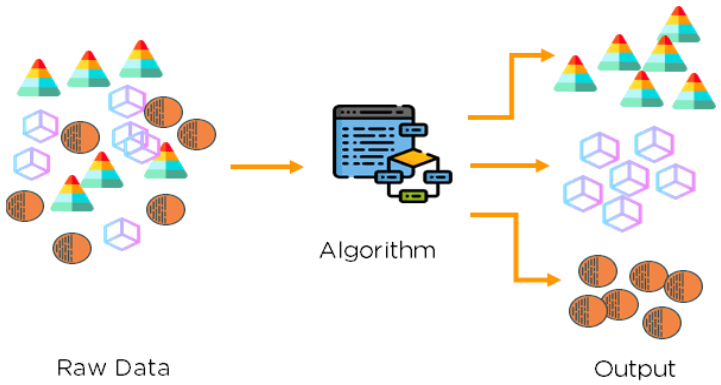
Top Papers on Clustering Algorithms



These papers provide a breadth of information about Clustering Algorithms that is generally useful and interesting from a computer science perspective.

Contents

8. CLARANS: A method for clustering objects for spatial data mining

9. FCM: The Fuzzy C-Means Clustering Algorithm

10. The Expectation Maximization Algorithm

11. The EM Algorithm

12. CURE: An Efficient Clustering Algorithm for Large Databases

13. A K-Means Clustering Algorithm

14. Algorithms for hierarchical clustering: An overview

15. Optimal algorithms for complete linkage clustering in d dimensions

SLINK : An optimally efficient algorithm for the single-link cluster method

R. Sibson

King's College Research Centre, King's College, Cambridge, and Cambridge University Statistical Laboratory

The SLINK algorithm carries out single-link (nearest-neighbour) cluster analysis on an arbitrary dissimilarity coefficient and provides a representation of the resultant dendrogram which can readily be converted into the usual tree-diagram. The algorithm achieves the theoretical order-of-magnitude bounds for both compactness of storage and speed of operation, and makes the application of the single-link method feasible for a number of OTU's well into the range 10^3 to 10^4. The algorithm is easily programmable in a variety of languages including FORTRAN.

(Received January 1972)

1. Background

The *single-link*, or *nearest-neighbour*, cluster method is one of the oldest methods of cluster analysis; it was suggested by workers in Poland in 1951 (Florek *et al.*, 1951a, b) and independently by McQuitty (1957) and Sneath (1957). Its obvious disadvantage—the 'chaining' effect—has long been well known, and has prompted the invention of many other cluster methods of either a hierarchic or a non-hierarchic (overlapping) kind; see Lance and Williams (1967) and Jardine and Sibson (1971). These methods also have their disadvantages. The alternative hierarchic methods have been criticised by Jardine and Sibson for lack of continuity, which they regard as being a far more severe defect than the chaining effect in many applications; it is also difficult to see how most of these methods could be programmed for more than a few hundred OTU's. Many problems with a large set of OTU's turn out on inspection to be distribution-mixture problems, rather than cluster-analysis problems in the strict sense in which the OTU's do not constitute a random sample from some larger population. Nevertheless there are many problems for which a large-scale cluster method is needed: this paper shows that the single-link method can be programmed efficiently enough to meet this need, and since its defects are well-enough understood and of such a nature as to cause it to be misleading only rather rarely, the method itself should generally be acceptable; in fact Jardine and Sibson have proposed an axiomatic framework for cluster methods within which it is uniquely acceptable, and in that context its defects must be viewed as those of hierarchic classification itself. They suggest overlapping methods to supplement single-link although these are applicable only up to about 100 OTU's. Fisher and van Ness (1971) have explored just which conditions are satisfied by the various hierarchic methods, and although they do not rule out other methods they point out that single-link (nearest-neighbour in their terminology) has many advantages. The present paper provides an algorithm for carrying out the single-link method which achieves the theoretical order of magnitude bounds on speed and compactness, and the author believes this algorithm to be superior in these respects to other general-purpose single-link algorithms known to him which have appeared in the literature (see Gower and Ross, 1969; Lance and Williams, 1967; Wishart, 1969; van Rijsbergen, 1970); it enables single-link cluster analysis to be applied on an unprecedented scale, and also renders its application to smaller numbers of OTU's a trivial matter in terms of computer usage.

2. The single-link method

Following Jardine and Sibson (1971), we define a *dissimilarity coefficient* (*DC*) to be a symmetric non-negative function $d: P \times P \to \mathscr{R}$ where P is the set of OTU's, and where $d(a, a) = 0$ for all $a \in P$. We also define a *dendrogram* to be a function $c: [0, \infty) \to \mathrm{E}(P)$, where $\mathrm{E}(P)$ is the set of equivalence relations on P, and c satisfies the conditions

$$h \leqslant h' \text{ implies } c(h) \subseteq c(h')$$
$$c(h) \text{ is eventually } P \times P$$
$$c(h + \delta) = c(h) \text{ for all small enough } \delta > 0$$

Thus a dendrogram is a nested sequence of partitions with associated numerical levels, the partition at a high enough level being the whole set P. A dendrogram is usually represented as the familiar tree-diagram, but there is a great deal of freedom—freedom which can be misused—over the order in which the OTU's are disposed along the baseline; this order forms no part of the dendrogram as such. The single-link method of cluster analysis is defined very simply as follows. Let d be the dissimilarity coefficient. At a fixed level h consider the graph whose vertices are OTU's and whose edges link just those pairs of OTU's of dissimilarity at most h. Then $c(h)$ is the equivalence relation corresponding to the partition of P defined by the connected components of this graph. It is very easy to check that the $c(h)$ defined in this way for different values of h do in fact give a function c which satisfies the conditions for a dendrogram. The transformation $d \to c$ so defined is the single-link cluster method. Some authors have regarded the partition $c(h)$ at one level or at some small number of levels as constituting the result of applying the method; we shall take the more usual and simple point of view that it is the whole dendrogram which is the result of the method.

3. Order-of-magnitude limitations

A dendrogram on N OTU's can have up to $N - 1$ distinct *splitting levels*—levels at which $c(h)$ changes—and so at the very least storage of $O(N)$ is required for a dendrogram. There are in fact numerous ways of achieving this order-of-magnitude bound. A DC on N objects can take up to $\frac{1}{2}N(N - 1)$ distinct values, and most cluster methods, in particular the single-link method, can be affected by changes in any one of these, so a cluster method operating on a DC will have a time-dependence at least $O(N^2)$ because each DC value must be examined at least once. The DC is the starting-point for cluster analysis, but almost always it is obtained from data held separately for each OTU. If the DC is to be held in core storage for random access, $O(N^2)$ locations will be needed, whereas both the original data and the dendrogram only require $O(N)$ locations, although if there is much data the constant may be large. Thus we want to avoid holding the DC in core if possible, and this is a failing of most clustering algorithms, which require repeated random access to the DC, for example to sort the values into numerical order. The SLINK algorithm avoids this problem by using the DC values a part-row at a time—at stage n random

access is needed only to values of the form $d(i, n)$ for $i < n$—and no sorting or rearrangement procedures are employed. The storage needed for a part-row is again $O(N)$, and so provided the DC values can be either generated on demand in the order 2-1; 3-1, 3-2; 4-1, 4-2, 4-3; 5-1, . . . or read in this order from an input stream or device, having been generated and written in this order to, for example, disc store, then the core store requirement is only $O(N)$ for the cluster method.

4. The pointer representation

Although the characterisation as a function $c:[0, \infty) \to E(P)$ certainly captures what is meant by a dendrogram, it is clearly not how the information would actually be kept. There are many ways of specifying a dendrogram on N objects in about $2N$ function values; we shall achieve it by means of two functions each defined on the set $1, \ldots, N$. The pair of functions will be called a *pointer representation*. $\pi:1, \ldots, N \to 1, \ldots, N$ and $\lambda:1, \ldots, N \to 1, \ldots, N \to [0, \infty]$ constitute a pointer representation if the following conditions hold

$$\pi(N) = N \qquad \lambda(N) = \infty$$
$$\pi(i) > i \qquad \lambda(\pi(i)) > \lambda(i) \qquad \text{for } i < N$$

We shall show that there is a natural 1-1 correspondence between pointer representations and dendrograms. Suppose first that c is a dendrogram. Define π, λ for $i < N$ by

$$\lambda(i) = \inf \{h:\exists j > i \text{ with } (i, j) \in c(h)\}$$
$$\pi(i) = \max \{j:(i, j) \in c(\lambda(i))\}$$

Thus $\lambda(i)$ is the lowest level at which i is no longer the last object in its cluster, and $\pi(i)$ is the last object in the cluster which it then joins; we are, of course, regarding the OTU's in P as being labelled by the integers $1, \ldots, N$. It is easy to see that π, λ so defined is a pointer representation. Now suppose that we are given a pointer representation π, λ. We define a function σ by taking $\sigma(i, h)$ to be the first element k in the sequence

$$i, \pi(i), \pi^2(i), \pi^3(i), \ldots, N$$

for which $\lambda(k) > h$. Then define

$$c(h) = \{(i, j) : \sigma(i, h) = \sigma(j, h)\}$$

It is easy to check that c defined in this way is a dendrogram. We now prove that these two transformations are mutually inverse.

Lemma

The transformations $c \to \pi, \lambda$ and $\pi, \lambda \to c$ defined above are mutually inverse, and so constitute a 1-1 correspondence between dendrograms and pointer representations.

Proof

We prove that $c \to \pi, \lambda \to c'$ in fact leads back to c, and that $\pi, \lambda \to c \to \pi', \lambda'$ leads back to π, λ. Consider first $c \to \pi, \lambda \to c'$. By definition $c'(h) = \{i, j) : \sigma(i, h) = \sigma(j, h)\}$. Now $(i, \sigma(i, h)) \in c(h)$ and $(j, \sigma(j, h)) \in c(h)$, so if $\sigma(i, h) = \sigma(j, h)$ we have $(i, j) \in c(h)$, that is, $c'(h) \subseteq c(h)$. Conversely, if $(i, j) \in c(h)$ then $(\sigma(i, h), \sigma(j, h)) \in c(h)$. Suppose that these are not equal; without loss of generality $\sigma(i, h) < \sigma(j, h)$. Then $\lambda(\sigma(i, h)) \leqslant h$, a contradiction. We deduce that $c(h) \subseteq c'(h)$ and hence that $c(h) = c'(h)$, that is, $c = c'$. Now consider $\pi, \lambda \to c \to \pi', \lambda'$. λ' is defined by

$$\lambda'(i) = \inf \{h:\exists j > i \text{ with } (i, j) \in c(h)\}$$
$$= \inf \{h:\exists j > i \text{ with } \sigma(i, h) = \sigma(j, h)\}$$

But $\sigma(i, h)$ is such a j if one exists, so

$$\lambda'(i) = \inf \{h:\sigma(i, h) > i\}$$
$$= \lambda(i)$$

Now

$$\pi'(i) = \max \{j:(i, j) \in c(\lambda'(i))\}$$
$$= \max \{j:(i, j) \in c(\lambda(i))\}$$
$$= \max \{j:\sigma(i, \lambda(i)) = \sigma(j, \lambda(i))\}$$
$$= \max \{j:\pi(i) = \sigma(j, \lambda(i))\}$$
$$= \pi(i)$$

So $\pi', \lambda' = \pi, \lambda$ and the proof is complete.

5. Recursive updating of the pointer representation

Our reason for considering the pointer representation of a dendrogram rather than any other comparably compact representation is that the pointer representation can be updated on the inclusion of a new OTU in a highly efficient way. We shall use the phrase 'the dendrogram on the first n OTU's' to mean the single-link dendrogram obtained from the restriction of the DC to the first n OTU's; this will in general be different from the restriction to the first n OTU's of the single-link dendrogram on all N OTU's, and the latter is a construct which we shall not use. Quantities relating to the dendrogram on the first n OTU's will be given subscript n, so the dendrogram is c_n and its pointer representation is π_n, λ_n.

For given n we define $\mu_n(i)$ recursively on i:

$$\mu_n(i) = \min \{d(i, n + 1), \min_{\pi_n(j) = i} \max \{\mu_n(j), \lambda_n(j)\}\} .$$

Thus $\mu_n(i)$ is defined for $i = 1, \ldots, n$ and since

$$\mu_n(i) \leqslant d(i, n + 1)$$

and d is a (finite) DC, $\mu_n(i)$ is finite for all i. We then define π, λ, which we shall prove to be the pointer representation of c_{n+1}, that is, π_{n+1}, λ_{n+1}, as follows.

$$\pi(n + 1) = n + 1 \qquad \lambda(n + 1) = \infty$$
$$\lambda(i) = \min \{\mu_n(i), \lambda_n(i)\} \text{ for } i < n + 1$$
$$\pi(i) = \pi_n(i), \text{ except that if } \mu_n(i) \leqslant \lambda_n(i) \text{ or}$$
$$\mu_n(\pi_n(i)) \leqslant \lambda_n(i) \text{ then } \pi(i) = n + 1, \text{ again for } i < n + 1 .$$

Lemma

$$\pi, \lambda = \pi_{n+1}, \lambda_{n+1}$$

Proof

We show first that π, λ is a pointer representation. Certainly $\pi(n + 1) = n + 1$, $\lambda(n + 1) = \infty$, so consider $i < n + 1$. $\pi(i) = \pi_n(i) > i$ or $= n + 1 > i$ if $i < n$, and if $i = n$ then $\mu_n(n) < \infty = \lambda_n(n)$ so $\pi(n) = n + 1 > n$. Thus in all cases $\pi(n) > i$ if $i < n + 1$. If $\pi(i) = n + 1$, and $i < n$, then $\lambda(i) < \infty = \lambda(n + 1)$, and if $i = n$ then $\lambda(n) = \mu_n(n) < \infty = \lambda(n + 1)$. If $\pi(i) = \pi_n(i)$ then $\mu_n(i) > \lambda_n(i)$ and $\mu_n(\pi_n(i)) > \lambda_n(i)$ so $\lambda(\pi(i)) = \lambda(\pi_n(i)) = \min \{\mu_n(\pi_n(i)), \lambda_n(\pi_n(i))\} > \lambda_n(i) = \lambda(i)$. In all cases we have $i < n + 1$ implies $\lambda(i) < \lambda(\pi(i))$. Having established that π, λ is a pointer representation, we must now show that it in fact represents the right dendrogram.

Consider some fixed level h. The clusters for c_{n+1} at level h are related to those for c_n as follows: add a one-OTU cluster consisting just of $n + 1$; unite with this each cluster containing an OTU i such that $d(i, n + 1) \leqslant h$. Define $\kappa_n(i, h) = \{j:(i, j) \in c_n(h)\}$. Then we can express this process in terms of σ by saying that $\sigma_{n+1}(i, h) = \sigma_n(i, h)$ unless there exists $j \in \kappa_n(i, h)$ with $d(j, n + 1) \leqslant h$, in which case $\sigma_{n+1}(i, h) = n + 1$. To establish that $\pi, \lambda = \pi_{n+1}, \lambda_{n+1}$ it will be enough to show that σ defined in terms of π, λ has the property required of σ_{n+1}, since clearly $\pi, \lambda \to \sigma$ is 1-1. It is easy to see that if $\sigma(i, h) \neq \sigma_n(i, h)$ then $\sigma(i, h) = n + 1$, so it is enough to check that $\sigma(i, h) = n + 1$ if and only if there exists $j \in \kappa_n(i, h)$ with $d(j, n + 1) \leqslant h$. Now

$\mu_n(\sigma_n(i, h)) \leqslant h$
if and only if
either $d(\sigma_n(i, h), n + 1) \leqslant h$
or for some j such that $\pi_n(j) = \sigma_n(i, h)$
we have $\mu_n(j) \leqslant h$ and $\lambda_n(j) \leqslant h$

i.e. if and only if

either $d(\sigma_n(i,h), n+1) \leqslant h$
or for some $j \in \kappa_n(i,h)$ such that
$\pi_n(j) = \sigma_n(i,h)$ we have $\mu_n(j) \leqslant h$

and so by an inductive argument we have

$\mu_n(\sigma_n(i,h)) \leqslant h$ if and only if there exists
$j \in \kappa_n(i,h)$ such that $d(j, n+1) \leqslant h$

Now $\sigma(i,h) = n+1$ if and only if
either $\pi(j) = n+1$ for some $j = i, \pi_n(i), \ldots < \sigma_n(i,h)$
or $\mu_n(\sigma_n(i,h)) \leqslant h$

But if the first of these alternatives holds, we must have
$\lambda_n(j) \geqslant \mu_n(\pi_n(j))$ for some $j = i, \pi_n(i), \ldots < \sigma_n(i,h)$
or $\lambda_n(j) \geqslant \mu_n(j)$ for some $j = i, \pi_n(i), \ldots < \sigma_n(i,h)$

and since for such a j $\lambda_n(j) \leqslant h$, this implies that for some $j \in \kappa_n(i,h)$ we have $\mu_n(j) \leqslant h$ and hence $\mu_n(\sigma_n(i,h)) \leqslant h$. Thus the first alternative implies the second, and

$\sigma(i,h) = n+1$
if and only if $\mu_n(\sigma_n(i,h)) \leqslant h$.
if and only if there exists $j \in \kappa_n(i,h)$
with $d(j, n+1) \leqslant h$
if and only if $\sigma_{n+1}(i,h) = n+1$

and this completes the proof.

If we start with π_1, λ_1, which must be given by $\pi_1(1) = 1$, $\lambda_1(1) = \infty$, then after $N-1$ steps of the above recursive process, we shall obtain π_N, λ_N which is the pointer representation of the single-link dendrogram on the whole set $P = 1, \ldots, N$.

6. The SLINK algorithm

The SLINK algorithm is simply a convenient way of carrying out the recursive process computationally. Three arrays of dimension N are used, and we shall denote them by Π, Λ, M. Suppose that Π, Λ contain π_n, λ_n in their first n locations. Then the SLINK algorithm overwrites these to place π_{n+1}, λ_{n+1} in the first $n+1$ locations as follows:

1. Set $\Pi(n+1)$ to $n+1$, $\Lambda(n+1)$ to ∞
2. Set $M(i)$ to $d(i, n+1)$ for $i = 1, \ldots, n$
3. For i increasing from 1 to n
   if $\Lambda(i) \geqslant M(i)$
   set $M(\Pi(i))$ to min $\{M(\Pi(i)), \Lambda(i)\}$
   set $\Lambda(i)$ to $M(i)$
   set $\Pi(i)$ to $n+1$
   if $\Lambda(i) < M(i)$
   set $M(\Pi(i))$ to min $\{M(\Pi(i)), M(i)\}$
4. For i increasing from 1 to n
   if $\Lambda(i) \geqslant \Lambda(\Pi(i))$
   set $\Pi(i)$ to $n+1$

The total space needed for this process, assuming that the DC values are available in the correct order, is clearly $O(N)$—in fact $3N$ plus overheads—and the number of operations needed to find π_N, λ_N is $O(N^2)$, so, as claimed, the SLINK algorithm constructs a representation of the single-link dendrogram in a way which is optimally efficient in order-of-magnitude terms. It is also clear that the amount of work done for each dissimilarity value is very small: generate or read it and load it into M; check it against the value in Λ and adjust values accordingly; check Λ entries against one another. It seems unlikely that this scheme of operations can be substantially reduced, and so it is unlikely that any other algorithm can improve much on the constant multiplying N^2 in any given language/machine context.

7. Classifiability

Jardine and Sibson (1971) suggest the use of the quantity

$$\Delta_1 = \sum_{i<j} (d(i,j) - d^*(i,j)) / \sum_{i<j} d(i,j)$$

as a measure of classifiability, where $d^*(i,j)$ is the ultrametric DC corresponding to the single-link dendrogram c and is defined by

$$d^*(i,j) = \inf \{h : (i,j) \in c(h)\} .$$

The smaller Δ_1 is, the more amenable to single-link classification the data is. The calculation of Δ_1 can readily be incorporated into an implementation of the SLINK algorithm, and this is recommended.

8. Presentation of results

The user of a cluster method may reasonably expect to be provided with output in a form which he can readily appreciate, and this will usually take the form of numerical output from which a tree-diagram can easily be drawn, possibly accompanied by the tree-diagram itself, either drawn on a plotter or approximated on a line-printer. For most purposes the latter is adequate. The pointer representation of a dendrogram is not particularly helpful from the user's point of view, and it is desirable to convert it into another representation called the *packed representation* for output. The packed representation consists of two functions τ, ν defined as follows.

$$\nu(i) = \lambda(\tau(i))$$
$$\tau^{-1}(\pi(\tau(i))) > i \text{ if } i < n, \text{ and}$$
$$\nu(j) \leqslant \nu(i) \text{ if } i \leqslant j < \tau^{-1}(\pi(\tau(i)))$$

This in fact characterises the dendrogram uniquely, and it is not difficult to convert the pointer representation to the packed representation, the conversion taking time $O(N^2)$ with a very small coefficient for N^2. It is convenient to provide an extra array of dimension N to facilitate the conversion, so the total store size is $4N$ plus overheads. The packed form representation is a numerically coded form of a tree-diagram, which may be constructed from it as follows: in positions $1, \ldots, N$ along the baseline insert OTU numbers, the number in position i being $\tau(i)$; above this draw a vertical to height $\nu(i)$ above the baseline; when all verticals have been drawn, draw a horizontal to the right (that is, in the direction of increasing position number) until it meets another vertical. This will give a tree-diagram representing the dendrogram, but with all vertical stems displaced to the extreme right of the clusters which they represent. This form of tree-diagram can be produced extremely easily from the packed form output on a line-printer, and this is normally to be recommended. If a more conventional form of tree-diagram is wanted, then either a more elaborate computer graphics technique can be used, or the dendrogram can simply be re-drawn by hand; this is easy because the OTU's are presented by the packed representation in a suitable order for a tree-diagram to be drawn on them.

Appendix

A FORTRAN SLINK PROGRAM

The program given here calculates the single-link dendrogram from a DC read in value-by-value from an input stream. Much of the main subroutine is special to this case, but the subprograms called from it are quite general and have been separated out to allow them to be used in calling programs designed, for example, to work with an internally generated DC. The calling program for the subroutine SLINK must declare NA, NB as integer arrays and HA, HB as real arrays, all singly subscripted and of the same dimension, and must set NMXOBJ to their dimension and TOP to a large positive real value such that TOP-1.0 is larger then every DC value. It must also set the stream numbers NRDATA, NWRECD, NPDEND as appropriate. Subroutine RCLOCK should be provided to set T to the time in seconds (data type REAL) from some appropriate point in the calling program. Experience with this program shows that it spends almost all its time reading DC values, and this emphasises the desirability of using internally generated DC values, or at least of avoiding the FORTRAN I/O package, for any substantial number of OTU's. The time taken

by the main part of the program *excluding* the reading or generation of DC values is, on Cambridge University Computer Laboratory TITAN, approximately 100 seconds for $N = 1{,}000$, and increases as N^2.

```
C
C
C     FORTRAN SLINK 003 CREATED 15/12/71
C
C
C     Calling program sets stream numbers, value of TOP (infinity)
C     and value of NMXOBJ (dimension of arrays).
C
          SUBROUTINE SLINK(NA,NB,HA,HB,NMXOBJ,TOP,NRDATA,NWRECD,NPDEND)
          DIMENSION NA(NMXOBJ),NB(NMXOBJ),HA(NMXOBJ),HB(NMXOBJ),REF(2),
     1    TITLE(6)
          DATA AD,AS,AW/1HD,1HS,1HW/
C
C     Read reference code, number of objects (OTUs), type (S for
C     similarities, otherwise dissimilarities) and mode (W for whole
C     matrix, D for subdiagonal with diagonal, otherwise strictly
C     subdiagonal).
C
          READ(NRDATA,9001) REF(1),REF(2),NOBJ,ATYPE,AMODE
C
C     Check that number of objects is within range, that type is not S,
C     and that mode is not W or D.
C
          IF(NOBJ.LT.2.OR.NOBJ.GT.NMXOBJ) STOP 1
          IF(ATYPE.EQ.AS) STOP 2
          IF(AMODE.EQ.AW.OR.AMODE.EQ.AD) STOP 3
C
C     Initialise for one object.
C
          NMISS = 0
          SIZE = 0.0
          NA(1) = 1
          HA(1) = TOP
C
C     For each of the remaining objects set NA(I) to I and HA(I)
C     to TOP, and read the current part-row into HB.
C
          DO 1  I = 2,NOBJ
          I1 = I-1
          NA(I) = I
          HA(I) = TOP
          READ(NRDATA,9002) (HB(J), J = 1,I1)
C
C     Check for missing DC values, signalled by negative entry, and
C     replace them by TOP-1.0.  Update NMISS, the number of missing
C     DC values, and SIZE, the sum of the DC values.
C
          DO 2  J = 1,I1
          IF(HB(J)) 3,2,2
    3     HB(J) = TOP-1.0
          NMISS = NMISS+1
    2     SIZE = SIZE+HB(J)
C
C     SLINK1 is a subroutine which carries out the rest of the SLINK
C     algorithm to produce the pointer representation of the complete
C     dendrogram in NA and HA.
C
    1     CALL SLINK1(NA,HA,HB,I1,NMXOBJ)
C
C     SLINK2 is a subroutine which converts the pointer representation
C     into the packed representation by a chain-building method.
C
          CALL SLINK2(NA,NB,HA,HB,NOBJ,NMXOBJ)
C
C     Object labels are read into HB and finally a title for the DC
C     is read.  SCALE is calculated as the largest value in HA, and
C     the packed representation and other information is written to a
C     printer-type stream.
C
          DO 4  I = 1,NOBJ
          J = NB(I)
    4     READ(NRDATA,9003) HB(J)
          READ(NRDATA,9004) (TITLE(J), J = 1,6)
          WRITE(NWRECD,9005) (TITLE(J), J = 1,6),REF(1),REF(2),NOBJ
          SCALE = 0.0
          DO 5  I = 1,NOBJ
          IF(HA(I)-TOP+1.0) 6,7,7
    6     SCALE = AMAX1(SCALE,HA(I))
          WRITE(NWRECD,9006) HB(I),HA(I)
          GOTO 5
    7     WRITE(NWRECD,9007) HB(I)
    5     CONTINUE
C
C     If there is missing data this is reported, otherwise the value
C     of DELTA-ONE-HAT is calculated using the function SLINK3,
C     which returns the sum of the values of the ultrametric DC
C     corresponding to the dendrogram.
C
          IF(NMISS) 999,8,9
    9     NPAIR = NOBJ*(NOBJ-1)/2
          WRITE(NWRECD,9008) NMISS,NPAIR
          GOTO 10
    8     DELHAT = (SIZE-SLINK3(NA,HA,NOBJ,NMXOBJ))/SIZE
          WRITE(NWRECD,9009) DELHAT
   10     DO 11  I = 1,NOBJ
          IF(HA(I)-TOP+1.0) 11,12,12
   12     HA(I) = -1.0
   11     CONTINUE
C
C     The packed representation is written to a punch-type stream
C     and finally the runtime in seconds is calculated.
C
          WRITE(NPDEND,9010) REF(1),NOBJ,SCALE,((HB(I),HA(I)), I = 1,NOBJ)
          WRITE(NPDEND,9004) (TITLE(J), J = 1,6)
          CALL RCLOCK(T)
          WRITE(NWRECD,9011) T
          RETURN
C
C     Dummy label.
C
  999     STOP 0
C
C     FORMAT statements.
C
 9001   FORMAT(4X,2A4/I5,2A1)
 9002   FORMAT(F10.4)
 9003   FORMAT(A4)
 9004   FORMAT(3A4/3A4)
 9005   FORMAT(1H1//1H0,34HFORTRAN SLINK 003 CREATED 15/12/71//1H0,
     1    6HDC IS ,6A4/1H0,10HREFERENCE ,2A4/1H0,3HON ,I4,8H OBJECTS//
     2    1H0,25HDENDROGRAM IN PACKED FORM/1H0,14HOBJECT   LEVEL/)
 9006   FORMAT(1H ,1X,A4,1X,F10.4)
 9007   FORMAT(1H ,1X,A4,6X,1H-)
 9008   FORMAT(/1H0,I8,26H DC VALUES MISSING OUT OF ,I8)
 9009   FORMAT(/1H0,17HDELTA-ONE-HAT IS ,F6.4)
 9010   FORMAT(4HDATA,A4,4H7777/I5,1X,F10.4/(A4,1X,F10.4))
 9011   FORMAT(/1H0,11HJOB RAN IN ,F7.2,5H SECS)
        END
C
C
C
        SUBROUTINE SLINK1(NA,HA,HB,I1,NMXOBJ)
        DIMENSION NA(NMXOBJ),HA(NMXOBJ),HB(NMXOBJ)
        DO 1  J = 1,I1
        NEXT = NA(J)
        IF(HA(J)-HB(J)) 2,3,3
    2   H = HB(J)
        IF(HB(NEXT)-H) 1,1,4
    3   H = HA(J)
        NA(J) = I1+1
        HA(J) = HB(J)
        IF(HB(NEXT)-H) 1,1,4
    4   HB(NEXT) = H
    1   CONTINUE
        DO 5  J = 1,I1
        NEXT = NA(J)
        IF(HA(J)-HA(NEXT)) 5,6
    6   NA(J) = I1+1
    5   CONTINUE
        RETURN
        END
C
C
C
        SUBROUTINE SLINK2(NA,NB,HA,HB,NOBJ,NMXOBJ)
        DIMENSION NA(NMXOBJ),NB(NMXOBJ),HA(NMXOBJ),HB(NMXOBJ)
        NB(NOBJ) = NOBJ
        DO 1  N = 2,NOBJ
        H = HA(NOBJ+1-N)
        NEXT = NOBJ
    2   NOW = NEXT
        NEXT = NB(NOW)
        IF(H-HA(NEXT)) 3,2,2
    3   NB(NOW) = NOBJ+1-N
    1   NB(NOBJ+1-N) = NEXT
        NEXT = NB(NOBJ)
    4   NOW = NEXT
        N = NA(NOW)
        IF(NB(NOW)) 5,999,6
    6   NEXT = NB(NOW)
        IF(NB(N)) 7,999,8
    8   NB(NOW) = NB(N)
        NB(N) = -NOW
        IF(NEXT-NOBJ) 4,11,999
    7   NOSE = -NB(N)
        NB(NOW) = NB(NOSE)
        NB(N) = -NOW
        NA(NOW) = NOSE
        IF(NEXT-NOBJ) 4,11,999
    5   NOSE = -NB(NOW)
        NEXT = NB(NOSE)
        IF(NB(N)) 9,999,10
   10   NB(NOSE) = NB(N)
        NB(N) = -NOSE
        IF(NEXT-NOBJ) 4,11,999
    9   NA(NOW) = -NB(N)
        NB(N) = -NOSE
        N = NA(NOW)
        NB(NOSE) = NB(N)
        IF(NEXT-NOBJ) 4,11,999
   11   NEXT = -NB(NOBJ)
        DO 12  N = 1,NOBJ
        HB(N) = HA(N)
        NB(N) = NEXT
   12   NEXT = NA(NEXT)
        DO 13  N = 1,NOBJ
        NOW = NB(N)
        NA(N) = NOW
   13   HA(N) = HB(NOW)
        DO 14  N = 1,NOBJ
        NOW = NA(N)
   14   NB(NOW) = N
        RETURN
  999   STOP 0
        END
C
C
C
        FUNCTION SLINK3(NA,HA,NOBJ,NMXOBJ)
        DIMENSION NA(NMXOBJ),HA(NMXOBJ)
        NOBJ1 = NOBJ-1
        SLINK3 = 0.0
        DO 1  I = 1,NOBJ1
        DO 2  J = 1,I
        N1 = I+1-J
        IF(HA(N1)-HA(I)) 2,2,3
    2   CONTINUE
        N1 = 0
    3   I1 = I+1
        DO 4  J = I1,NOBJ
        IF(HA(J)-HA(I)) 4,1,1
    4   CONTINUE
    1   SLINK3 = SLINK3+FLOAT((I-N1)*(J-I))*HA(I)
        RETURN
        END
```


References

FISHER, L., and VAN NESS, J. W. (1971). Admissible clustering procedures, *Biometrika*, Vol. 58, pp. 91-104.

FLOREK, K., ŁUKASZEWICZ, J., PERKAL, J., STEINHAUS, H., and ZUBRZYCKI, S. (1951a). Sur la liaison et la division des points d'un ensemble fini, *Colloquium Math.*, Vol. 2, pp. 282-285.

FLOREK, K., ŁUKASZEWICZ, J., PERKAL, J., STEINHAUS, H., and ZUBRZYCKI, S. (1951b). Taksonomia Wrocławska, *Przegl. antrop.*, Vol. 17, pp. 93-207 (in Polish with English summary).

GOWER, J. C., and ROSS, G. J. S. (1969). Minimum spanning trees and single-linkage cluster analysis, *Appl. Statist.*, Vol. 18, pp. 54-64.

JARDINE, N., and SIBSON, R. (1971). *Mathematical Taxonomy*, J. Wiley and Sons Ltd., London and New York.
LANCE, G. N., and WILLIAMS, W. T. (1967). A general theory of classificatory sorting strategies, I. Hierarchical Systems, *The Computer Journal*, Vol. 9, pp. 373-380.
MCQUITTY, L. L. (1957). Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies, *Educ. Psychol. Measmt.*, Vol. 17, pp. 207-222.
SNEATH, P. H. A. (1957). The application of computers to taxonomy, *J. gen. Microbiol.* Vol. 17, pp. 201-226.
VAN RIJSBERGEN, C. J. (1970). A fast hierarchic clustering algorithm. *The Computer Journal*, Vol. 13, pp. 324-326.
WISHART, D. (1969). An algorithm for hierarchical classifications, *Biometrics*, Vol. 25, pp. 165-170.

Book review

Understanding Natural Language, by Terry Winograd, 1972; 195 pages. (*Edinburgh University Press*, £4·00)

This is a reprint in book form of an article that recently filled an entire issue of the journal *Cognitive Psychology*.
Mr Winograd is to be congratulated on a most impressive piece of work. He has an imaginary robot called SHRDLU (I did not find any explanation of this name) which operates on a 'world' consisting of five cuboids of various shapes, colours and sizes, three pyramids and a box, all sitting on a table top. This 'world' does not in fact exist, but can be seen on a television screen. The robot has an arm that can lift these objects, move them elsewhere within the limits of the table top, and set them down again.

The robot can be asked questions, and be given instructions to perform removal and building operations. The book includes a fairly long example to demonstrate the sort of conversation and operations that are possible. While this example looks remarkable, one is not told what one would really like to know, namely

1. are all the author's conversations with the machine as good as this, or was the best one picked for the book?
2. what happens when someone other than the author gives the instructions?
3. what happens if the user, while using correct English, is deliberately perverse in trying to fool the machine?

The discussion of disentangling the syntax of English in general, and also trying to take the meaning into account within the limited world of SHRDLU's experience, is detailed and thoughtful. Yet many questions and difficulties arise that the book does not discuss at all.

Two examples must suffice:

In a section on 'Analysis of Word Endings' it is shown how, given a word that is not in the dictionary, it may be modified to try for a more basic word. If you use the word 'babies' it will correctly try 'baby', but the flow-diagram given will also try 'ty' if 'ties' is not in the dictionary, without thinking of trying 'tie'.

In describing the definition facility it is said that if we say 'A "marb" is a red block which is behind a box', the system recognises that we are defining a new word .... If we then talk about 'two big marbs', the system will build a description exactly like the one for 'two big red blocks which are behind a box'.

This seems to lead us to the situation that if we define a train as 'an engine pulling a set of coaches' then two long trains must be 'two long engines pulling a set of coaches'.

But I do not wish to be too critical in face of such a fine effort. I admire not only the programming, but also the excellent work that has gone into producing such an informative and readable book. What a pity that it should have been given a front cover of so juvenile an appearance.

I. D. HILL (London)

[Note: SHRDLU is the top line of characters on a linotype machine, corresponding to QWERTYUIOP on a typewriter.

Book Review Editor]

An efficient algorithm for a complete link method

D. Defays

Service de Mathématiques appliquées à la Psychologie, Université de Liège au Sart-Tilman, Par 4000 Liège 1, Belgium

An improved algorithm for a complete linkage clustering is discussed. The algorithm is based, like the algorithm for the single link cluster method (Slink) presented by Sibson (1973), on a compact representation of a dendrogram: the pointer representation. This approach offers economy in computation. The algorithm is easily programmable.

(Received May 1976)

1. Introduction

Two of the well-known methods of cluster analysis are the single and complete linkage clustering. The first one was developed by Florek *et al.* (1951), Sneath (1957), and Johnson (1971). An optimally efficient algorithm proposed by Sibson (1973) made its application feasible for a number of OTU's well into the range 10^3 to 10^4. To avoid the extremes of this first method—the well-known 'chaining' effect—it may be necessary to apply on the same set of data alternative hierarchic methods. We showed (Defays, 1975) in a fuzzy sets context that complete linkage clustering, developed by Lance and Williams (1967), and Johnson (1967), among others, generates one or some of the minimal ultrametric dissimilarities superior to the initial dis-similarity. The present paper provides an efficient algorithm for carrying out one of the minimal superior ultrametric dissimilarities. Like Sibson's algorithm, Slink, it enables a complete link cluster analysis to be applied on an unprecedented scale. This method is dependent on the labelling of the objects. Modifications of the labelling permit us to obtain different minimal superior ultrametric dissimilarities.

2. Notation, terminology and preliminary definitions

A fuzzy relation $\mathbf{R}$ is defined as a fuzzy collection of ordered pairs. If $X = \{i | i = 1, \ldots, N\}$, a fuzzy relation on X is characterised by a membership function $R(.,.)$ which associates with each pair (i, j) its 'grade of membership', or in this case the dissimilarity between i and j, $R(i, j) \in [0, \infty]$. In this paper, we consider fuzzy relations $\mathbf{R}$ satisfying the two conditions:

$$\forall i \in X,\ R(i, i) = 0 \text{ (reflexivity)} ,$$

$$\forall i, j \in X,\ R(i, j) = R(j, i) \text{ (symmetry)} .$$

If $\mathbf{R}$ and $\mathbf{Q}$ are defined on X, the min-max composition of $\mathbf{R}$ and $\mathbf{Q}$ is denoted by $\mathbf{R} \circ \mathbf{Q}$ and is defined by

$$R \circ Q(i, j) = \wedge \{(Q(i, k) \vee R(k, j)) | k \in X\},\ i \in X,\ j \in X .$$

The r-fold composition $\mathbf{R} \circ \mathbf{R} \circ \mathbf{R} \ldots \circ \mathbf{R}$ is denoted by $\mathbf{R}^r$. $\mathbf{R}$ is transitive if $\mathbf{R}^2 \supset \mathbf{R}$. We shall call an ultrametric relation, a fuzzy relation which is reflexive, symmetric and transitive. Note that the membership function of an ultrametric relation is an ultrametric dissimilarity. We showed (Defays, 1975) that if $\mathbf{R}$ is a fuzzy reflexive symmetric relation its transitive closure $\overline{\mathbf{R}} = \mathbf{R}^{N-1}$ may be obtained by a single linkage clustering and that complete linkage clustering gives one (or some) minimal ultrametric relation (MUR) superior to $\mathbf{R}$.

Like Sibson, we define a pointer representation as a pair (π, λ) of functions (π: 1, 2, . . ., $N \to$ 1, 2, . . ., N and λ: 1, 2, . . ., $N \to [0. \infty]$) which satisfies the following conditions:

$$\pi(N) = N .$$
$$\lambda(N) = \infty .$$
$$\forall i < N,\ i < \pi(i) .$$
$$\forall i < N,\ \lambda(i) < \lambda(\pi(i)) .$$

This is the definition given by R. Sibson showed, in a slightly different context, that there is a natural 1-1 correspondence between pointer representations and ultrametric relations. Suppose first that $\mathbf{L}$ is an ultrametric relation on X. Define π, λ by

$$\pi(N) = N;$$
$$\lambda(N) = \infty;$$

and for $i < N$,

$$\lambda(i) = \wedge \{L(i, j) | j > i\};$$
$$\pi(i) = \vee \{j | L(i, j) = \lambda(i)\};$$

(π, λ) is called the pointer representation of $\mathbf{L}$. Reciprocally, suppose (π, λ) is a pointer representation. Define $\mathbf{R}$ by its membership function:

$$\begin{aligned} R(i, j) &= \lambda(i) \text{ if } j = \pi(i) > i; \\ &= \lambda(j) \text{ if } i = \pi(j) > j; \\ &= 0 \text{ if } i = j; \\ &= \infty \text{ otherwise} . \end{aligned}$$

$\mathbf{L} = \overline{\mathbf{R}}$ is an ultrametric relation associated with (π, λ). It may be shown that these two transformations are mutually inverse.

3. Algorithm

The problem is to find one of the MUR $\mathbf{L}$ superior to a reflexive symmetric fuzzy relation $\mathbf{R}$. $\overline{\mathbf{R}}$ and $\mathbf{L}$ will then be two extreme clusterings of X. The interest of $\mathbf{L}$ is to shade the results obtained by the single linkage clustering. We shall note $\mathbf{R}_n$ the restriction of $\mathbf{R}$ to the first n OTU's $\{1, 2, \ldots, n\}$ of X. If $\mathbf{L}_n$ is a MUR superior to $\mathbf{R}_n$, as we shall show it, a MUR $\mathbf{L}_{n+1}$ superior to $\mathbf{R}_{n+1}$ may be easily obtained from $\mathbf{L}_n$. Like in Slink, the reason for considering a pointer representation is that it can be updated on the inclusion of a new OTU in an efficient way. Quantities defined on the first n elements will be given subscript n. So, we shall note (π_n, λ_n) as the pointer representation of a MUR $\mathbf{L}_n$ superior to $\mathbf{R}_n$. The present paper gives a method to generate from (π_n, λ_n) the pointer representation $(\pi_{n+1}, \lambda_{n+1})$ of a MUR $\mathbf{L}_{n+1}$ superior to $\mathbf{R}_{n+1}$.

For given n we define $\mu_n(i)$ recursively on i:

$$\mu_n(i) = \vee \{R(i, n+1), \mu_n(j) | j : \pi_n(j) = i,\ \lambda_n(j) < \mu_n(j)\}$$

and then, $v_n(n - i)$ which, when unset, will be noted $*$,

$$v_n(n - i) = \mu_n(n - i)$$

if

$$\lambda_n(n - i) \geqslant \vee \{\mu_n(n - i), \mu_n(\pi_n(n - i))\}$$

and if $v_n(\pi_n(n - i)) \neq *$,

$$v_n(n - i) = * \text{ otherwise} .$$

If $a = \vee \{i | \forall j,\ v_n(j) \geqslant v_n(i) \text{ or } v_n(j) = *\}$, we may define (π, λ) which we shall prove to be the pointer representation of a MUR $\mathbf{L}_{n+1}$ superior to $\mathbf{R}_{n+1}$ as follows:

$$\pi(n + 1) = n + 1 ;$$
$$\lambda(n + 1) = \infty ;$$

$$\pi(a) = n + 1 ;$$
$$\lambda(a) = v_n(a) ;$$
$$\forall k \geqslant 1,\ \pi(\pi_n^k(a)) = n + 1 ;$$
$$\forall k \geqslant 1,\ \lambda(\pi_n^k(a)) = \lambda_n(\pi^{k-1}_n(a))$$

if we note $\pi_n^0(a) = a$ and if $\pi^{k-1}_n(a) < n$
and recursively on i, if for all $k \geqslant 0$, $i \neq \pi_n^k(a)$:

$$\lambda(i) = \lambda_n(i) ;$$
$$\pi(i) = \pi_n(i)$$

except that if $\pi(\pi_n(i)) = n + 1$ and $\lambda_n(i) \geqslant \lambda(\pi_n(i))$, $\pi(i) = n + 1$.

Theorem
(π, λ) is the pointer representation of a MUR $\mathbf{L}_{n+1}$ superior to $\mathbf{R}_{n+1}$.

Proof
Let us show first that (π, λ) is a pointer representation. We have defined $\pi(n + 1) = n + 1$ and $\lambda(n + 1) = 0$. Since $\pi(i) \geqslant \pi_n(i)$, for $i < n$ we have $i < \pi_n(i) \leqslant \pi(i)$. If $i = n$, since $n = \pi_n^k(a)$ for some $k \geqslant 0$, we have $n < \pi(n) = n + 1$. In all cases, if $i < n + 1$, we have $i < \pi(i)$. If $\pi(i) \neq n + 1$, we have $\lambda(i) = \lambda_n(i)$. Then, if $\pi_n(i) < n$, we have $\lambda(i) = \lambda_n(i) < \lambda_n(\pi_n(i))$. If $\lambda(\pi(i)) = \lambda_n(\pi_n(i))$, then $\lambda(i) < \lambda(\pi(i))$. If $\lambda(\pi(i)) = \lambda(\pi_n(i)) \neq \lambda_n(\pi_n(i))$, then $\pi(\pi_n(i)) = n + 1$. But, since $\pi(i) \neq n + 1$, we have $\lambda_n(i) < \lambda(\pi_n(i)) = \lambda(\pi(i))$. If $\pi_n(i) = n$, then $\pi(\pi_n(i)) = n + 1$. But, since $\pi(i) \neq n + 1$, we have $\lambda(i) = \lambda_n(i) < \lambda(\pi_n(i)) = \lambda(\pi(i))$. In all cases, $\pi(i) \neq n + 1$ implies $\lambda(i) < \lambda(\pi(i))$ and (π, λ) is a pointer representation.

Let us show now that (π, λ) is the pointer representation of a relation superior to $\mathbf{R}_{n+1}$. We shall note $\mathbf{L}$ the ultrametric relation associated with (π, λ). First of all, it is easy to prove that $L(i, j) \leqslant L_n(i, j)$ for $i, j \leqslant n$. For, if $\pi(i) \neq \pi_n(i)$, and if $i = a$ or $i = \pi_n^k(a)$ for some $k \geqslant 1$, we have $L(i, n + 1) \leqslant \lambda(i) \leqslant \lambda_n(i)$ and $L(\pi_n(i), n + 1) \leqslant \lambda(\pi_n(i)) = \lambda_n(i)$. In virtue of transitivity, we must have $L(i, \pi_n(i)) \leqslant \lambda_n(i)$; if $\pi(i) \neq \pi_n(i)$, and $i \neq a$ and $i \neq \pi_n^k(a)$ for all $k \geqslant 1$, we have $\pi_n(i) = a$ or $\pi_n(i) = \pi_n^k(a)$ for some $k \geqslant 1$ and $\lambda_n(i) \geqslant \lambda(\pi_n(i))$. Since $L(i, n + 1) \leqslant \lambda_n(i)$ and $L(\pi_n(i), n + 1) \leqslant \lambda(\pi_n(i)) \leqslant \lambda_n(i)$, in virtue of transitivity, $L(i, \pi_n(i)) \leqslant \lambda_n(i)$. If $\pi(i) = \pi_n(i)$, we have $L(i, \pi_n(i)) \leqslant \lambda_n(i)$. Therefore, we shall have $L(i, j) \leqslant L_n(i, j)$ for all $i, j \leqslant n$. To prove that $\mathbf{L} \supset \mathbf{R}_{n+1}$, it is sufficient to prove that for all $h \in [0, \infty]$, $L^h \supset R_{n+1}^h$ if we note

$$L^h = \{(i, j) | L(i, j) \leqslant h\},\ R_{n+1}^h = \{(i, j) | R_{n+1}(i, j) \leqslant h\} .$$

If $\mathbf{L}$ is an ultrametric relation, Zadeh (1971) has shown that for all $h \in [0, \infty]$, L^h is an equivalence. Let C be a class of the equivalence L^h and let us prove that for all $i, j \in C$, $R(i, j) \leqslant h$. If C is also a class of the equivalence $L_n^h = \{(i, j) | L_n(i, j) \leqslant h\}$, since $\mathbf{L}_n \supset \mathbf{R}_n$ the assertion is established. If C is not a class of L_n^h, since $\mathbf{L}_n$ is a MUR superior to $\mathbf{R}_n$, we have $n + 1 \in C$ and $C - \{n + 1\}$ is a class of L_n^h. The assertion will be established if we prove that for all $i \in C$, $R(i, n + 1) \leqslant h$. For all $i < n + 1$, we define $\sigma(i)$ to be $\lambda(\pi^{k-1}(i))$ if $\pi^{k-1}(i) \neq \pi^k(i) = n + 1$ and we define $\sigma(n + 1) = 0$. By construction of $\mathbf{L}$, $C = \{i | \sigma(i) \leqslant h\}$. This assertion may be easily established. The proof may be found in (Sibson, 1973) and is not given. We suppose $C - \{n + 1\} \neq \phi$. In this case $a \in C$ for, if $i \in C$ and $i \neq n + 1$, we have $\sigma(i) \geqslant \lambda(a)$. Let us show first that if $\pi(j) = n + 1$ and $j \in C$, $R(j, n + 1) \leqslant \mu_n(j) \leqslant h$. If $j = a$, it is true. If $\pi_n(j) = a$ and $\pi(j) = n + 1$, we have $\lambda_n(j) \geqslant \lambda(a)$. If $\mu_n(j) > \lambda_n(j)$, we have $h \geqslant \mu_n(a) = \lambda(a) \geqslant \mu_n(j)$. If $\mu_n(j) \leqslant \lambda_n(j)$, since $\lambda_n(j) = \lambda(j)$ and $\sigma(j) = \lambda(j) \leqslant h$, we have $\mu_n(j) \leqslant h$. In all cases, if $\pi_n(j) = a$ and $\pi(j) = n + 1$, we have $\mu_n(j) \leqslant h$ if $j \in C$. If $j = \pi_n^k(a)$ for some $k \geqslant 1$ and $j \in C$, we have $h \geqslant \sigma(j) = \lambda(j) = \lambda(\pi_n^k(a)) = \lambda_n(\pi^{k-1}_n(a)) \geqslant \mu_n(\pi_n^k(a)) = \mu_n(j)$ for $v_n(\pi^k(a)) \neq *$. If $\pi_n(j) = \pi_n^k(a)$ for some $k \geqslant 1$ and $\lambda_n(j) \geqslant \lambda(\pi_n^k(a))$, we have $\mu_n(j) \leqslant h$ too for if $\mu_n(j) \leqslant \lambda_n(j)$, we have $\mu_n(j) \leqslant \lambda_n(j) = \lambda(j) = \sigma(j) \leqslant h$ and if $\mu_n(j) > \lambda_n(j)$, since $h \geqslant \sigma(j) = \lambda(j) = \lambda_n(j) \geqslant \lambda(\pi_n^k(a))$ we have $\pi_n^k(a) \in C$ and $h \geqslant \mu_n(\pi_n^k(a)) \geqslant \mu_n(j)$. Let us show now that if $\pi(j) \neq \pi^2(j) = n + 1$ and $j \in C$, we have $R(j, n + 1) \leqslant \mu_n(j) \leqslant h$. If $\lambda_n(j) < \mu_n(j)$, recursively we have $h \geqslant \mu_n(\pi(j)) \geqslant \mu_n(j)$; if $\lambda_n(j) \geqslant \mu_n(j)$, since $\pi(j) \neq n + 1$ we have $\pi(j) = \pi_n(j)$ and $\mu_n(j) \leqslant \lambda_n(j) = \lambda(j) < \lambda(\pi(j)) = \sigma(j) \leqslant h$. If $\pi^2(j) \neq \pi^3(j) = n + 1$ and $j \in C$, it can be established as precedently that $h \geqslant \mu_n(j) \geqslant R(j, n + 1)$. Recursively, it can be shown that for all $i \in C$, $h \geqslant \mu_n(i) \geqslant R(i, n + 1)$. Thus we have $\mathbf{L} \supset \mathbf{R}_{n+1}$.

To complete the proof, we must establish that for every ultrametric relation $\mathbf{L}'$ such that $\mathbf{L} \supset \mathbf{L}' \supset \mathbf{R}_{n+1}$ we have $\mathbf{L} = \mathbf{L}'$. Let us suppose $\mathbf{L}'$ to be such an ultrametric relation $(\mathbf{L} \supset \mathbf{L}' \supset \mathbf{R}_{n+1})$ and let us show first that for all $i \leqslant n$, $L'(i, n + 1) \geqslant \mu_n(i)$. If $i = 1$, it is obvious. Suppose it is true for all $i < k \leqslant n$. We shall establish that $L'(k, n + 1) \geqslant \mu_n(k)$. Since $\mathbf{L}' \supset \mathbf{R}_{n+1}$, we have $L'(k, n + 1) \geqslant R(k, n + 1)$. If $\pi_n(j) = k$ and $\mu_n(j) > \lambda_n(j)$, since $L'(j, k) \leqslant L(j, k) \leqslant L_n(j, k) \leqslant \lambda_n(j) < \mu_n(j)$ and since $L'(j, n + 1) \geqslant \mu_n(j)$ for $j < k$, we have in virtue of transitivity of $\mathbf{L}'$, $L'(k, n + 1) \geqslant \mu_n(j)$ and therefore, by construction of $\mu_n(k)$, $\mu_n(k) \leqslant L'(k, n + 1)$. We note (π', λ') the pointer representation of $\mathbf{L}'$. The theorem will be established if we suppose that $\mathbf{L}' \neq \mathbf{L}$ and we show that it induces an absurdity. Since $\mathbf{L}_n$ is a MUT superior to $\mathbf{R}_n$, if $\mathbf{L}' \neq \mathbf{L}$, there exists i with $\lambda'(i) \leqslant \lambda(i)$ and $\pi'(i) = n + 1 \neq \pi(i)$. Since $\mu_n(i) \leqslant L'(i, n + 1) \leqslant \lambda'(i)$, we have $\mu_n(i) \leqslant \lambda(i)$ and for all $k \geqslant 1$, $L'(i, n + 1) \leqslant L'(i, \pi_n^k(i)) \leqslant L_n(i, \pi_n^k(i)) \leqslant \lambda_n(\pi^{k-1}_n(i))$. Thus, in virtue of transitivity of $\mathbf{L}'$, we have $\lambda_n(\pi^{k-1}_n(i)) \geqslant L'(\pi_n^k(i), n + 1) \geqslant \mu_n(\pi_n^k(i))$ and by construction of v_n, $v_n(i) = \mu_n(i)$. An immediate consequence is $v_n(a) < v_n(i) = \mu_n(i) \leqslant \lambda'(i)$ or $v_n(a) = v_n(i) = \mu_n(i) \leqslant \lambda'(i)$ and $a > i$. In $L'^{\lambda'(i)} = \{(r, s) | L'(r, s) \leqslant \lambda'(i)\}$, i and a must be in the same class. Since $\mathbf{L}_n$ is a MUT superior to $\mathbf{R}_n$ and since for all $i, j \leqslant n$, $L'_n(i, j) \geqslant L(i, j) \geqslant L'(i, j)$, i and a must be in the same class of $L'^{\lambda'(i)}$. Since $n + 1$ belongs to that class and $\pi(i) \neq n + 1$, it implies $\lambda(i) < \lambda'(i)$ which is absurd. This completes the proof of our theorem.

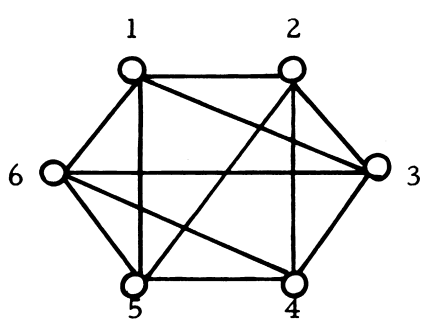


Fig. 1 Relation R

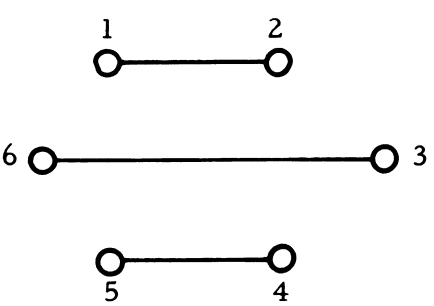


Fig. 2 MUR L superior to R which cannot be obtained with our algorithm

Notice that the choice of $a = \vee \{i | \forall j, v_n(j) \geqslant v_n(i)$ or $v_n(j) = *\}$ is arbitrary. Other choices are possible.

If we start with $\pi_1(1) = 1$ and $\lambda_1(1) = \infty$, then after $N - 1$ steps of the above recursive process, we shall obtain the pointer representation of a MUR superior to **R**. We have noticed at the beginning of this paper that the result will vary with a relabelling of the elements of X.

An interesting question is: is it possible by modifications of the labellings of the OTU's to obtain with our algorithm all the MUR's superior to **R**? Unhappily, this hope is not founded. The very simple example given below proves it. In **Fig. 1,** we link OTU's with dissimilarity 0. The dissimilarity between two not linked OTU's is supposed to be 1. In **Fig. 2,** the relation **L** represented with the same conventions as in Fig. 1, is a MUR superior to the relation **R** of Fig. 1, which obviously cannot be obtained with our algorithm.

If one hopes to obtain a result **L** not too far from the initial relation **R**, a choice of the labels such as

$$\sum_{j=1}^{N} R(1,j) \leqslant \sum_{j=1}^{N} R(2,j) \leqslant \ldots \leqslant \sum_{j=1}^{N} R(N,j)$$

may be judicious.

The results may be presented as in Slink; a conversion of the pointer representation into a packed representation provides readable output.

4. The CLINK algorithm

We are going now to give a statement of the algorithm from the computational point of view. We shall try to give it as similarly as possible to the statement of the Slink algorithm. In fact, to perform our algorithm, the subroutine Slink1 of Slink has only to be modified. As in Slink, three arrays of dimension N are used; we shall denote them by π, $\wedge$, M. Suppose that π, $\wedge$ contain π_n, λ_n. The Clink algorithm will change them into π_{n+1}, λ_{n+1} as follows:

1. Set $\pi(n+1)$ to $n+1$, $\wedge(n+1)$ to ∞.
2. Set $M(i)$ to $R(i, n+1)$ for $i = 1, \ldots, n$.
3. For i increasing from 1 to n
   if $\wedge(i) < M(i)$
   set $M(\pi(i))$ to $\max\{M(\pi(i)), M(i)\}$.
   set $M(i)$ to ∞.
4. Set a to n.
5. For i increasing from 1 to n
   if $\wedge(n-i+1) \geqslant M(\pi(n-i+1))$
   set a to $n-i+1$ if $M(n-i+1) < M(a)$.
   if $\wedge(n-i+1) < M(\pi(n-i+1))$
   set $M(n-i+1)$ to ∞.
6. Set b to $\pi(a)$, c to $\wedge(a)$, $\pi(a)$ to $n+1$ and $\wedge(a)$ to $M(a)$.
7. If $a < n$
   if $b < n$
   set d to $\pi(b)$, e to $\wedge(b)$.
   set $\pi(b)$ to $n+1$, $\wedge(b)$ to c.
   set b to d, c to e.
   go to 7.
   if $b = n$
   set $\pi(b)$ to $n+1$, $\wedge(b)$ to c.
8. For i increasing from 1 to n
   if $\pi(\pi(i)) = n+1$
   set $\pi(i)$ to $n+1$ if $\wedge(i) \geqslant \wedge(\pi(i))$.

Appendix A FORTRAN CLINK program

We only give here a subroutine CLINK which must be inserted in the Slink program in the place of the subroutine Slink1. Note that the calling program for the subroutine CLINK must declare TOP which is not declared for Slink1. The measure Δ_1 of classifiability will have a negative value as the result is an ultrametric relation superior to the initial relation.

```
      SUBROUTINE CLINK(NA,HA,HB,I1,NMXOBJ,TOP)
      DIMENSION NA(NMXOBJ),HA(NMXOBJ),HB(NMXOBJ)
      DO 1 J=1,I1
      NEXT=NA(J)
      IF (HA(J)-HB(J))2,1,1
    2 H=HB(J)
      HB(J)=TOP
      IF (HB(NEXT)-H)3,1,1
    3 HB(NEXT)=H
    1 CONTINUE
      IMAX=I1
      DO 4 JI=1,I1
      J=I1-JI+1
      NEXT=NA(J)
      IF (HA(J)-HB(NEXT))6,5,5
    5 IF (HB(J)-HB(IMAX))7,4,4
    7 IMAX=J
      GO TO 4
    6 HB(J)=TOP
    4 CONTINUE
      I1S=NA(IMAX)
      H1S=HA(IMAX)
      NA(IMAX)=I1+1
      HA(IMAX)=HB(IMAX)
      IF (IMAX-I1)10,11,11
   10 IF (I1S-I1)8,9,9
    8 K=NA(I1S)
      HK=HA(I1S)
      NA(I1S)=I1+1
      HA(I1S)=H1S
      I1S=K
      H1S=HK
      GO TO 10
    9 NA(I1S)=I1+1
      HA(I1S)=H1S
   11 DO 12 J=1,I1
      NEXT=NA(J)
      IF (NA(NEXT)-I1)12,12,13
   13 IF (HA(J)-HA(NEXT))12,14,14
   14 NA(J)=I1+1
   12 CONTINUE
      RETURN
      END
```


References

DEFAYS, D. (1975). Ultrametriques et relations floues, *Bull. Soc. Roy. Sc. de Liège*, No. 1-2, pp. 104-118

SIBSON, R. (1973). An optimally efficient algorithm for the single link cluster method, *The Computer Journal*, Vol. 16, pp. 30-45

ZADEH, L. A. (1971). Similarity relations and fuzzy orderings, *Inf. Sciences*, No. 3, pp. 177-200

Robust Hierarchical Clustering

Maria Florina Balcan
Georgia Institute of Technology
School of Computer Science
ninamf@cc.gatech.edu

Pramod Gupta
Georgia Institute of Technology
School of Computer Science
pramodgupta@gatech.edu

Abstract

One of the most widely used techniques for data clustering is agglomerative clustering. Such algorithms have been long used across many different fields ranging from computational biology to social sciences to computer vision in part because their output is easy to interpret. Unfortunately, it is well known, however, that many of the classic agglomerative clustering algorithms are *not* robust to noise [14]. In this paper we propose and analyze a new robust algorithm for bottom-up agglomerative clustering. We show that our algorithm can be used to cluster accurately in cases where the data satisfies a number of natural properties and where the traditional agglomerative algorithms fail. We also show how to adapt our algorithm to the inductive setting where our given data is only a small random sample of the entire data set.

1 Introduction

Many data mining and machine learning applications ranging from computer vision to biology problems have recently faced an explosion of data. As a consequence it has become increasingly important to develop effective, accurate, robust to noise, fast, and general clustering algorithms, accessible to developers and researchers in a diverse range of areas.

One of the oldest and most commonly used methods for clustering data, widely used in many scientific applications, is hierarchical clustering [5, 6, 9, 8, 11, 12, 14, 10, 13]. In hierarchical clustering the goal is not to find a single partitioning of the data, but a hierarchy (generally represented by a tree) of partitions which may reveal interesting structure in the data at multiple levels of granularity. The most widely used hierarchical methods are the agglomerative clustering techniques; most of these techniques start with a separate cluster for each point and then progressively merge the two closest clusters until only a single cluster remains. In all cases, we assume that we have a measure of similarity between pairs of objects, but the different schemes are distinguished by how they convert this into a measure of similarity between two clusters. For example, in single linkage the similarity between two clusters is the maximum similarity between points in these two different clusters. In complete linkage, the similarity between two clusters is the minimum similarity between points in these two different clusters. Average linkage has various variants, for example, a common one defines the similarity between two clusters as the average similarity between points in these two different clusters [8, 12].

Such algorithms have been used in a wide range of application domains ranging from biology applications to social sciences to computer vision applications mainly because they are quite fast and the output is quite easy to interpret. It is well known, however, that one of the main limitations of the agglomerative clustering algorithms is that they are *not* robust to noise [14]. In this paper we propose and analyze a robust algorithm for bottom-up agglomerative clustering. We show that our algorithm satisfies formal robustness guarantees and it will be successful in many cases where the traditional agglomerative algorithms fail.

In order to formally analyze correctness of our algorithm we use the framework introduced by Balcan et. al [2]. In this framework, we assume there is some target clustering (much like a k-class target function in the multi-class learning setting) and we say that an algorithm correctly clusters data satisfying property P if on any data set having property P, the algorithm produces a tree such that the target is some pruning of the tree. For example if all points are more similar to points

in their own target cluster than to points in any other cluster (this is called the strict separation property), then any of the standard agglomerative algorithms will succeed. See Figure 1. However, with just tiny bit of noise, for example if each point has even just one point from a different cluster that it is similar too, then the standard algorithms will all fail (we elaborate on this in Section 2.2). See Figure 2. This brings up the question: is it possible to design an agglomerative algorithm that is robust to these types of situations and more generally can tolerate a substantial degree of noise? The contribution of our paper is to provide a strong positive answer to this question; we develop a robust, linkage based algorithm that will succeed in many interesting cases where standard agglomerative algorithms will fail. At a high level, our new algorithm is robust to noise in two different and important ways. First, it uses more global information for creating an interesting starting point for a linkage procedure (a set of not too small, but also not too large blobs that are mostly "pure"); second, it uses a robust linkage procedure for merging large enough blobs.

1.1 Our Results

We present a new and robust algorithm for agglomerative clustering and we show that our algorithm will be successful in many cases where standard agglomerative algorithms will fail.

In particular, we show that if the data satisfies a natural good neighborhood property, then our algorithm can be used to cluster well in the tree model (i.e., to output a hierarchy such that the target clustering is a pruning of that hierarchy). The good neighborhood property roughly says after a small number of malicious points have been removed, for the remaining points, most of their nearest neighbors are from their target cluster. We also show how to adapt our algorithm to the inductive setting, where our given data is only a small random sample of the entire data set. Based on such a sample, our algorithm outputs an implicit hierarchy of clusterings of the full domain, that is evaluated with respect to the underlying distribution. A nice property of the condition and of the algorithm we analyze is that they are insensitive to any monotone transformation of the similarities.

It is worth noting that the good neighborhood property is much broader than the ν-strict separation property, a generalization of the simple strict separation property discussed above, requiring that after a small number of outliers have been removed all points are strictly more similar to points in their own cluster than to points in other clusters. Balcan et. al [2] also analyzed the ν-strict separation condition and provided an algorithm for producing a hierarchy with the desired property, but via a much more computationally expensive (non-agglomerative) algorithm. Our algorithm is simpler, substantially faster, and much more generally applicable compared to the algorithm in [2] specifically designed for ν-strict separation.

1.2 Related Work

In agglomerative hierarchical clustering [9, 8, 11, 12] the goal is not to find a single partitioning of the data, but a hierarchy (generally represented by a tree) of partitionings which may reveal interesting structure in the data at multiple levels of granularity. Traditionally, only clusterings at a certain level are considered, but as we argue in Section 2 it is more desirable to consider all the prunings of the tree, since this way we can then handle much more general situations. As mentioned above, it is well known that standard agglomerative hierarchical clustering techniques are not tolerant to noise.

2 Definitions. A Formal Setup

We consider a clustering problem (S, ℓ) specified as follows. Assume we have a data set S of n objects. Each $x \in S$ has some (unknown) "ground-truth" label $\ell(x)$ in $Y = \{1, \ldots, k\}$, where we will think of k as much smaller than n. We let $C_i = \{x \in S : \ell(x) = i\}$ denote the set of points of label i (which could be empty), and denote the target clustering as $\mathcal{C} = \{C_1, \ldots, C_k\}$. Given another proposed clustering h, $h : S \to Y$, we define the error of h with respect to the target clustering to be the the fraction of points on which h and $\mathcal{C}$ disagree under the optimal matching of clusters in h to clusters in $\mathcal{C}$; i.e.,

$$err(h) = \min_{\sigma \in \mathcal{S}_k} \left[\Pr_{x \in S} [\sigma(h(x)) \neq \ell(x)] \right],$$

where $\mathcal{S}_k$ is the set of all permutations on $\{1, \ldots, k\}$. Equivalently, the error of a clustering $\mathcal{C}' = \{C'_1, \ldots, C'_k\}$ can be expressed as

$$\min_{\sigma \in \mathcal{S}_k} \frac{1}{n} \sum_i |C_i - C'_{\sigma(i)}|.$$

We will be considering clustering algorithms whose only access to their data is via a pairwise similarity function $\mathcal{K}(x, x')$ that given two examples outputs a number in the range $[-1, 1]$. We will say that $\mathcal{K}$ is a symmetric similarity function if $\mathcal{K}(x, x') = \mathcal{K}(x', x)$ for all x, x'. In this paper we

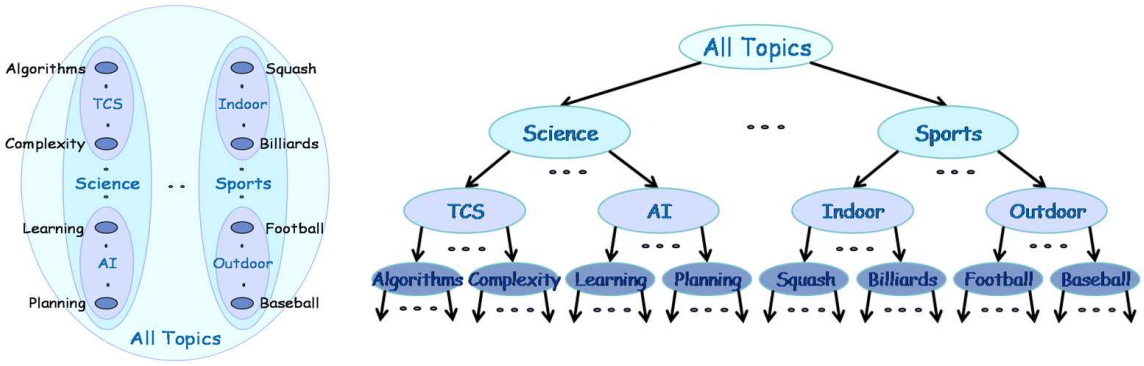


Figure 1: Consider a document clustering problem. Assume that data lies in multiple regions Algorithms, Complexity, Learning, Planning, Squash, Billiards, Football, Baseball. Suppose that $\mathcal{K}(x,y) = 0.999$ if x and y belong to the same inner region; $\mathcal{K}(x,y) = 3/4$ if $x \in$ Algorithms and $y \in$ Complexity, or if $x \in$ Learning and $y \in$ Planning, or if $x \in$ Squash and $y \in$ Billiards, or if $x \in$ Football and $y \in$ Baseball; $\mathcal{K}(x,y) = 1/2$ if x is in (Algorithms or Complexity) and y is in (Learning or Planning), or if x is in (Squash or Billiards) and y is in (Football or Baseball); define $\mathcal{K}(x,y) = 0$ otherwise. Both clusterings $\{$Algorithms $\cup$ Complexity $\cup$ Learning $\cup$ Planning, Squash $\cup$ Billiards, Football$\cup$Baseball$\}$ and $\{$Algorithms$\cup$Complexity, Learning$\cup$Planning, Squash$\cup$Billiards$\cup$ Football $\cup$ Baseball$\}$ satisfy the strict separation property.

assume that the similarity function $\mathcal{K}$ is symmetric. For $A \subseteq S$, we denote by n_A the number of points in A.

Our goal is to produce a hierarchical clustering that contains a pruning that is close to the target clustering. Formally, the goal of the algorithm is to produce a hierarchical clustering: that is, a tree on subsets such that the root is the set S, and the children of any node S' in the tree form a partition of S'. The requirement is that there must exist a *pruning* h of the tree (not necessarily using nodes all at the same level) that has error at most ϵ. Balcan et. al [2] have shown that this type of output is necessary in order to be able to analyze non-trivial properties of the similarity function. For example, even if the similarity function satisfies the requirement that all points are more similar to all points in their own cluster than to any point in any other cluster (this is called the strict separation property) and even if we are told the number of clusters, there can still be multiple different clusterings that satisfy the property. In particular, one can show examples of similarity functions and two significantly different clusterings of the data satisfying the strict separation property. See Figure 1 for an example. However, under the strict separation property, there is a single hierarchical decomposition such that any consistent clustering is a pruning of this tree. This motivates clustering in the tree model and this is the model we consider in this work as well.

Given a similarity function satisfying the strict separation property (see Figure 1 for an example), we can efficiently construct a tree such that the ground-truth clustering is a pruning of this tree [2]. Moreover, almost any of the standard linkage based algorithms (*e.g.*, single linkage, average linkage, or complete linkage) would work well under this property. However, one can show that if the similarity function slightly deviates from the strict separation condition, then all the standard agglomerative algorithms will fail (we elaborate on this in section 2.2). In this context, the main question we address in this work is: Can we develop other more robust, linkage based algorithms that will succeed under more realistic and yet natural conditions on the similarity function?

Note: Note that strict separation does not guarantee that all the cutoffs for different points x are the same, so single linkage would not necessarily have the right clustering if just stopped once it has k clusters; however the target clustering will provably be a pruning of the final single linkage tree; this is why we define success based on prunings.

2.1 Properties of the similarity function

We describe here some natural properties of the similarity functions that we analyze in this paper. We start with a noisy version of the simple strict separation property (mentioned above) which was introduced in [2] and we then define an interesting and natural generalization of it.

Property 1 *The similarity function $\mathcal{K}$ satisfies* **ν-strict separation** *for the clustering problem (S, ℓ) if for some $S' \subseteq S$ of size $(1-\nu)n$, $\mathcal{K}$ satisfies strict separation for (S', ℓ). That is, for all $x, x', x'' \in S'$ with $x' \in C(x)$ and $x'' \notin C(x)$ we have $\mathcal{K}(x, x') > \mathcal{K}(x, x'')$.*

So, in other words we require that the strict separation is satisfied after a number of bad points have been removed. A somewhat different condition is to allow each point to have some bad immediate neighbors as long as most of its immediate neighbors are good. Formally:

Property 2 *The similarity function $\mathcal{K}$ satisfies* **α-good neighborhood** *property for the clustering problem (S, ℓ) if for all points x we have that all but αn out of their $n_{C(x)}$ nearest neighbors belong to the cluster $C(x)$.*[1]

Note that α-good neighborhood property is different from the ν-strict separation property. For the ν-strict separation property we can have up to νn bad points that can misbehave; in particular, these νn bad points can have similarity 1 to *all* the points in S; however, once we remove these points the remaining points are more similar to points in their own cluster than to points in other cluster. On the other hand, for the α-good neighborhood property we require that for all points x all but αn out of their $n_{C(x)}$ nearest neighbors belong to the cluster $C(x)$. (So we cannot have a point that has similarity 1 to all the points in S.) Note however that different points might misbehave on different αn neighbors. We can also consider a property that generalizes both the ν-strict separation property and the α-good neighborhood. Specifically:

Property 3 *The similarity function $\mathcal{K}$ satisfies* **(α, ν)-good neighborhood** *property for the clustering problem (S, ℓ) if for some $S' \subseteq S$ of size $(1-\nu)n$, $\mathcal{K}$ satisfies α-good neighborhood property for (S', ℓ). That is, for all for all points $x \in S'$ we have that all but αn out of their $n_{C(x) \cap S'}$ nearest neighbors in S' belong to the cluster $C(x)$.*

It is easy to see that:

Fact 1 *If the similarity function $\mathcal{K}$ satisfies the α-good neighborhood property for the clustering problem (S, ℓ), then $\mathcal{K}$ also satisfies the $(\alpha, 0)$-good neighborhood property for the clustering problem (S, ℓ).*

Fact 2 *If the similarity function $\mathcal{K}$ satisfies the ν-strict separation property for the clustering problem (S, ℓ), then $\mathcal{K}$ also satisfies the $(0, \nu)$-good neighborhood property for the clustering problem (S, ℓ).*

Balcan et. al [2] have shown that if $\mathcal{K}$ satisfies the strict separation property with respect to the target clustering, then as long as the smallest target cluster has size $5\nu n$, one can in polynomial time construct a hierarchy with the guarantee that the ground-truth is ν-close to a pruning of the hierarchy. Unfortunately the algorithm presented in [2] is computationally very expensive: it first generate a large list of $\Omega(n^2)$ candidate clusters and repeatedly runs pairwise tests in order to laminarize these clusters; its running time is a large unspecified polynomial. Our new robust linkage algorithm can be used to get a simpler and much faster algorithm for clustering accurately under the ν-strict separation property. Additionally, our algorithm is much more general as well.

As shown in [2], the $(2, \epsilon)$ BBG-condition for k-median implies the ν-strict separation condition [1], for $\nu = 5\epsilon$. One can show a similar result for the (ν, c, ϵ)-condition for k-median introduced by [3], and so the condition we analyze is strictly more general than these conditions.

As we show below, if the data satisfies the good neighborhood property, then most of the standard linkage based algorithms will fail. The contribution of our paper is to develop a robust, linkage based algorithm that will succeed under these natural conditions.

2.2 Standard linkage based algorithms are not robust

We can show an example where simple linkage based algorithm would perform very badly, but where our algorithm would work well. In particular, if we slightly modify the example in Figure 1, by adding a little bit of noise, to form links of high similarity between points in different inner blobs,

[1] Note that we assume that for any given point we have a canonical order for its neighbors, and so the set of t nearest neighbors of a given point is always well defined.

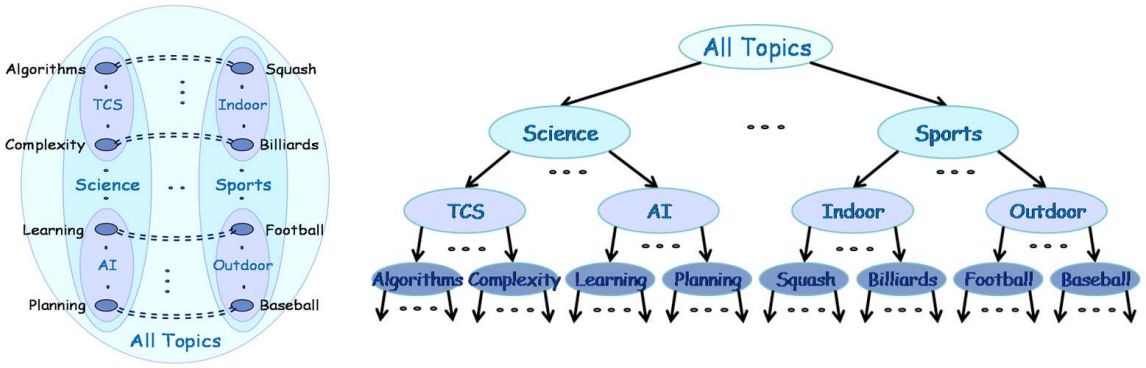


Figure 2: Same as Figure 1 except let us match each point in Algorithms with a point in Squash, each point in Complexity with a point in Billiards, each point in Learning with a point in Football, and each point in Planning with a point in region Baseball. Define the similarity measure to be the same as in Figure 1 except that we let $\mathcal{K}(x, y) = 1$ if x and y are matched. Note that for $\alpha = 1/n$ the similarity function satisfies the α-good neighborhood property with respect to any of the prunings of the tree above. However, single linkage, average linkage, and complete linkage would initially link the matched pairs and produce clusters with very high error with respect to any such clustering.

we can show that many of the classic linkage based algorithms will perform poorly[2]. See Figure 2 for a precise description of the similarity measure.

In particular, the single linkage algorithm, the average linkage algorithm, and the complete linkage algorithm, would in the first $n/2$ stages merge the matched pairs of points. From that moment on, no matter how they perform, none of the natural and desired clusterings will be even $1/2$ close to any of the prunings of the hierarchy produced. Notice however, that the similarity $\mathcal{K}$ satisfies α-good neighborhood property with respect to any of the desired clusterings (for $\alpha = 1/n$), and that our algorithm will be successful on this instance. The ν-strict separation is not satisfied in this example either, for any constant ν.

3 Robust Hierarchical Clustering

In this section we describe an algorithm that we prove is successful if the data satisfies the good neighborhood property. This procedure has two phases: first, it uses somewhat more global information for creating an interesting starting point for a linkage procedure – a set of not too small, but also not too large blobs that are mostly "pure". In a second phase, it runs robust linkage procedure on this set of blobs. Both steps have to be done with care and we will describe in detail in the following sections both steps of our algorithm. In particular, in section 3.1 we describe the procedure for generating a set of interesting blobs and in section 3.2 we describe the linkage procedure.

Algorithm 1 Robust Agglomerative Hierarchical Clustering

Input: Similarity function $\mathcal{K}$, set of points S, $\nu > 0$, $\alpha > 0$.

1. Run Algorithm 2 with parameters ν, α to generate an interesting list L of blobs that partitions the whole set S.
2. Run the linkage Algorithm 3 on these blobs to get the tree T.

Output: Tree T on subsets of S.

We start with a useful definition.

Definition 1 *For $A \subseteq S$, $B \subseteq S$ we define $\mathcal{K}_{\text{median}}(A, B) = \text{median}\{\mathcal{K}(x, x'); x \in A, x' \in B\}$ and we call this the* median similarity *of A to B.*

[2]Since, usually, the similarity function between pairs of objects is constructed based on heuristics, this can easily happen; for example we could have a similarity measure that puts a lot of weight on features such as date or names, and so we could easily have a document about Learning being more similar to a document about Football than to other documents about Learning.

For simplicity we denote $\mathcal{K}_{\text{median}}(\{x\}, B)$ as $\mathcal{K}_{\text{median}}(x, B)$.

Notation: For the rest of this section we assume that the similarity function $\mathcal{K}$ satisfies the (α, ν)–good neighborhood property for the clustering problem (S, ℓ). Let $S' \subseteq S$ be the set of $(1-\nu)n$ points such that $\mathcal{K}$ satisfies α–good neighborhood property with respect to S'. We call the points in S' good points and the points in $S \setminus S'$ bad points. Let $G_i = C_i \cap S'$ be the good set of label i. Let $G = \cup G_i$ the whole set of good points; so $G = S'$. Clearly $|G| \geq n - \nu n$. Denote by $\mathcal{C}_G$ the restriction of the target clustering to the set G.

Note that the following is a useful consequence of the definition.

Claim 2 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (S, ℓ). As long as t is smaller than n_{C_i} for any good point $x \in C_i$ all but at most $(\nu + \alpha)n$ out of its t nearest neighbors lie in its good set $C_i(x) \cap G$.*

3.1 Generating an interesting starting point

Algorithm 2 Generate interesting blobs

Input: similarity function $\mathcal{K}$, set of points S, $\nu > 0$, $\alpha > 0$.

Let the initial threshold $t = 6(\nu + \alpha)n + 1$. Let L be empty. Let $A_S = S$.

Step 1 Construct the graph F_t where we connect points x and y in A_S if they share at least $t - 2(\nu + \alpha)n$ points in common out of their t nearest neighbors with respect to the whole set of points S.

Step 2 Construct the graph H_t by connecting points x and y if they share at least $3(\nu + \alpha)n$ neighbors in the graph F_t.

Step 3 (i) Add to L all the components C of H_t with $|C| \geq 3(\nu + \alpha)n$ and remove from A_S all the points in all these components.
(ii) For all points x in A_S check if $(\nu + \alpha)n$ out of their $5(\nu + \alpha)n$ nearest neighbors are in L. If so, then assign point x to any of the blobs in L of highest median. Remove the points in all these components from A_S.

Step 4 While $|A_s| \geq 3(\nu + \alpha)n$ and $t < n$, increase the critical threshold and go to Step 1.

Step 5 Assign all points x that do not belong to any of the blobs in L arbitrarily to one of the blobs.

Output: A list of blobs which form a partition of S.

We can show the following:

Theorem 3 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (S, ℓ). So long as the smallest target cluster has size greater than $9(\nu + \alpha)n$, then we can use Algorithm 2 to create a list L of blobs each of size at least $3(\nu + \alpha)n$ such that:*

- *The blobs in L form a partition of S.*
- *Each blob in the list L contains good points from only one good set; i.e., for any $C \in L$, $C \cap G \subseteq G_i$ for some $i \leq k$.*

Proof: In the following we denote by n_{C_i} the number of points in the target cluster i. Without loss of generality assume that $n_{C_1} \leq n_{C_2} \leq ... \leq n_{C_k}$. We will show by inductions on $i \leq k$ that:

(a) For any $t \leq n_{C_i}$, any blob in the list L only contains good points from a single good set G_i; all blobs have size at least $3(\nu + \alpha)n$.

(b) At the beginning of the iteration $t = n_{C_i} + 1$, any good point $x \in C_j \cap G$, $j \in \{1, 2, \ldots, i\}$ has already been assigned to a blob in the list L that contains points only from $C_j \cap G$ and has more good points than bad points.

These two claims clearly imply that each blob in the list we output contains good points from only one good set. Moreover at $t = n_{C_k}$ all good points have been assigned to one of the blobs in L. Since we assign the remaining points x that do not belong to any of the blobs in L (these can only

be bad points) arbitrarily to one of the blobs, we also get that the blobs in L form a partition of S, as desired.

Claims (a) and (b) are clearly both true initially. We show now that as long as $t \leq n_{C_1}$, the graphs F_t and H_t have the following properties:

(1) No good point in cluster i is connected in F_t to a good point in a different cluster j, for $i, j > 1$, $i \neq j$. Since $\mathcal{K}$ satisfies the (α, ν)-good neighborhood property for the clustering problem (S, ℓ), by Claim 2, we know that as long as t is smaller than n_{C_i} for any good point $x \in C_i$ all but at most $(\nu + \alpha)n$ out of its t nearest neighbors lie in its good set, i.e., $C_i(x) \cap G$; similarly, as long as t is smaller than n_{C_j} for any good point $y \in C_j$, all but at most $(\nu + \alpha)n$ out of its t nearest neighbors lie in its good set, i.e., $|C_j(x) \cap G|$; so it cannot be the case that for $6(\nu+\alpha)n \leq t \leq n_{C_1}$ two good points in two different clusters $i, j \geq 1$ share $t - 2(\nu+\alpha)n$ points in common out of their t nearest neighbors.

(2) No bad point is connected in F_t to both a good point in cluster i and a good point in different cluster j, for $i, j > 1$, $i \neq j$. This again follows from the fact that since $t \leq n_{C_i}$ for all i, for any good point x all but at most $(\nu+\alpha)n$ out of its t nearest neighbors lie in its good set $C_i(x) \cap G$ (by Claim 2); so for a bad point z to share $t - 2(\nu + \alpha)n$ points out of its t nearest neighbors in common with the t nearest neighbors of a good point x in G_i it must be the case that z has $t - 3(\nu + \alpha)n$ points out of its t nearest neighbors in G_i; but that means that there cannot be other good point y in G_j, where $j \neq i$ such that z and y share $t - 2(\nu+\alpha)n$ points among their t nearest neighbors, because we would need to have that $t - 3(\nu + \alpha)n$ of points out of z's t nearest neighbors lie in G_i and $t - 3(\nu+\alpha)n$ of points out of z's t nearest neighbors in G_j; but this cannot happen if $t > 6(\nu + \alpha)n$ since $2(t - 3(\nu + \alpha)n) > t$ for $t > 6(\nu + \alpha)n$.

(3) All the components of H_t of size at least $3(\nu + \alpha)n$ will only contain good points from one cluster. Since in F_t bad points can only connect to one good set, we get that no two good points in the different clusters connect in H_t.

We can use (1), (2), and (3) to argue that as long as $t \leq n_{C_1}$, each blob in L contains good points from at most one target cluster. This is true at the beginning and by (3), for any $t \leq n_{C_1}$, anytime we insert a whole new blob in L in Step 3(i), that blob must contain good points from at most one target cluster. We now argue that this property is never violated as we assign points to blobs already in L based on the median test in Step 3(ii). Note that at all time steps all the blobs in L have size at least $3(\nu + \alpha)n$. Assume that a good point x has more than $(\nu + \alpha)n$ out of its $5(\nu + \alpha)n$ nearest neighbors in S in the list L. By Lemma 5, there must exist a blob in L that contains only good points from $C(x)$. By Lemma 4, if we assign x based on the median test in Step 3(ii), then we will add x to a blob containing good points only from $C(x)$, and so we maintain the invariant that each blob in L contains good points from at most one target cluster.

We now show that at the beginning of the iteration $t = n_{C_1} + 1$, all the good points in C_1 have already been assigned to a blob in the list L that only contains good points from $C_1 \cap G$. There are a few cases. First, if prior to $t = n_{C_1}$ we did not yet extract in the list L a blob with good points from C_1, then it must be the case that all good points in C_1 connect to each other in the graph F_t; so there will be a component of H_t that will contain all good points from C_1 and potentially bad points, but no good points from another target cluster; moreover this $|C_1| \geq 9(\nu + \alpha)n$, this component will be output in Step 3(i). Second, if prior to $t = n_{C_1}$ we did extract some, but still, more than $3(\nu + \alpha)n$ points from the good set G_1 do not belong to blobs in the list L, then more than $3(\nu + \alpha)n$ of good points will connect to each other in F_t, and then in H_t, so we will add one blob to L containing these good points (plus at most νn bad points). Finally, it could be that by the time we reach $t = n_{C_1}$ all but $l < 3(\nu + \alpha)n$ good points in C_1 have been assigned to a blob in the list L that has good points only from C_1. Since $|C_1| \geq 9(\nu + \alpha)n$ we must have assigned at least $9(\nu + \alpha)n - 3(\nu + \alpha) - \nu n \geq 5(\nu + \alpha)n$ good points from C_1 to the list L. This together with the (α, ν)-good neighborhood property implies that the good points in C_1 that do not belong to the list L yet, must have $(\nu + \alpha)n$ out of their $5(\nu + \alpha)n$ nearest neighbors in S in the list L (at most ν out of the $5(\nu + \alpha)n$ nearest neighbors can be bad points, at most αn can be good points from a different cluster, and at most $3(\nu + \alpha)n$ can be good points in C_1 that do not yet belong to L). So we will assign these points to blobs in L based on the median test in Step 3(ii). By Lemma 4, when we assign them based on the median test in Step 3(ii), we will add them to a blob containing good points from C_1 and no good points from other cluster C_j, as desired.

We then iterate the argument on the remaining set A_S. The key point is that for $t \geq n_i$, $i > 1$, once we start analyzing good points in C_{n_i+1} we have that *all* the good points in C_{n_i}, $C_{n_{i-1}}$, ..., C_{n_1} have already been assigned to blobs in L. ■

We prove below two useful lemmas used in the above proof.

Lemma 4 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (S, ℓ). Assume that L is a list of disjoint clusters each of size at least $3(\nu+\alpha)n$. Assume also that each cluster in L intersects at most a good set; i.e., for any C in L, we have $C \cap G \subseteq G_i$ for some i. Consider $x \in G$ such that there exist C in L with $C \cap G \subseteq C(x) \cap G$. Let $\tilde{C}$ be the blob in L of highest median similarity to x. Then $\tilde{C} \cap G \subseteq C(x) \cap G$.*

Proof: Let us fix a good point x. Let C' and C'' be such that $C' \cap G \subseteq C(x) \cap G$ and $C'' \cap G \subseteq C_i \cap G$, for $C_i \neq C(x)$. Since $\mathcal{K}$ is a symmetric similarity function satisfying the (α, ν)-good neighborhood property, by Claim 2, we have that x can be more similar to at most $\nu n + \alpha n$ points in C'' than with any point in $C' \cap G$. Since $|C'| \geq 3(\nu+\alpha)n$ and $|C''| \geq 3(\nu+\alpha)n$ we get that

$$\mathcal{K}_{\text{median}}(x, C') \geq \mathcal{K}_{\text{median}}(x, C'').$$

This then implies that the blob $\tilde{C}$ in L of highest median similarity to x must satisfy $\tilde{C} \cap G \subseteq C(x) \cap G$, as desired. ■

Lemma 5 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (S, ℓ). Assume that L is a list of clusters each of size at least $3(\nu+\alpha)n$. Assume also that each cluster in L intersects at most a good set; i.e., for any C in L, we have $C \cap G \subseteq G_i$ for some i. Consider $x \in G$ such that there is no cluster C in L with $C \cap G \subseteq C(x) \cap G$. Then at most $(\alpha+\nu)n$ of its t nearest neighbors for any $t \leq n_{C(x)}$ can be in L and all the rest are outside.*

Proof: Since $\mathcal{K}$ satisfies the (α, ν)-good neighborhood property, by Claim 2, for all $x \in G$, for all $t \leq n_{C(x)}$ at most $(\alpha+\nu)n$ of its t nearest neighbors are not from $C(x)$. Consider $x \in G$ such that there is no cluster C in L with $C \cap G \subseteq C(x) \cap G$. So, the list L contains no good points from C, which implies that at most $(\alpha+\nu)n$ of its t nearest neighbors for any $t \leq n_{C(x)}$ can be in L. ■

3.2 A robust linkage procedure

Our linkage procedure is given by Algorithm 3.

Algorithm 3 A robust linkage procedure

Input: A list L of blobs; similarity function $\mathcal{K}$ on pairs of points.

- Repeat till only one cluster remains in L:
  (a) Find clusters C, C' in the current list which maximize score(C, C')
  (b) remove C and C' from L merge them into a single cluster and add that cluster to L.
- Let T be the tree with single elements as leaves and internal nodes corresponding to all the merges performed.

Output: Tree T on subsets of S.

We describe in the following the notion of similarity between pairs of blobs used in Algorithm 3.

Definition 6 *Let $L = \{A_1, \ldots, A_l\}$ be a list of disjoint subsets of S. For each i, for each point x in A_i we compute $\mathcal{K}_{\text{median}}(\{x\}, A_j)$, $j \neq i$, sort them in increasing order, and define* $\text{rank}(x, A_j)$ *as the rank of A_j in the order induced by x. We define*

$$\text{rank}(A_i, A_j) = \text{median}_{x \in A_i}[\text{rank}(x, A_j)].$$

For example, if A_{j_1} is the subset of highest median similarity to x out of all A_j, $j \neq i$, then $\text{rank}(x, A_{j_1}) = l$. Similarly, if A_{j_2} is the subset of smallest median similarity to x out of all A_j, $j \neq i$, then $\text{rank}(x, A_{j_2}) = 1$.

Definition 7 *Let $L = \{A_1, \ldots, A_l\}$ be a list of disjoint subsets of S. We define the score between A_i and A_j as*

$$\text{score}(A_i, A_j) = \min[\text{rank}(A_i, A_j), \text{rank}(A_j, A_i)].$$

Note that while the rank$(\cdot,\cdot)$ might be asymmetric, score$(\cdot,\cdot)$ is designed to be symmetric. We now present a useful lemma.

Lemma 8 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α,ν)-good neighborhood property for the clustering problem (S,ℓ). Let L be a list of disjoint clusters, all of size at least $2(\alpha+\nu)n$. Assume that $B\cap G\subseteq G_i$, $B'\cap G\subseteq G_i$, and $(B''\cap G)\cap G_i=\emptyset$. Then we have both*

$$\text{score}(B,B')<\text{score}(B,B'') \qquad \text{and} \qquad \text{score}(B,B')<\text{score}(B',B'').$$

Proof: Let x be a good point. The (α,ν)-good neighborhood property implies that there exists c_x such that at most αn points $z\in G$, $z\notin C(x)$ can have similarity $K(x,z)$ greater or equal to c_x and at most αn points $y\in G\cap C(x)$ can have similarity $K(x,z)$ strictly smaller than c_x. Since each of the blobs has size at least $2(\alpha+\nu)n$ and since each blob contains at most νn bad points, we get that for all blobs B' and B'' such that $B'\cap G\subseteq C(x)\cap G$ and $(B''\cap G)\cap(C(x)\cap G)=\emptyset$ we have

$$\mathcal{K}_{\text{median}}(\{x\},B')>\mathcal{K}_{\text{median}}(\{x\},B'').$$

So a good point x will rank blobs B' s.t. $B'\cap G\subseteq C(x)\cap G$ later than blobs B'' such that $(B''\cap G)\cap(C(x)\cap G)=\emptyset$ in the order it induces. Assume that there are exactly r blobs B in L such that $(B\cap G)\cap(C(x)\cap G)=\emptyset$. Since there are at most νn bad points and each of the blobs has size at least $2(\alpha+\nu)n$, we obtain that for all B, B' in L such that $B\cap G\subseteq C_i\cap G$ and $B'\cap G\subseteq C_i\cap G$, and for all B'' in L with $(B''\cap G)\cap(C_i\cap G)=\emptyset$ we have both

$$\text{rank}(B,B')>r\geq\text{rank}(B,B'') \quad \text{and} \quad \text{rank}(B',B)\geq r\geq\text{rank}(B',B'').$$

This then implies that

$$\text{score}(B,B')=\text{score}(B',B)=\min[\text{rank}(B,B'),\text{rank}(B',B)]>r.$$

Similarly,

$$\text{score}(B,B'')=\text{score}(B'',B)=\min[\text{rank}(B,B''),\text{rank}(B'',B)]>r.$$

Finally, we have

$$\text{score}(B',B'')=\text{score}(B'',B')=\min[\text{rank}(B',B''),\text{rank}(B'',B')]\leq r.$$

These imply:

$$\text{score}(B,B')>\text{score}(B,B'') \qquad \text{and} \qquad \text{score}(B,B')>\text{score}(B',B''),$$

as desired. ■

We now show that is the similarity function we have satisfies the good neighborhood property, given a good starting point, Algorithm 3 will be successful in outputting a good hierarchy.

Theorem 9 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α,ν)-good neighborhood property for the clustering problem (S,ℓ). Assume that L is a list of clusters each of size at least $3(\nu+\alpha)n$ that partition the entire set of points. Assume also that each cluster in L intersects at most a good set; i.e., for any C in L, we have $C\cap G\subseteq G_i$ for some i. Then Algorithm 3 constructs a binary tree such that the ground-truth clustering is ν-close to a pruning of this tree.*

Proof: First note that at each moment the list L of clusters is a partition of the whole dataset and that all clusters in L have size at least $3(\nu+\alpha)n$. We prove by induction that at each time step the list of clusters restricted to G is laminar w.r.t. $\mathcal{C}_G$.

In particular, assume that our current list of clusters restricted to G is laminar with respect to $\mathcal{C}_G$ (which is true at the start). This implies that for each cluster C in our current clustering and each C_r in the ground truth, we have either

$$C\cap G\subseteq G(C_r) \quad \text{or} \quad G(C_r)\subseteq C\cap G \quad \text{or} \quad (C\cap G)\cap G(C_r)=\emptyset.$$

Now, consider a merge of two clusters C and C'. The only way that laminarity could fail to be satisfied after the merge is if for one of the two clusters, say, C', we have that $C'\cap G$ is strictly contained inside $C_{r'}\cap G$, for some ground-truth cluster $C_{r'}$ (so, $(C_{r'}\cap G)\setminus(C'\cap G)\neq\emptyset$, $(C'\cap G)\subset C_{r'}$) and yet $C\cap G$ is disjoint from $C_{r'}\cap G$. But there must exist C'' in the list L such that $(C''\cap G)\subset C_{r'}\setminus(C'\cap G)$, $|C''|\geq 3(\nu+\alpha)n$. By Lemma 8 we know that

$$\text{score}(C',C'')>\text{score}(C',C).$$

However, this contradicts the specification of the algorithm, since by definition it merges the pair C, C' such that score(C',C) is greatest. ■

3.3 The Main Result

Our main result is the following:

Theorem 10 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (S, ℓ) As long as the smallest target cluster has size greater than $9(\nu+\alpha)n$, then we can use Algorithm 1 in order to produce a tree such that the ground-truth clustering is ν-close to a pruning of this tree in $O(n^{\omega+1})$ time, where $O(n^{\omega})$ is the state of the art for matrix multiplication.*

Proof: The correctness follows immediately from Theorems 3 and 9. The running time follows from Lemma 14 in Appendix A. ■

3.4 The Inductive Setting

In this section we consider an *inductive* model in which S is merely a small random subset of points from a much larger abstract instance space X. Based on such a sample, our algorithm outputs a hierarchy over the sample, which also *implicitly* represents a hierarchy of the whole space which is evaluated with respect to the underlying distribution. Let us assume for simplicity that X is finite and that the underlying distribution is uniform over X.

Our goal is to design an algorithm that based on the sample produces a tree of small error with respect to the whole distribution. Formally, we assume that each node in the tree derived over the sample S induces a cluster (a subset of X) which is implicitly represented as a function $f : X \to \{0, 1\}$. For a fixed tree T and a point x, we define $T(x)$ as the subset of nodes in T that contain x (the subset of nodes $f \in T$ with $f(x) = 1$). We say that a tree T has error at most ϵ if $T(X)$ has a pruning $f_1, ..., f_{k'}$ of error at most ϵ.

Algorithm 4 Inductive Robust Agglomerative Hierarchical Clustering

Input: Similarity function $\mathcal{K}$, parameters α, ν, $k \in Z^+$; $n = n(\alpha, \nu, k, \delta)$;

- Pick a set $S = \{x_1, \ldots, x_n\}$ of n random examples from X.
- Run Algorithm 1 with parameters 2α, 2ν on the set S and obtain a tree T on the subsets of S. Let Q be the set of leaves of this tree.
- Associate each node u in T a function f_u (which induces a cluster) specified as follows:
  Consider $x \in X$, and let $q(x) \in Q$ be the leaf given by $\text{argmax}_{q \in Q} \mathcal{K}_{\text{median}}(x, q)$; if u appears on the path from $q(x)$ to the root, then set $f_u(x) = 1$, otherwise set $f_u(x) = 0$.
- Output the tree T.

Let $N = |X|$. For the rest of this section we assume that the similarity function $\mathcal{K}$ satisfies the (α, ν)–good neighborhood property for the clustering problem (X, ℓ). Let $S' \subseteq S$ be the set of $(1-\nu)N$ points such that $\mathcal{K}$ satisfies α–good neighborhood property with respect to S'. We call the points in S' good points and the points in $S \setminus S'$ bad points. Let $G_i = C_i \cap S'$ be the good set of label i. Let $G = \cup G_i$ the whole set of good points; so $G = S'$.

Our main result in this section is the following:

Theorem 11 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (X, ℓ). As long as the smallest target cluster has size greater than $18(\nu + \alpha)N$, then using Algorithm 4 with parameters α, ν, k, and $n = \Theta\left(\frac{1}{\min(\alpha,\nu)} \ln \frac{k}{\delta \cdot \min(\alpha,\nu)}\right)$, we can produce a tree with the property that the ground-truth is $\nu + \delta$-close to a pruning of this tree with probability $1 - \delta$. Moreover, the size of this tree is $O\left(\frac{1}{\min(\alpha,\nu)} \ln \frac{k}{\delta \cdot \min(\alpha,\nu)}\right)$.*

Proof: Note that n is large enough so that with probability at least $1 - \delta/2$ we have that S contains at most $2\nu n$ bad points and $\mathcal{K}$ satisfies the $(2\alpha, 2\nu)$-good neighborhood property with respect to the clustering induced over the sample (by Lemma 12) and each target cluster has at least $9(\nu + \alpha)n$ points in the sample (by Chernoff bounds).

Assume below that this happens. So, by Theorem 10 we get that the tree T induced over the sample has error at most 2ν over the sample. Let L be the list of leaves of T. By Theorem 3, we

know that L forms a partition of S and that each element of L has size at least $6(\nu+\alpha)n$ and it contains good points from only one good set i.e., for any $C \in L$, $C \cap G \subseteq G_i$ for some $i \leq k$. Let us fix a good point x. By Lemma 13, with probability at least $1-\delta^2/2$ at most $2\alpha n$ of the $n_{\tilde{C}_G(x)}$ nearest points to x from $G \cap S$ can be outside $C(x) \cap G$, where $n_{\tilde{C}_G(x)}$ is $|C(x) \cap G \cap S|$. We can show that $\tilde{C}$ be the blob in L of highest median similarity to x satisfies $\tilde{C} \cap G \subseteq C(x) \cap G$. To see this, let C' and C'' in L be such that $C' \cap G \subseteq C(x) \cap G$ and $C'' \cap G \subseteq C_i \cap G$, for $C_i \neq C(x)$. By the above facts we know that x can be more similar to at most $2\nu n + 2\alpha n$ points in C'' than with any point in $C' \cap G$. Since $|C'| \geq 6(\nu+\alpha)n$ and $|C''| \geq 6(\nu+\alpha)n$ we get that

$$\mathcal{K}_{\text{median}}(x, C') \geq \mathcal{K}_{\text{median}}(x, C'').$$

This then implies that the blob $\tilde{C}$ in L of highest median similarity to x must satisfy $\tilde{C} \cap G \subseteq C(x) \cap G$, as desired. So, for any given point x, with probability $1-\delta^2/2$ over the draw of the random sample, the leaf in T of highest median similarity to x has the property that all its good points are from $C(x)$. Since this is true for any x, by Markov inequality, we get that with probability $1-\delta/2$ a $1-\delta$ fraction of the good points connect to a leaf that contain good points from their own cluster only.

Adding back the $\delta/2$ chance of failure due to either $\mathcal{K}$ not satisfying the $(2\alpha, 2\nu)$-good neighborhood property or having more than $2\nu n$ bad points in S, we get that with probability $1-\delta$ the the error rate of the hierarchy implied by T over the whole set X is at most $\nu+\delta$. ■

Lemma 12 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (X, ℓ). If we draw a set S of $n = \Theta\left(\frac{1}{\min(\alpha,\nu)} \ln \frac{1}{\delta \min(\alpha,\nu)}\right)$, then with probability $1-\delta$, S contains at most $2\nu n$ bad points and the similarity function $\mathcal{K}$ satisfies the $(2\alpha, 2\nu)$-good neighborhood property with respect to the target clustering restricted to the sample S.*

Proof: Since $n \geq \frac{3}{\nu} \ln \frac{2}{\delta}$, by Chernoff bounds, we have that with probability $1-\delta/2$ at most $2\nu n$ bad points fall into the sample. Let us fix a good point x in S and let us denote by $n_{\tilde{C}_G(x)}$ the number of points in $C(x) \cap G \cap S$. By Lemma 13 we have that for $n = \Theta\left(\frac{1}{\alpha} \ln \frac{n}{\delta}\right)$, with probability at least $1-\delta/(2n)$ (over the draw of the other points in the sample) we have that all but $2\alpha n$ of the $n_{\tilde{C}_G(x)}$ nearest neighbors of x from $S \cap G$ are points from the set $C(x) \cap G$. By union bound over all points x in S we have that simultaneously for all good points x in S, all but $2\alpha n$ of their $n_{\tilde{C}_G(x)}$ nearest neighbors in $S \cap G$ come from $C(x) \cap G$.

These together imply that if $n = \Theta\left(\frac{1}{\min(\alpha,\nu)} \ln \frac{n}{\delta}\right)$, then with probability $1-\delta$ at most $2\nu n$ bad points fall into the sample and the similarity function $\mathcal{K}$ satisfies the $(2\alpha, 2\nu)$-good neighborhood property with respect to the target clustering restricted to the sample S. Using the inequality $\ln x \leq \alpha x - \ln \alpha - 1$ for $\alpha, x > 0$, we then get the desired result. ■

Lemma 13 *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood property for the clustering problem (X, ℓ). Consider $x \in G$. If we draw a set S of $n = \Theta\left(\frac{1}{\alpha} \ln \frac{1}{\delta}\right)$ random points from X, then with probability at most $1-\delta$ we have that at most $2\alpha n$ of the $n_{\tilde{C}_G(x)}$ nearest points to x from $G \cap S$ can be outside $C(x) \cap G$, where $n_{\tilde{C}_G(x)}$ is $|C(x) \cap G \cap S|$.*

Proof: Let $C_G(x)$ denote $C(x) \cap G$ and let $n_{C_G(x)} = |C(x) \cap G|$. Let us define $NN(x)$ to be the nearest $n_{C_G(x)}$ points to x in G. Since $\mathcal{K}$ satisfies the (α, ν)-good neighborhood property for the clustering problem (X, ℓ) we have:

$$Pr_{z \sim X}[z \in NN(x) \setminus C_G(x)] \leq \alpha.$$

Since $NN(x)$ and $C_G(x)$ have the same size, this is equivalent to the statement:

$$Pr_{z \sim X}[z \in C_G(x) \setminus NN(x)] \leq \alpha.$$

So, by Chernoff bounds applied to both of the above, with probability at most $1-\delta$ we have that at $2\alpha n$ points are in $(NN(x) \setminus C_G(x)) \cap S$ and at most $2\alpha n$ points are in $(C_G(x) \setminus NN(x)) \cap S$.

We now argue that at most $2\alpha n$ of the $n_{\tilde{C}_G(x)}$ nearest points to x in $G \cap S$ can be outside $C(x) \cap G$, where $n_{\tilde{C}_G(x)} = |C(x) \cap G \cap S|$. Let n_1 be the number of points in $(NN(x) \setminus C_G(x)) \cap S$. Let n_2 be the number of points in $(C_G(x) \setminus NN(x)) \cap S$. Let n_3 be the number of points in $(C_G(x) \cap NN(x)) \cap S$. By construction, we have

$$n_{\tilde{C}_G(x)} = n_2 + n_3,$$

and we are given that $n_1, n_2 \leq 2\alpha n$. We now distinguish two cases.

The first case is $n_1 \geq n_2$. In this case we have

$$n_1 + n_3 \geq n_2 + n_3 = n_{\tilde{C}_G(x)}.$$

This implies that the nearest $n_{\tilde{C}_G(x)}$ points to x in $G \cap S$ all lie inside $NN(x)$, since by definition all points inside $NN(x)$ are closer to x than any point in G outside $NN(x)$. Since we are given that at most $n_1 \leq 2\alpha n$ of them can be outside $C_G(x)$, we get that at most $2\alpha n$ of the $n_{\tilde{C}_G(x)}$ nearest neighbors of x are not from $C_G(x)$, as desired.

The second case is $n_1 \leq n_2$. This implies that the nearest $n_{\tilde{C}_G(x)}$ good points to x in the sample include *all* the points in $NN(x)$ in the sample, plus possibly some others too. But this implies in particular that it includes all the n_3 points in $C_G(x) \cap NN(x)$ in the sample. So, it can include at most

$$n_{\tilde{C}_G(x)} - n_3 \leq 2\alpha \cdot n$$

points not in $C_G(x) \cap NN(x)$, and even if all those are not in $C_G(x)$, it is still $\leq 2\alpha n$; so at most $2\alpha n$ of the $n_{\tilde{C}_G(x)}$ nearest neighbors of x are not from $C_G(x)$, as desired.

■

Note: Note that if we are willing to lose a bit in the accuracy the analysis in this section allows us to speed up the algorithm in Theorem 10.

4 Conclusions

In this paper we propose and analyze a new robust algorithm for bottom-up agglomerative clustering. We show that our algorithm can be used to cluster accurately in cases where the data satisfies a number of natural properties and where the traditional agglomerative algorithms fail. We also show how to adapt our algorithm to the inductive setting where our given data is only a small random sample of the entire data set.

It would be interesting to see if our algorithmic approach can be shown to work for other natural properties. For example, it would be particularly interesting to analyze a relaxation of the max stability property in [2] which was shown to be a necessary and sufficient condition for single linkage, or the average stability property which was shown to be a sufficient condition for average linkage.

Acknowledgments: We thank Avrim Blum for numerous useful discussions. This work was supported in part by NSF grant CCF-0953192.

References

[1] M. F. Balcan, A. Blum, and A. Gupta. Approximate clustering without the approximation. In *ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[2] M. F. Balcan, A. Blum, and S. Vempala. A discrimantive framework for clustering via similarity functions. In *40th ACM Symposium on Theory of Computing*, 2008.

[3] M. F. Balcan, H. Roeglin, and S. Teng. Agnostic clustering. In *The 20th International Conference on Algorithmic Learning Theory*, 2009.

[4] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. Endre Tarjan. Linear time bounds for median computations. In *Proceedings of the fourth Annual ACM symposium on Theory of computing*, 1972.

[5] D. Bryant and V. Berry. A structured family of clustering and tree construction methods. *Advances in Applied Mathematics*, 27(4):705–732, 2001.

[6] D. Cheng, R. Kannan, S. Vempala, and G. Wang. A divide-and-merge methodology for clustering. *ACM Trans. Database Syst.*, 31(4):1499–1525, 2006.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the 19-th Annual ACM conference on Theory of computing*, 1987.

[8] S. Dasgupta and P. Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555 – 569, 2005.

[9] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Willey, 2000.

[10] S. Gollapudi, R. Kumar, and D. Sivakumar. Programmable clustering. In *Symposium on Principles of Database Systems*, pages 348 – 354, 2006.

[11] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[12] A. K. Jain, M. N. Murthy, and P. J. Flynn. Data clustering: A review. In *ACM Computing Reviews*, 1999.

[13] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.
[14] M. Narasimhan, N. Jojic, and J. Bilmes. Q-clustering. In *Neural Information Processing Systems*, 2005.

A Run Time Analysis

Lemma 14 *Algorithm 1 has a running time of $O(n^{\omega+1})$, where $O(n^{\omega})$ is the state of the art for matrix multiplication. (The current value of ω is 2.376 [7]).*

Proof: We analyze the running times of Algorithms 2 and 3 separately. For Algorithm 2, we also discuss certain data structures which are utilized throughout the algorithm for speed up.

In the preprocessing step, we construct a list of nearest neighbors for each point in S by sorting $n-1$ other points in decreasing order of similarity. This takes $O(n \log n)$ time for each point and thus the entire preprocessing step costs $O(n^2 \log n)$ time.

Now, think of a directed t-regular graph E_t, where, for each point j in the t nearest neighbors of a point i, there is a directed edge from i to j in E_t. We construct two n x n matrices Adj^E and $Nbrs^E$. Adj^E is the adjacency matrix for the graph E_t and $Nbrs^E = Adj^E \text{x} (Adj^E)^T$. $Nbrs^E_{ij}$ gives the number of common neighbors between i and j in E_t and from this, we can know whether to draw an edge between them in F_t. Constructing $Nbrs^E$ for the first time takes $O(n^{\omega})$ time. Notice, however, we do not require to recompute $Nbrs^E$ from scratch in every iteration over t as the graph E_t is monotonically increasing. In iteration $t+1$, for each point i exactly one new edge is added to another point k in the graph E_t where k is the $(t+1)_{th}$ nearest neighbor of i. If there was already an edge from k to i in E_t, the value $Nbrs^E_{ik}$ (and $Nbrs^E_{ki}$) increases by 1. For every point j that has a directed edge to k, $Nbrs^E_{ij}$ (and $Nbrs^E_{ji}$) increases by 1. This can be computed by comparing the row Adj^E_i of Adj^E with column Adj^E_k of Adj^E. This requires $O(n)$ time for each point i. Since there are n such points the total cost of the iteration is $O(n^2)$. There can be a maximum of $O(n)$ such iterations. Hence, the total cost of updating $Nbrs^E$ over all iterations is $O(n^3)$. Now, let us consider the case when a new blob of size at least $3(\nu+\alpha)n$ is found and added to L. We remove this blob from A_s and correspondingly all the points in the blob from Adj^E and Adj^E. For each such removal we need to recompute Adj^E and $Nbrs^E$. There can be at most $n/3(\nu+\alpha)n = 1/3(\nu+\alpha)$ such iterations. Thus, we take at most $O(n^{\omega}/(\nu+\alpha))$ time. Therefore, the total cost of constructing the graph F_t over all iterations is $O(n^{\omega}(1/(\nu+\alpha) + n^{3-\omega}))$.

Similarly, let us have two n x n matrices Adj^F and $Nbrs^F$, where Adj^F is the adjacency matrix of the undirected graph F_t and $Nbrs^F = Adj^F \text{x} Adj^F$ (notice, F_t is undirected). Note that the same trick does not hold while constructing the graph H_{t+1} from H_t as the graph F_t is not monotonically increasing. e.g. two points x and y which had an edge in F_t as they had exactly $t-2(\nu+\alpha)n$ neighbors in common may not have an edge in F_{t+1} any more as they might not have $t+1-2(\nu+\alpha)n$ neighbors in common. Thus, for each iteration we need to recompute the matrix $Nbrs^F$. Thus, to construct H_t, it takes a total of $O(n^{\omega+1})$ time over all iterations.

In Step 3(i), we can find all the components of H_t in at most $O(|V_{H_t}| + |E_{H_t}|)$ time where $|V_{H_t}| = n$ and $|E_{H_t}| = O(n^2)$. Thus, we can do this in at most $O(n^2)$ time. Now let's look at the Step 3(ii). It is easy to see that we do this Step at most $1/3(\nu+\alpha)$ times, i.e. once for each new blob. For the first stage of this Step, we maintain a set P_i for each point i that is a set of points from its first $5(\nu+\alpha)$ neighbors that are not yet in the list L and a count c_i of the points already present in L. Every time a new blob B is added to L or we get a non-empty set T (construction explained below) after an iteration of Step 3(ii), we check for the common points between P_i and B or T, then remove all such points from P_i and add the number of these points to c_i. Since, there can be at most $O(n)$ such iterations, this step can take $O(n^3)$ time over all iterations for all points. For each point x that makes it to the next stage of Step 3(ii), we compute the median to each blob B present in L. This can be done in $O(|B|)$ time for each blob [4]. For all the blobs, this can be done in $\sum_{B_i \in L} O(|B_i|) \leq O(n)$ time. Once we have the median for all blobs, we can find the highest of these ($\#blobs \leq 1/3(\nu+\alpha)$) medians which requires at most $O(1/(\nu+\alpha))$ time. Notice, that once a point x gets to this stage, it will be added to one of the blobs in L and not be considered again, which means we hit this stage at most $O(n)$ times. Thus, over all iterations, for all points, this stage of Step 3(ii) can be done in $O(n^2)$ time. Notice, we need to do an additional post-processing step. We add all such x to a set T. This set T is compared with P_i (as explained above) at the beginning of the next iteration. The addition of these points to L could result in more points satisfying the first condition of Step 3(ii). Therefore, the time required for Step 3 is $O(n^3)$.

The costliest Steps in the Algorithm 2 is Step 2 and causes the algorithm to have $O(n^{\omega+1})$ running time.

Algorithm 3 has a running time of $O(n/(\nu+\alpha)(n+1/(\nu+\alpha)\log(1/(\nu+\alpha))))$. At each level, we compute the score of each cluster with respect to all other clusters. Firstly, we compute the median similarity of each cluster C_i with respect to a point x in $\mathrm{O}(|C_i|)$ time [4]. This is done for all clusters except $C(x)$. The total running time is $\sum_{C_i \in L: C_i \neq C(x)} O(|C_i|) < O(n)$. Secondly, we compute the rank by sorting these similarity values in ascending order. Since there can be at most $1/3(\nu+\alpha)$ clusters this can be done in $O(1/(\nu+\alpha)\log 1/(\nu+\alpha))$ time. This is done for all points, hence the total cost of computing ranks of all clusters with respect to all points is $O(n(n+1/(\nu+\alpha)\log(1/(\nu+\alpha))))$. Finally, we compute the rank for a cluster C_i with respect to the cluster C_j by taking the median of the ranks given by each point $x \in C_j$ to the cluster C_i and this takes $\mathrm{O}(|C_j|)$ time. This is done by C_j for all other clusters. Thus, the total time taken by C_j to compute ranks for all other clusters is $O(|C_j|/(\nu+\alpha))$. Since we do it for all clusters it requires $\sum_{C_j \in L} O(|C_j|/(\nu+\alpha)) \leq O(n/(\nu+\alpha))$ time. Thus, we can be done with each level in $O(n(n+1/(\nu+\alpha)\log(1/(\nu+\alpha))))$ time. The number of levels is at most the number of clusters which is $O(1/(\nu+\alpha))$. Therefore, the total running time of the algorithm is $O(n/(\nu+\alpha)(n+1/(\nu+\alpha)\log(1/(\nu+\alpha))))$.

For Algorithm 1, the costliest step is Algorithm 2 and thus has a running time of $O(n^{\omega+1})$. ■

Optimal Implementations of UPGMA and Other Common Clustering Algorithms

Ilan Gronau Shlomo Moran

June 28, 2007

Abstract

In this work we consider hierarchical clustering algorithms, such as UPGMA, which follow the closest-pair joining scheme. We survey optimal $O(n^2)$-time implementations of such algorithms which use a 'locally closest' joining scheme, and specify conditions under which this relaxed joining scheme is equivalent to the original one (i.e. 'globally closest').

Key Words: Hierarchical clustering, UPGMA, design of algorithms, input-output specification, computational complexity

1 Introduction

Hierarchical clustering algorithms were already developed in the 1960's. They are used in numerous applications such as pattern recognition, computational biology and data mining. These algorithms deal today with a vast amount of input, as introduced for instance by high throughput experiments in biology [10] and by data mining of the world wide web [12]. Throughout the years, extensive research has been done dealing with efficient implementations for such algorithms [19, 15, 4, 9, 7, 17, 6, 1]. In this paper we focus on a common class of hierarchical clustering algorithms, which we call *Globally Closest Pair* (or GCP) clustering algorithms. Specifically, we discuss several known techniques for implementing such algorithms in asymptotically optimal time.

A general scheme for GCP clustering algorithms is described in Table 1. Such algorithms receive as input a set of elements and a matrix containing all dissimilarities between element-pairs, and return a *hierarchy* of clusters over this set. Initially, a singleton-cluster is defined for every element in the set. Clustering then proceeds with the main loop, in which at each iteration two clusters of **minimal dissimilarity** are selected and replaced by their union. The following iteration then continues with the smaller cluster-set. In each such iteration the dissimilarities between the new cluster and all other clusters need to be determined (Step 3). This is typically done using some *reduction formula* to compute new dissimilarities using old ones.

Different variants of GCP clustering algorithms are determined by the specification of the reduction in Step 3. UPGMA (Unweighted Pair Grouping Method

Input: A dissimilarity matrix D over a set of elements S.
Output: A cluster-hierarchy $\mathcal{H}$ over S.

Initialization: Initialize the cluster-set $\mathcal{C}$ by defining a singleton cluster $C_i = \{i\}$ for every element $i \in S$. Initialize output hierarchy $\mathcal{H} \leftarrow \mathcal{C}$.

Loop: While $|\mathcal{C}| > 1$ do:

1. **Cluster-pair selection:** Select a pair of distinct clusters $\{C_i, C_j\} \subseteq \mathcal{C}$ of minimal dissimilarity under D (ties are broken arbitrarily).
2. **Cluster-pair joining:** Remove C_i, C_j from the cluster set $\mathcal{C}$ and replace them with $C_i \cup C_j$. Add $C_i \cup C_j$ to the hierarchy $\mathcal{H}$.
3. **Reduction:** Calculate the dissimilarity $D(C_k, (C_i \cup C_j))$ for every $C_k \in \mathcal{C}' \setminus \{C_i \cup C_j\}$.

Finalize: Return the hierarchy $\mathcal{H}$.

Table 1: A general scheme for 'globally closest pair' clustering algorithms.

with Arithmetic-mean) is one of the most commonly used variants. It defines the dissimilarity between clusters as their average dissimilarity (hence its name). This is achieved by using the following reduction formula:

$$D(C_k, (C_i \cup C_j)) \leftarrow \frac{|C_i|}{|C_i| + |C_j|} D(C_k, C_i) \; + \; \frac{|C_j|}{|C_i| + |C_j|} D(C_k, C_j) \qquad (1)$$

Other common GCP clustering algorithms, such as WPGMA [20] and the single linkage algorithm [2, 19], are determined by different reduction formulae:

$$\textbf{WPGMA:} \qquad D(C_k, (C_i \cup C_j)) \leftarrow \frac{1}{2} \left(D(C_k, C_i) \; + \; D(C_k, C_j) \right) \qquad (2)$$

$$\textbf{Single-Linkage:} \qquad D(C_k, (C_i \cup C_j)) \leftarrow \min \{ D(C_k, C_i), D(C_k, C_j) \} \qquad (3)$$

Most GCP algorithms mentioned in the literature (see [15, 16]) use a *convex* reduction formulae in step 3, meaning that the value set for $D(C_k, (C_i \cup C_j))$ lies between $D(C_k, C_i)$ and $D(C_k, C_j)$. We denote such algorithms as *convex GCP algorithms*. Some works (e.g. [15, 4, 5]) consider a weaker property of the reduction step, called the *reducibility property*: if $D(C_i, C_j) \leq \min\{D(C_k, C_i), D(C_k, C_j)\}$, then $\min\{D(C_k, C_i), D(C_k, C_j)\} \leq D(C_k, (C_i \cup C_j))$. It is easy to see that convex GCP algorithms satisfy the reducibility property. Whenever convexity is assumed in this paper, it is actually sufficient to assume reducibility.

Due to their wide applicability, several attempts were made to implement GCP clustering algorithms in $O(n^2)$ time, which is the obvious lower bound. Eppstein [9] achieves this goal by using a data structure called *quadtree*, which

enables performing the following operations in $O(n)$ time: insertion/deletion of an element and computation of a closest pair. Other implementations of GCP clustering algorithms, which are more common in practice, use simpler data structures that allow $O(n^2)$ implementations only in some special cases [19, 4, 7].

In this paper we focus on an approach which reduces running time by relaxing the selection criterion used in step 1: instead of selecting a *globally closest* cluster-pair in each iteration, the chosen pair is only required to be *locally closest*, meaning that each of the two clusters is closest to the other, but their dissimilarity is not necessarily minimal among all pairwise dissimilarities. We term this relaxed selection scheme the *locally closest pair* (LCP) scheme. A technique which implements convex LCP clustering algorithms in $O(n^2)$ time was presented in [3, 13, 15]. Recently we used a similar technique in a different context in [11]. While this optimal implementation is less general than the one in [9], it is considerably simpler and uses only elementary data structures, which makes it particularly appealing.

A natural question raised by this relaxation in the clustering scheme is whether the resulted LCP algorithms are *equivalent* to the original GCP algorithms, in the sense that they have the same input-output relation. This question was already addressed in [15, 16]. It is informally argued there that both schemes are equivalent when a convex reduction formula is used. However, as we demonstrate in Section 3, convexity alone is insufficient for implying this equivalence. The main contribution of this paper is a formulation of conditions under which an LCP clustering algorithm is equivalent to the GCP clustering algorithm which uses the same reduction. We also show that whereas these conditions hold for the reduction formulae used by UPGMA (1) and WPGMA (2), a seemingly minor change (which preserves convexity) in the reduction formula can violate this equivalence.

The rest of this paper is organized as follows. In Section 2 we review relevant implementations of GCP and LCP clustering algorithms, including the simple $O(n^2)$ implementation of LCP algorithms from [15]. In Section 3 we discuss the equivalence of LCP and GCP clustering algorithms and present conditions on the reduction formula under which this equivalence holds.

2 Efficient Implementations of Clustering Algorithms Using Nearest Neighbors

In this section we review several known implementations of GCP and LCP clustering algorithms. These algorithms receive an input dissimilarity matrix D over a set S of n elements, and perform $n-1$ iterations as outlined by Table 1. Each such iteration involves joining a cluster-pair and reducing the input matrix. For all commonly used reduction formulae (see [15]), the reduction step is easily implemented in $O(n)$ time. Thus, the running time of the algorithm is typically dominated by the time required for selecting the cluster-pairs. A

naive approach, which requires $\theta(n^2)$ time in **each iteration** (and a total time complexity of $\theta(n^3)$), scans all possible cluster-pairs to find a globally closest pair. Faster implementations are obtained by maintaining *nearest-neighbors* for all clusters in the current set.

Cluster C_j is said to be a nearest-neighbor (NN) of C_i if it is closest to it. Given the dissimilarity matrix D, finding such a cluster takes $O(n)$ time. Once a NN is kept for each cluster, a globally closest cluster-pair is found in $O(n)$ time by selecting a pair $(C_i, C_j = NN(C_i))$ of minimal dissimilarity. Thus when using nearest-neighbors, the complexity of the algorithm is actually determined by the time it takes to recompute NNs over the reduced cluster set. In particular, NNs should be recalculated for every cluster C_k previously having one of the joined clusters (C_i, C_j) as its NN. In general, this may result in $\Omega(n)$ NN updates during each iteration (see discussion in [15, 16]). By maintaining the entries of each row of D in a heap, these updates can be made in $O(nlog(n))$ time, resulting in a general $O(n^2log(n))$ implementation.

$O(n^2)$ implementations of GCP algorithms using nearest-neighbors are known in some special cases. Day and Edelsbrunner [4] show (using geometrical considerations) that when the input set S consists of points in a bounded-dimension space, the number of NN updates required in each iteration is bounded by a constant. This naturally leads to time complexity of $O(n^2)$ (for all common reduction formulae). In the special case of the single-linkage algorithm, there is a well known $O(n^2)$ implementation [19], which uses the following observation: if either C_i or C_j was a NN of C_k before their joining, then $C_i \cup C_j$ is a NN of C_k after the reduction. The same idea is also used in the fast variant of UPGMA implemented in MUSCLE [7]. However, since the above observation does not hold for the reduction formula of UPGMA, the implied algorithm is not equivalent to UPGMA, as it is guaranteed to yield the same output as UPGMA only on a **limited** set of inputs (e.g. *ultrametrics* [14]). In some sense, the same approach is also taken in the recent fast version of Saitou and Nei's Neighbor Joining algorithm [18, 8].

The techniques used in [4, 7] do not imply, therefore, an $O(n^2)$ algorithm which is equivalent to UPGMA. However, as shown in [15, 11], a reduction in time complexity can be obtained by relaxing the selection criterion of the algorithm to join in each stage a **locally** (rather than globally) closest cluster-pair. Such cluster pairs are referred to in [15, 16] as *reciprocal* (or *mutual*) *nearest-neighbors* (RNNs). Finding RNNs can be easily done by maintaining a *complete nearest-neighbor chain*[1]. A sequence of distinct clusters $P = (C_{i_1}, C_{i_2}, \ldots, C_{i_l})$ is a *nearest-neighbor chain* (with respect to D) if $C_{i_{r+1}} = NN(C_{i_r})$ for all $1 \leq r < l$. A NN-chain is said to be *complete* if it contains at least two clusters, and the last two clusters are RNNs. We conclude this section by reviewing the nearest-neighbor-chain technique [15] which leads to optimal $O(n^2)$ time complexity.

The algorithm starts with an arbitrary cluster and extends a NN-chain starting from it (by computing NNs) until this chain is complete. Given a complete

[1]Similar chains are referred to in [11] as *complete ascending paths*.

NN-chain $P = (C_{i_1}, \ldots, C_{i_l})$, the algorithm joins $C_{i_{l-1}}$ and C_{i_l} (as they are 'locally closest'), performs a reduction of the dissimilarity matrix, and removes $C_{i_{l-1}}, C_{i_l}$ from the NN-chain. If the chain was consequently emptied, it initializes the chain with some arbitrary cluster. The important observation to be made here is that if the reduction is **convex**, the remaining chain is a valid NN-chain corresponding to the reduced dissimilarity matrix. Thus a complete NN-chain P' can be obtained by iteratively extending this chain. Extending the NN-chain by a single cluster and checking whether it is complete takes $O(n)$ time. This operation is invoked no more than $2(n-1)$ times overall, since every cluster added to the chain (except one) is removed from it at some point, and only 2 clusters are removed during each iteration. Thus the resulting total time-complexity is $O(n^2)$.

3 Equivalence of the LCP and GCP Schemes

In the previous section we described the NN-chain technique for implementing convex LCP clustering algorithms in $O(n^2)$ time. In this section we study the properties of the reduction formula under which an LCP clustering algorithm is guaranteed to be equivalent to the corresponding GCP algorithm. In order to demonstrate equivalence for a given reduction formula, we need to show that every clustering obtained by the corresponding LCP algorithm can also be obtained on the same input using the corresponding GCP algorithm. To simplify the presentation in our analysis below, the set S, over which the input dissimilarities are defined, is viewed as a set of singleton clusters ($\mathcal{C}$).

Lemma 3.1. *Let D be a dissimilarity matrix over a cluster set $\mathcal{C}$, s.t. $|\mathcal{C}| > 1$. Then, when executed on $(\mathcal{C}, D)$, a convex LCP clustering algorithm joins (at some point) a cluster-pair $\{C_i, C_j\} \subseteq \mathcal{C}$, which is globally closest under D.*

Proof. The lemma is proved by induction on $|\mathcal{C}|$. It holds trivially when $|\mathcal{C}| = 2$. Assume that $|\mathcal{C}| > 2$, and let $d_{\min}$ denote the minimal pairwise dissimilarity in D. Consider the first cluster-pair $\{C_k, C_l\}$ joined by the algorithm. If $D(C_k, C_l) = d_{\min}$, the lemma follows immediately. Otherwise, $D(C_k, C_l) > d_{\min}$, and since C_k and C_l are RNNs, all pairwise dissimilarities in D involving either C_k or C_l are strictly greater than $d_{\min}$. Now let D' denote the reduced dissimilarity matrix over the reduced cluster-set $\mathcal{C}' = \mathcal{C} \setminus \{C_k, C_l\} \cup \{(C_k \cup C_l)\}$. Since the reduction is convex, all pairwise dissimilarities in D' involving $C_k \cup C_l$ are also strictly greater than $d_{\min}$. Furthermore, $d_{\min}$ is still the minimal pairwise dissimilarity in D'. Therefore, a cluster pair $\{C_i, C_j\} \subseteq \mathcal{C}'$ is globally closest under D' iff it is globally closest under D. The induction hypothesis on $(\mathcal{C}', D')$ implies that at some point in the execution, a globally closest cluster-pair $\{C_i, C_j\} \subseteq \mathcal{C}'$ must be joined. By the previous observation, $\{C_i, C_j\}$ is also globally closest under D. □

Informally, in order to complete the proof of equivalence, we need to show that the joining of the globally closest pair guaranteed by Lemma 3.1 can be

moved to the beginning of the execution without affecting the output. This can be achieved when the reduction formula satisfies an additional property – *commutativity* – which is defined below.

Definition 3.2 (Commutativity of reduction formula)**.** *A reduction formula F is said to be commutative, if the following holds for every dissimilarity matrix D over a cluster set $\mathcal{C}$ (s.t. $|\mathcal{C}| \geq 4$), when using reduction formula F: Given four arbitrary clusters $\{C_{i_1}, C_{j_1}, C_{i_2}, C_{j_2}\} \subseteq \mathcal{C}$, the dissimilarity matrix obtained by first joining $\{C_{i_1}, C_{j_1}\}$ and then joining $\{C_{i_2}, C_{j_2}\}$ (according to steps 2,3 in Table 1) is equal to the dissimilarity matrix obtained by first joining $\{C_{i_2}, C_{j_2}\}$ and then joining $\{C_{i_1}, C_{j_1}\}$.*

Discussion: The commutativity property holds trivially when the reduction formula F induces a dissimilarity function D_F over the set of all clusters, s.t. $D_F(C_1, C_2)$ depends only on the dissimilarities between elements in $C_1 \cup C_2$. Such are for instance the reduction formulae for UPGMA (1) and the single linkage algorithm (3):

$$D_{UPGMA}(C, C') \;=\; \frac{1}{|C||C'|} \sum_{i \in C, j \in C'} D(i,j) \tag{4}$$

$$D_{single-linkage}(C, C') \;=\; \min_{i \in C, j \in C'} \{D(i,j)\} \tag{5}$$

Some common reduction formulae, such as the one used by WPGMA (2), do not induce such a function. In fact, it is possible for two executions of WPGMA on the **same input** to lead to different reduced dissimilarities (between the same clusters). Nevertheless, it is not hard to see that the reduction formula of WPGMA is in fact commutative.

Convex reduction formulae are not necessarily commutative, as demonstrated by the following reduction formula:

$$D(C_k, (C_i \cup C_j)) \;\leftarrow\; \left\lceil \frac{|C_i|}{|C_i| + |C_j|} D(C_k, C_i) \;+\; \frac{|C_j|}{|C_i| + |C_j|} D(C_k, C_j) \right\rceil \tag{6}$$

This formula is identical to the one used by UPGMA, apart from the fact that it rounds up the result. It is clearly convex, but as illustrated in Figure 1, the LCP algorithm which uses this formula is not equivalent to its GCP counterpart. The example described in Figure 1 details a specific execution of the LCP algorithm which leads to different clustering than the one produced by the (unique) execution of the GCP algorithm on the same input. The different output is a direct result of the non-commutativity of the above reduction formula.

Finally, we show that convexity and commutativity imply the desired equivalence between the LCP and GCP clustering schemes:

Theorem 3.3. *Let D be an arbitrary dissimilarity matrix over a cluster-set $\mathcal{C}$, and let F be a convex and commutative reduction formula. Then every execution (on $(\mathcal{C}, D)$) of the LCP clustering algorithm which uses F has an equivalent execution of the corresponding GCP algorithm.*

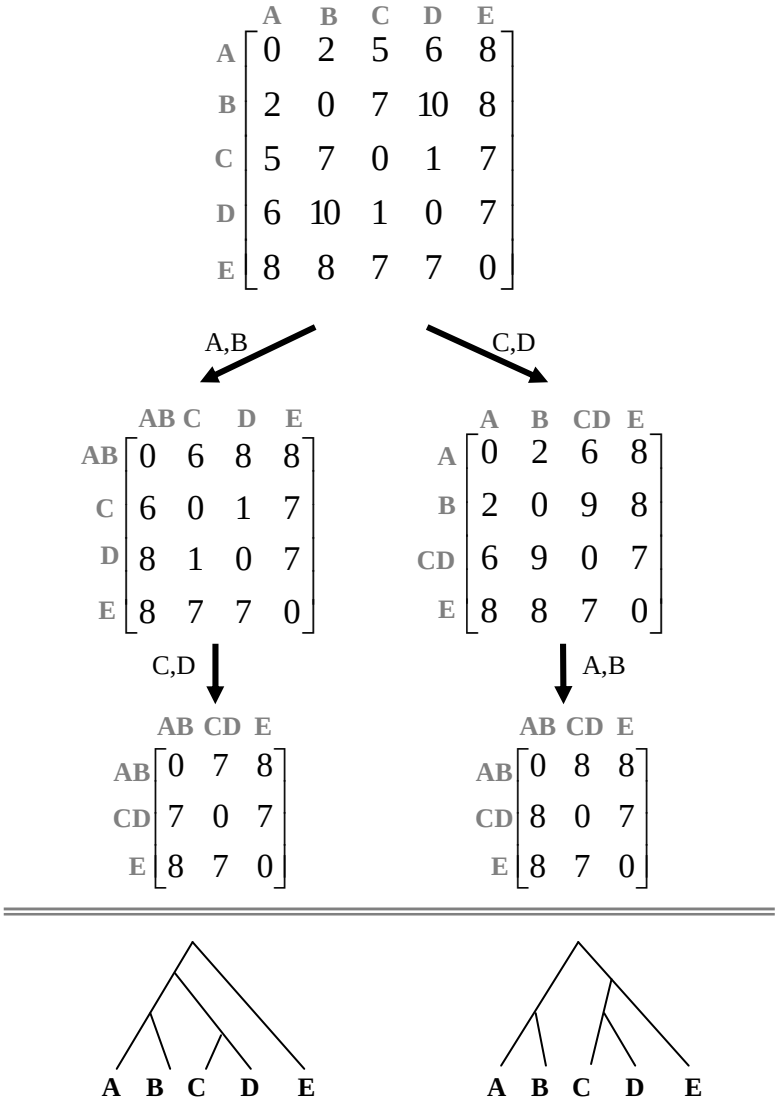


Figure 1: **Convexity does not guarantee equivalence**: when using the convex reduction formula from (6), the unique GCP execution first joins C and D and then joins A and B (right path), while an LCP execution invokes the same joining events in reverse order (left path). The resulting dissimilarity matrices are different and may lead to different clustering.

Proof. By induction on $|\mathcal{C}|$. The claim holds for $|\mathcal{C}| \leq 3$, since in such a case, a locally-closest cluster-pair is globally-closest as well. Assume, therefore, that $|\mathcal{C}| > 3$, and let LE be an execution of the LCP algorithm on $(\mathcal{C}, D)$ which uses F. Let $\{C_i, C_j\} \subseteq \mathcal{C}$ be a globally closest cluster-pair in D which is joined during LE, as guaranteed by Lemma 3.1. Since F is commutative, we can move up the joining of this cluster-pair to the beginning of LE, without changing the output of the execution. Thus we can assume that the first cluster-pair joined by the execution LE is $\{C_i, C_j\}$. Observe *LE'*, the suffix execution of LE defined on the reduced cluster set $\mathcal{C}' = \mathcal{C} \setminus \{C_i, C_j\} \cup \{(C_i \cup C_j)\}$ and the corresponding reduced matrix D'. The induction hypothesis implies that there is an execution *GE'* of the corresponding GCP algorithm on $(\mathcal{C}', D')$ which is equivalent to *LE'*. By appending *GE'* to the joining of $\{C_i, C_j\}$ we get an execution GE of the GCP algorithm which is equivalent to LE. □

In conclusion, Theorem 3.3 implies that every GCP clustering algorithm which uses a reduction formula which is both convex and commutative can be implemented by the NN-chain technique presented in Section 2. This includes many common variants, such as UPGMA, WPGMA and the single linkage algorithm.

Acknowledgement

We would like to thank David Bryant for drawing our attention to [15].

References

[1] S. Akella, J. Davis, and P. Waddell. Accelerating phylogenetics computing on the desktop: experiments with executing UPGMA in programmable logic. In *Engineering in Medicine and Biology Society (IEMBS)*, volume 4, pages 2864–2868, 2004.

[2] J. Barthelemy and A. Guenoche. *Trees and proximities representations.* Wiley, 1991.

[3] J. Benzécri. Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques. *Les Cahiers de l'Analyse des Données*, VII(2):209–219, 1982.

[4] W. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984.

[5] C. Ding and X. He. Cluster aggregate inequality and multi-level hierarchical clustering. In *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 224–231, 2005.

[6] Z. Du and F. Lin. A novel parallelization approach for hierarchical clustering. *Parallel Comput.*, 31(5):523–527, 2005.

[7] R. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throuhput. *Nucleic Acids Research*, 32(5).

[8] I. Elias and J. Lagergren. Fast neighbor joining. In *Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1263–1274. Springer-Verlag, July 2005.

[9] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *J. Exp. Algorithmics*, 5:1–23, 2000.

[10] V. Filkov and S. Skiena. Heterogeneous data integration with the consensus clustering formalism. In *International Workshop on Data Integration in the Life Sciences (DILS)*, pages 110–123, 2004.

[11] I. Gronau and S. Moran. Neighbor joining algorithms for inferring phylogenies via LCA-distances. *Journal of Computational Biology*, 14(1):1–15, 2007.

[12] A. Have T. Christiansen J. Larsen, L. Hansen and T. Kolenda. Webmining: learning from the world wide web. *Computational Statistics and Data Analysis*, 38(4):517–532, 2002.

[13] 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(2):219–225, 1982.

[14] M. Křivánek. The complexity of ultrametric partitions on graphs. *Inform. Process. Lett.*, 27:265–270, 1988.

[15] F. Murtagh. Complexities of hierarchic clustering algorithms: state of the art. *Computational Statistic Quarterly*, 1(2):101–113, 1984.

[16] F. Murtagh. Clustering in massive data sets. In P. Pardalos J. Abello and M. Reisende, editors, *Handbook of massive data sets*, pages 501–543. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[17] C. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313–1325, 1995.

[18] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4:406–425, 1987.

[19] R. Sibson. SLINK: an optimally effcient algorithm for the single link cluster method. *The Computer Journal*, 16:30–34, 1973.

[20] P. Sneath and R. Sokal. *Numerical Taxonomy : the principles and practice of numerical classification.* W. H. Freeman, San Francisco, 1973.

An Efficient *k*-Means Clustering Algorithm: Analysis and Implementation

Tapas Kanungo, *Senior Member*, *IEEE*, David M. Mount, *Member*, *IEEE*, Nathan S. Netanyahu, *Member*, *IEEE*, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, *Senior Member*, *IEEE*

Abstract—In k-means clustering, we are given a set of n data points in d-dimensional space $\mathbf{R}^d$ and an integer k and the problem is to determine a set of k points in $\mathbf{R}^d$, called centers, so as to minimize the mean squared distance from each data point to its nearest center. A popular heuristic for k-means clustering is Lloyd's algorithm. In this paper, we present a simple and efficient implementation of Lloyd's k-means clustering algorithm, which we call the filtering algorithm. This algorithm is easy to implement, requiring a kd-tree as the only major data structure. We establish the practical efficiency of the filtering algorithm in two ways. First, we present a data-sensitive analysis of the algorithm's running time, which shows that the algorithm runs faster as the separation between clusters increases. Second, we present a number of empirical studies both on synthetically generated data and on real data sets from applications in color quantization, data compression, and image segmentation.

Index Terms—Pattern recognition, machine learning, data mining, k-means clustering, nearest-neighbor searching, k-d tree, computational geometry, knowledge discovery.

1 Introduction

Clustering problems arise in many different applications, such as data mining and knowledge discovery [19], data compression and vector quantization [24], and pattern recognition and pattern classification [16]. The notion of what constitutes a good cluster depends on the application and there are many methods for finding clusters subject to various criteria, both ad hoc and systematic. These include approaches based on splitting and merging such as ISODATA [6], [28], randomized approaches such as CLARA [34], CLARANS [44], methods based on neural nets [35], and methods designed to scale to large databases, including DBSCAN [17], BIRCH [50], and ScaleKM [10]. For further information on clustering and clustering algorithms, see [34], [11], [28], [30], [29].

Among clustering formulations that are based on minimizing a formal objective function, perhaps the most widely used and studied is k-means clustering. Given a set of n data points in real d-dimensional space, $\mathbf{R}^d$, and an integer k, the problem is to determine a set of k points in $\mathbf{R}^d$, called *centers*, so as to minimize the mean squared distance from each data point to its nearest center. This measure is often called the *squared-error distortion* [28], [24] and this type of clustering falls into the general category of variance-based clustering [27], [26].

Clustering based on k-means is closely related to a number of other clustering and location problems. These include the Euclidean k-*medians* (or the *multisource Weber problem*) [3], [36] in which the objective is to minimize the sum of distances to the nearest center and the geometric k-*center* problem [1] in which the objective is to minimize the maximum distance from every point to its closest center. There are no efficient solutions known to any of these problems and some formulations are NP-hard [23]. An asymptotically efficient approximation for the k-means clustering problem has been presented by Matousek [41], but the large constant factors suggest that it is not a good candidate for practical implementation.

One of the most popular heuristics for solving the k-means problem is based on a simple iterative scheme for finding a locally minimal solution. This algorithm is often called the k-*means algorithm* [21], [38]. There are a number of variants to this algorithm, so, to clarify which version we are using, we will refer to it as *Lloyd's algorithm*. (More accurately, it should be called the *generalized Lloyd's algorithm* since Lloyd's original result was for scalar data [37].)

Lloyd's algorithm is based on the simple observation that the optimal placement of a center is at the centroid of the associated cluster (see [18], [15]). Given any set of k centers Z, for each center $z \in Z$, let $V(z)$ denote its *neighborhood*, that is, the set of data points for which z is the nearest neighbor. In geometric terminology, $V(z)$ is the set of data points lying in the Voronoi cell of z [48]. Each stage of Lloyd's algorithm moves every center point z to the centroid of $V(z)$ and then updates $V(z)$ by recomputing the distance from each point to

- *T. Kanungo is with the IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. E-mail: kanungo@almaden.ibm.com.*
- *D.M. Mount is with the Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail: mount@cs.umd.edu.*
- *N.S. Netanyahu is with the Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel. E-mail: nathan@cs.biu.ac.il.*
- *C.D. Piatko is with the Applied Physics Laboratory, The John Hopkins University, Laurel, MD 20723. E-mail: christine.piatko@jhuapl.edu.*
- *R. Silverman is with the Center for Automation Research, University of Maryland, College Park, MD 20742. E-mail: ruth@cfar.umd.edu.*
- *A.Y. Wu is with the Department of Computer Science and Information Systems, American University, Washington, DC 20016. E-mail: awu@american.edu.*

Manuscript received 1 Mar. 2000; revised 6 Mar. 2001; accepted 24 Oct. 2001.
Recommended for acceptance by C. Brodley.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 111599.

its nearest center. These steps are repeated until some convergence condition is met. See Faber [18] for descriptions of other variants of this algorithm. For points in general position (in particular, if no data point is equidistant from two centers), the algorithm will eventually converge to a point that is a local minimum for the distortion. However, the result is not necessarily a global minimum. See [8], [40], [47], [49] for further discussion of its statistical and convergence properties. Lloyd's algorithm assumes that the data are memory resident. Bradley et al. [10] have shown how to scale k-means clustering to very large data sets through sampling and pruning. Note that Lloyd's algorithm does not specify the initial placement of centers. See Bradley and Fayyad [9], for example, for further discussion of this issue.

Because of its simplicity and flexibility, Lloyd's algorithm is very popular in statistical analysis. In particular, given any other clustering algorithm, Lloyd's algorithm can be applied as a postprocessing stage to improve the final distortion. As we shall see in our experiments, this can result in significant improvements. However, a straightforward implementation of Lloyd's algorithm can be quite slow. This is principally due to the cost of computing nearest neighbors.

In this paper, we present a simple and efficient implementation of Lloyd's algorithm, which we call the *filtering algorithm*. This algorithm begins by storing the data points in a kd-tree [7]. Recall that, in each stage of Lloyd's algorithm, the nearest center to each data point is computed and each center is moved to the centroid of the associated neighbors. The idea is to maintain, for each node of the tree, a subset of *candidate centers*. The candidates for each node are pruned, or "filtered," as they are propagated to the node's children. Since the kd-tree is computed for the data points rather than for the centers, there is no need to update this structure with each stage of Lloyd's algorithm. Also, since there are typically many more data points than centers, there are greater economies of scale to be realized. Note that this is not a new clustering method, but simply an efficient implementation of Lloyd's k-means algorithm.

The idea of storing the data points in a kd-tree in clustering was considered by Moore [42] in the context of estimating the parameters of a mixture of Gaussian clusters. He gave an efficient implementation of the well-known EM algorithm. The application of this idea to k-means was discovered independently by Alsabti et al. [2], Pelleg and Moore [45], [46] (who called their version the blacklisting algorithm), and Kanungo et al. [31]. The purpose of this paper is to present a more detailed analysis of this algorithm. In particular, we present a theorem that quantifies the algorithm's efficiency when the data are naturally clustered and we present a detailed series of experiments designed to advance the understanding of the algorithm's performance.

In Section 3, we present a *data-sensitive* analysis which shows that, as the separation between clusters increases, the algorithm runs more efficiently. We have also performed a number of empirical studies, both on synthetically generated data and on real data used in applications ranging from color quantization to data compression to image segmentation. These studies, as well as a comparison we ran against the popular clustering scheme, BIRCH[1] [50], are reported in Section 4. Our experiments show that the filtering algorithm is quite efficient even when the clusters are not well-separated.

1. Balanced Iterative Reducing and Clustering using Hierarchies.

2 The Filtering Algorithm

In this section, we describe the filtering algorithm. As mentioned earlier, the algorithm is based on storing the multidimensional data points in a kd-tree [7]. For completeness, we summarize the basic elements of this data structure. Define a *box* to be an axis-aligned hyper-rectangle. The *bounding box* of a point set is the smallest box containing all the points. A kd-tree is a binary tree, which represents a hierarchical subdivision of the point set's bounding box using axis aligned splitting hyperplanes. Each node of the kd-tree is associated with a closed box, called a *cell*. The root's cell is the bounding box of the point set. If the cell contains at most one point (or, more generally, fewer than some small constant), then it is declared to be a *leaf*. Otherwise, it is split into two hyperrectangles by an axis-orthogonal hyperplane. The points of the cell are then partitioned to one side or the other of this hyperplane. (Points lying on the hyperplane can be placed on either side.) The resulting subcells are the *children* of the original cell, thus leading to a binary tree structure. There are a number of ways to select the splitting hyperplane. One simple way is to split orthogonally to the longest side of the cell through the median coordinate of the associated points [7]. Given n points, this produces a tree with $O(n)$ nodes and $O(\log n)$ depth.

We begin by computing a kd-tree for the given data points. For each internal node u in the tree, we compute the number of associated data points $u.count$ and weighted centroid $u.wgtCent$, which is defined to be the vector sum of all the associated points. The actual centroid is just $u.wgtCent/u.count$. It is easy to modify the kd-tree construction to compute this additional information in the same space and time bounds given above. The initial centers can be chosen by any method desired. (Lloyd's algorithm does not specify how they are to be selected. A common method is to sample the centers at random from the data points.) Recall that, for each stage of Lloyd's algorithm, for each of the k centers, we need to compute the centroid of the set of data points for which this center is closest. We then move this center to the computed centroid and proceed to the next stage.

For each node of the kd-tree, we maintain a set of *candidate centers*. This is defined to be a subset of center points that might serve as the nearest neighbor for some point lying within the associated cell. The candidate centers for the root consist of all k centers. We then propagate candidates down the tree as follows: For each node u, let C denote its cell and let Z denote its candidate set. First, compute the candidate $z^* \in Z$ that is closest to the midpoint of C. Then, for each of the remaining candidates $z \in Z \backslash \{z^*\}$, if no part of C is closer to z than it is to z^*, we can infer that z is not the nearest center to any data point associated with u and, hence, we can prune, or "filter," z from the list of candidates. If u is associated with a single candidate (which must be z^*) then z^* is the nearest neighbor of all its data points. We can assign them to z^* by adding the associated weighted centroid and counts to z^*. Otherwise, if u is an internal node, we recurse on its children. If u is a leaf node, we compute the distances from its associated data point to all the candidates in Z and assign the data point to its nearest center. (See Fig. 1.)

It remains to describe how to determine whether there is any part of cell C that is closer to candidate z than to z^*. Let H be the hyperplane bisecting the line segment $\overline{zz^*}$. (See Fig. 2.) H defines two halfspaces; one that is closer to z and

```
Filter(kdNode u, CandidateSet Z) {
    C ← u.cell;
    if (u is a leaf) {
        z* ← the closest point in Z to u.point;
        z*.wgtCent ← z*.wgtCent + u.point;
        z*.count ← z*.count + 1;
    }
    else {
        z* ← the closest point in Z to C's midpoint;
        for each (z ∈ Z \ {z*})
            if (z.isFarther(z*, C)) Z ← Z \ {z};
        if (|Z| = 1) {
            z*.wgtCent ← z*.wgtCent + u.wgtCent;
            z*.count ← z*.count + u.count;
        }
        else {
            Filter(u.left, Z);
            Filter(u.right, Z);
        }
    }
}
```

Fig. 1. The filtering algorithm.

the other to z^*. If C lies entirely to one side of H, then it must lie on the side that is closer to z^* (since C's midpoint is closer to z^*) and so z may be pruned. To determine which is the case, consider the vector $\vec{u} = z - z^*$, directed from z^* to z. Let $v(H)$ denote the vertex of C that is extreme in this direction, that is, the vertex of C that maximizes the dot product $(v(H) \cdot \vec{u})$. Thus, z is pruned if and only if $dist(z, v(H)) \geq dist(z^*, v(H))$. (Squared distances may be used to avoid taking square roots.) To compute $v(H)$, let $[C_i^{\min}, C_i^{\max}]$ denote the projection of C onto the ith coordinate axis. We take the ith coordinate of $v(H)$ to be $C_i^{\min}$ if the ith coordinate of $\vec{u}$ is negative and $C_i^{\max}$ otherwise. This computation is implemented by a procedure $z.isFarther(z^*, C)$, which returns true if every part of C is farther from z than to z^*.

The initial call is to $\mathrm{Filter}(r, Z_0)$, where r is the root of the tree and Z_0 is the current set of centers. On termination, center z is moved to the centroid of its associated points, that is, $z \leftarrow z.wgtCent/z.count$.

Our implementation differs somewhat from those of Alsabti et al. [2] and Pelleg and Moore [45]. Alsabti et al.'s implementation of the filtering algorithm uses a less effective pruning method based on computing the minimum and maximum distances to each cell, as opposed to the bisecting hyperplane criterion. Pelleg and Moore's implementation uses the bisecting hyperplane, but they define z^* (called the *owner*) to be the candidate that minimizes the distance to the cell rather than the midpoint of the cell. Our approach has the advantage that if two candidates lie within the cell, it will select the candidate that is closer to the cell's midpoint.

3 Data Sensitive Analysis

In this section, we present an analysis of the time spent in each stage of the filtering algorithm. Traditional worst-case analysis is not really appropriate here since, in principle, the algorithm might encounter scenarios in which it degenerates to brute-force search. This happens, for example, if the center points are all located on a unit sphere centered at the origin and the data points are clustered tightly around the origin. Because the centers are nearly equidistant to any subset of data points, very little pruning takes place and the algorithm degenerates to a brute force $O(kn)$ search. Of course, this is an absurdly contrived scenario. In this section, we will analyze the algorithm's running time not only as a function of k and n, but as a function of the degree to which the data set consists of well-separated clusters. This sort of approach has been applied recently by Dasgupta [13] and Dasgupta and Shulman [14] in the context of clustering data sets that are generated by mixtures of Gaussians. In contrast, our analysis does not rely on any assumptions regarding Gaussian distributions.

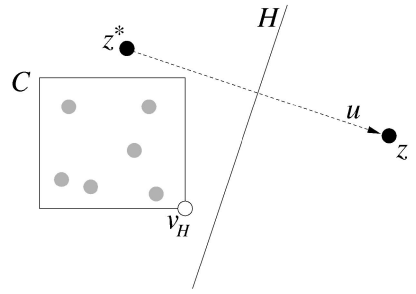


Fig. 2. Candidate z is pruned because C lies entirely on one side of the bisecting hyperplane H.

For the purposes of our theoretical results, we will need a data structure with stronger geometric properties than the simple kd-tree. Consider the basic kd-tree data structure described in the previous section. We define the *aspect ratio* of a cell to be the ratio of the length of its longest side to its shortest side. The *size* of a cell is the length of its longest side. The cells of a kd-tree may generally have arbitrarily high aspect ratio. For our analysis, we will assume that, rather than a kd-tree, the data points are stored in a structure called a *balanced box-decomposition tree* (or *BBD-tree*) for the point set [5]. The following two lemmas summarize the relevant properties of the BBD-tree. (See [5] for proofs.)

Lemma 1. *Given a set of n data points P in $\mathbf{R}^d$ and a bounding hypercube C for the points, in $O(dn \log n)$ time it is possible to construct a BBD-tree representing a hierarchical decomposition of C into cells of complexity $O(d)$ such that:*

1. *The tree has $O(n)$ nodes and depth $O(\log n)$.*
2. *The cells have bounded aspect ratio and, with every $2d$ levels of descent in the tree, the sizes of the associated cells decrease by at least a factor of $1/2$.*

Lemma 2 (Packing Constraint). *Consider any set $\mathcal{C}$ of cells of the BBD-tree with pairwise disjoint interiors, each of size at least s, that intersect a ball of radius r. The size of such a set is, at most, $\left(1 + \lceil \frac{4r}{s} \rceil\right)^d$.*

Our analysis is motivated by the observation that a great deal of the running time of Lloyd's algorithm is spent in the later stages of the algorithm when the center points are close to their final locations but the algorithm has not yet converged [25]. The analysis is based on the assumption that the data set

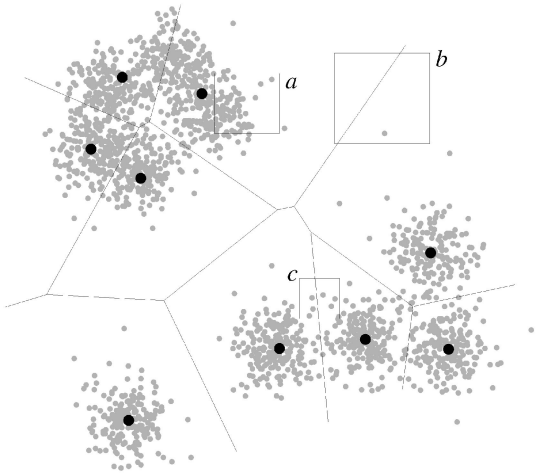


Fig. 3. Filtering algorithm analysis. Note that the density of points near the edges of the Voronoi diagram is relatively low.

can indeed be clustered into k natural clusters and that the current centers are located close to the true cluster centers. These are admittedly strong assumptions, but our experimental results will bear out the algorithm's efficiency even when these assumptions are not met.

We say that a node is *visited* if the Filter procedure is invoked on it. An internal node is *expanded* if its children are visited. A nonexpanded internal node or a leaf node is a *terminal node*. A nonleaf node is terminal if there is no center z that is closer to any part of the associated cell than the closest center z^*. Note that, if a nonleaf cell intersects the Voronoi diagram (that is, there is no center that is closest to every part of the cell), then it cannot be a terminal node. For example, in Fig. 3, the node with cell a is terminal because it lies entirely within the Voronoi cell of its nearest center. Cell b is terminal because it is a leaf. However, cell c is not terminal because it is not a leaf and it intersects the Voronoi diagram. Observe that, in this rather typical example, if the clusters are well-separated and the center points are close to the true cluster centers, then a relatively small fraction of the data set lies near the edges of the Voronoi diagram of the centers. The more well-separated the clusters, the smaller this fraction will be and, hence, fewer cells of the search structure will need to be visited by the algorithm. Our analysis formalizes this intuition.

We assume that the data points are generated independently from k different multivariate distributions in $\mathbf{R}^d$. Consider any one such distribution. Let $\mathbf{X} = (x_1, x_2, \ldots, x_d)$ denote a random vector from this distribution. Let $\mu \in \mathbf{R}^d$ denote the mean point of this distribution and let Σ denote the $d \times d$ *covariance matrix* for the distribution [22]

$$\Sigma = E((\mathbf{X} - \mu)(\mathbf{X} - \mu)^T).$$

Observe that the diagonal elements of Σ are the variances of the random variables that are associated, respectively, with the individual coordinates. Let $\mathrm{tr}(\Sigma)$ denote the *trace* of Σ, that is, the sum of its diagonal elements. We will measure the *dispersion* of the distribution by the variable $\sigma = \sqrt{\mathrm{tr}(\Sigma)}$. This is a natural generalization of the notion of standard deviation for a univariate distribution.

We will characterize the degree of separation of the clusters by two parameters. Let $\mu^{(i)}$ and $\sigma^{(i)}$ denote, respectively, the mean and dispersion of the ith cluster distribution. Let

$$r_{\min} = \frac{1}{2}\min_{i \neq j} |\mu^{(i)} - \mu^{(j)}| \qquad \text{and} \qquad \sigma_{\max} = \max_i \sigma^{(i)},$$

where $|q - p|$ denotes the Euclidean distance between points q and p. The former quantity is half the minimum distance between any two cluster centers and the latter is the maximum dispersion. Intuitively, in well-separated clusters, $r_{\min}$ is large relative to $\sigma_{\max}$. We define the *cluster separation* of the point distribution to be

$$\rho = \frac{r_{\min}}{\sigma_{\max}}.$$

This is similar to Dasgupta's notion of pairwise c-separated clusters, where $c = 2\rho$ [13]. It is also similar to the separation measure for cluster i of Coggins and Jain [12], defined to be $\min_{j \neq i} |\mu^{(i)} - \mu^{(j)}|/\sigma^{(i)}$.

We will show that, assuming that the candidate centers are relatively close to the cluster means, $\mu^{(i)}$, then, as ρ increases (i.e., as clusters are more well-separated) the algorithm's running time improves. Our proof makes use of the following straightforward generalization of Chebyshev's inequality (see [20]) to multivariate distributions. The proof proceeds by applying Chebyshev's inequality to each coordinate and summing the results over all d coordinates.

Lemma 3. *Let $\mathbf{X}$ be a random vector in $\mathbf{R}^d$ drawn from a distribution with mean μ and dispersion σ (the square root of the trace of the covariance matrix). Then, for all positive t,*

$$\Pr(|\mathbf{X} - \mu| > t\sigma) \leq \frac{d}{t^2}.$$

For $\delta \geq 0$, we say that a set of candidates is δ-*close* with respect to a given set of clusters if, for each center $c^{(i)}$, there is an associated cluster mean $\mu^{(i)}$ within distance at most $\delta r_{\min}$ and vice versa. Here is the main result of this section. Recall that a node is *visited* if the Filter procedure is invoked on it.

Theorem 1. *Consider a set of n points in $\mathbf{R}^d$ drawn from a collection of cluster distributions with cluster separation $\rho = r_{\min}/\sigma_{\max}$, and consider a set of k candidate centers that are δ-close to the cluster means, for some $\delta < 1$. Then, for any ϵ, $0 < \epsilon < 1 - \delta$, and for some constant c, the expected number of nodes visited by the filtering algorithm is*

$$O\left(k\left(\frac{c\sqrt{d}}{\epsilon}\right)^d + 2^d k \log n + \frac{dn}{\rho^2(1-\epsilon)^2}\right).$$

Before giving the proof, let us make a few observations about this result. The result provides an upper bound the number of nodes visited. The time needed to process each node is proportional to the number of candidates for the node, which is at most k and is typically much smaller. Thus, the total running time of the algorithm is larger by at most a factor of k. Also, the total running time includes an additive contribution of $O(dn \log n)$ time needed to build the initial BBD-tree.

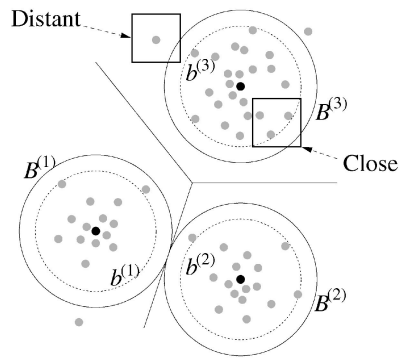


Fig. 4. Distant and close nodes.

We note that, for fixed d, ρ, and ϵ (bounded away from 0 and $1-\delta$), the number of nodes visited as a function of n is $O(n)$ (since the last term in the analysis dominates). Thus, the overall running time is $O(kn)$, which seems to be no better than the brute-force algorithm. The important observation, however, is that, as cluster separation ρ increases, the $O(kn/\rho^2)$ running time decreases and the algorithm's efficiency increases rapidly. The actual expected number of cells visited may be much smaller for particular distributions.

Proof. Consider the ith cluster. Let $c^{(i)}$ denote a candidate center that is within distance $\delta r_{\min}$ from its mean $\mu^{(i)}$. Let $\sigma^{(i)}$ denote its dispersion. Let $B^{(i)}$ denote a ball of radius $r_{\min}$ centered at $\mu^{(i)}$. Observe that, because no cluster centers are closer than $2r_{\min}$, these balls have pairwise disjoint interiors. Let $b^{(i)}$ denote a ball of radius $r_{\min}(1-\epsilon-\delta)$ centered at $c^{(i)}$. Because $c^{(i)}$ is δ-close to μ_i, $b^{(i)}$ is contained within a ball of radius $r_{\min}(1-\epsilon)$ centered at $\mu^{(i)}$ and, hence, is contained within $B^{(i)}$. Moreover, the distance between the boundaries of $B^{(i)}$ and $b^{(i)}$ is at least $\epsilon r_{\min}$.

Consider a visited node and let C denote its associated cell in the BBD-tree. We consider two cases. First, suppose that, for some cluster i, $C \cap b^{(i)} \neq \emptyset$. We call C a *close node*. Otherwise, if the cell does not intersect any of the $b^{(i)}$ balls, it is a *distant node*. (See Fig. 4.) The intuition behind the proof is as follows: If clustering is strong, then we would expect only a few points to be far from the cluster centers and, hence, there are few distant nodes. If a node is close to its center and it is not too large, then there is only one candidate for the nearest neighbor and, hence, the node will be terminal. Because cells of the BBD tree have bounded aspect ratio, the number of large cells that can overlap a ball is bounded.

First, we bound the number of distant visited nodes. Consider the subtree of the BBD-tree induced by these nodes. If a node is distant, then its children are both distant, implying that this induced subtree is a full binary tree (that is, each nonleaf node has exactly two children). It is well-known that the total number of nonleaf nodes in a full binary tree is not greater than the number of leaves and, hence, it suffices to bound the number of leaves.

The data points associated with all the leaves of the induced subtree cannot exceed the number of data points that lie outside of $b^{(i)}$. As observed above, for any cluster i, a data point of this cluster that lies outside of $b^{(i)}$ is at distance from $\mu^{(i)}$ of at least

$$r_{\min}(1-\epsilon) = \frac{r_{\min}}{\sigma^{(i)}}(1-\epsilon)\sigma^{(i)} \;\geq\; \rho(1-\epsilon)\sigma^{(i)}.$$

By Lemma 3, the probability of this occurring is at most $d/(\rho(1-\epsilon))^2$. Thus, the expected number of such data points is at most $dn/(\rho(1-\epsilon))^2$. It follows from standard results on binary tree that the number of distant of nodes is at most twice as large.

Next, we bound the number of close visited nodes. Recall a visited node is said to be *expanded* if the algorithm visits its children. Clearly, the number of close visited nodes is proportional to the number of close expanded nodes. For each cluster i, consider the leaves of the induced subtree consisting of close expanded nodes that intersect $b^{(i)}$. (The total number of close expanded nodes will be larger by a factor of k.) To bound the number of such nodes, we further classify them by the size of the associated cell. An expanded node v whose size is at least $4r_{\min}$ is *large* and, otherwise, it is *small*. We will show that the number of large expanded nodes is bounded by $O(2^d \log n)$ and the number of small expanded nodes is bounded by $O((c\sqrt{d}/\epsilon)^d)$, for some constant c.

We first show the bound on the number of large expanded nodes. In the descent through the BBD-tree, the sizes of the nodes decrease monotonically. Consider the set of all expanded nodes of size greater than $4r_{\min}$. These nodes induce a subtree in the BBD-tree. Let L denote the leaves of this tree. The cells associated with the elements of L have pairwise disjoint interiors and they intersect $b^{(i)}$ and, hence, they intersect $B^{(i)}$. It follows from Lemma 2 (applied to $B^{(i)}$ and the cells associated with L) that there are at most $(1+\lceil 4r_{\min}/(4r_{\min})\rceil)^d = 2^d$ such cells. By Lemma 1, the depth of the tree is $O(\log n)$ and, hence, the total number of expanded large nodes is $O(2^d \log n)$, as desired.

Finally, we consider the small expanded nodes. We assert that the diameter of the cell corresponding to any such node is at least $\epsilon r_{\min}$. If not, then this cell would lie entirely within a ball of radius $r_{\min}(1-\delta)$ of $c^{(i)}$. Since the cell was expanded, we know that there must be a point in this cell that is closer to some other center c_j. This implies that the distance between $c^{(i)}$ and $c^{(j)}$ is less than $2r_{\min}(1-\delta)$. Since the candidates are δ-close, it follows that there are two cluster means $\mu^{(i)}$ and $\mu^{(j)}$ that are closer than $2r_{\min}(1-\delta)+2\delta r_{\min} = 2r_{\min}$. However, this would contradict the definition of $r_{\min}$. A cell in dimension d of diameter x has longest side length of at least $x/\sqrt{d}$. Thus, the size of each such cell is at least $\epsilon r_{\min}/\sqrt{d}$. By Lemma 2, it follows that the number of such cells that overlap $B^{(i)}$ is at most

$$\left(\frac{c\sqrt{d}}{\epsilon}\right)^d.$$

Applying a similar analysis used by Arya and Mount [4] for approximate range searching, the number of expanded nodes is asymptotically the same. This completes the proof. □

4 Empirical Analysis

To establish the practical efficiency of the filtering algorithm, we implemented it and tested its performance on a number of

data sets. These included both synthetically generated data and data used in real applications. The algorithm was implemented in C++ using the g++ compiler and was run on a Sun Ultra 5 running Solaris 2.6. The data structure implemented was a kd-tree that was constructed from the ANN library [43] using the sliding midpoint rule [39]. This decomposition method was chosen because our studies have shown that it performs better than the standard kd-tree decomposition rule for clustered data sets. Essentially, we performed two different types of experiments. The first type compared running times of the filtering algorithm against two different implementations of Lloyd's algorithm. The second type compared cluster distortions obtained for the filtering algorithm with those obtained for the BIRCH clustering scheme [50].

For the running time experiments, we considered two other algorithms. Recall that the essential task is to compute the closest center to each data point. The first algorithm compared against was a simple *brute-force* algorithm, which computes the distance from every data point to every center. The second algorithm, called *kd-center*, operates by building a kd-tree with respect to the center points and then uses the kd-tree to compute the nearest neighbor for each data point. The kd-tree is rebuilt at the start of each stage of Lloyd's algorithm. We compared these two methods against the filtering algorithm by performing two sets of experiments, one involving synthetic data and the other using data derived from applications in image segmentation and compression.

We used the following experimental structure for the above experiments: For consistency, we ran all three algorithms on the same data set and the same initial placement of centers. Because the running time of the algorithm depends heavily on the number of stages, we report the average running time per stage by computing the total running time and then dividing by the number of stages. In the case of the filtering algorithm, we distinguished between two cases according to whether or not the preprocessing time was taken into consideration. The reason for excluding preprocessing time is that, when the number of stages of Lloyd's algorithm is very small, the effect of the preprocessing time is amortized over a smaller number of stages and this introduces a bias.

We measured running time in two different ways. We measured both the CPU time (using the standard `clock()` function) and a second quantity called the number of *node-candidate pairs*. The latter quantity is a machine-independent statistic of the algorithm's complexity. Intuitively, it measures the number of interactions between a node of the kd-tree (or data point in the case of brute force) and a candidate center. For the brute-force algorithm, this quantity is always kn. For the kd-center algorithm, for each data point, we count the number of nodes that were accessed in the kd-tree of the centers for computing its nearest center and sum this over all data points. For the filtering algorithm, we computed a sum of the number of candidates associated with every visited node of the tree. In particular, for each call to $\mathrm{Filter}(u, Z)$, the cardinality of the set Z of candidate centers is accumulated. The one-time cost of building the kd-tree is not included in this measure.

Note that the three algorithms are functionally equivalent if points are in general position. Thus, they all produce the same final result and, so, there is no issue regarding the quality of the final output. The primary issue of interest is the efficiency of the computation.

4.1 Synthetic Data

We ran three experiments to determine the variations in running time as a function of cluster separation, data set size, and dimension. The first experiment tested the validity of Theorem 1. We generated $n = 10,000$ data points in $\mathbf{R}^3$. These points were distributed evenly among 50 clusters as follows: The 50 cluster centers were sampled from a uniform distribution over the hypercube $[-1, 1]^d$. A Gaussian distribution was then generated around each center, where each coordinate was generated independently from a univariate Gaussian with a given standard deviation. The standard deviation varied from 0.01 (very well-separated) up to 0.7 (virtually unclustered). Because the same distribution was used for cluster centers throughout, the expected distances between cluster centers remained constant. Thus, the expected value of the cluster separation parameter ρ varied inversely with the standard deviation. The initial centers were chosen by taking a random sample of data points.

For each standard deviation, we ran each of the algorithms (brute-force, kd-center, and filter) three times. For each of the three runs, a new set of initial center points was generated and all three algorithms were run using the same data and initial center points. The algorithm ran for a maximum of 30 stages or until convergence.

The average CPU times per stage, for all three methods, are shown for $k = 50$ in Fig. 5a and for $k = 20$ in Fig. 6a. The results of these experiments show that the filtering algorithm runs significantly faster than the other two algorithms. Ignoring the bias introduced by preprocessing, its running time improves when the clusters are more well-separated (for smaller standard deviations). The improvement in CPU time predicted by Theorem 1 is not really evident for very small standard deviations because initialization and preprocessing costs dominate. However, this improvement is indicated in Figs. 5b and 6b, which plot the (initialization independent) numbers of node-candidate pairs. The numbers of node candidate pairs for the other two methods are not shown, but varied from 4 to 10 times greater than those of the filtering algorithm.

Our theoretical analysis does not predict that the filtering algorithm will be particularly efficient for unclustered data sets, but it does not preclude this possibility. Nonetheless, we observe in this experiment that the filtering algorithm ran significantly faster than the brute-force and kd-center algorithms, even for large standard deviations. This can be attributed, in part, to the fact that the filtering algorithm simply does a better job in exploiting economies of scale by storing the much larger set of data points in a kd-tree (rather than center points as kd-center does).

The objective of the second experiment was to study the effects of data size on the running time. We generated data sets of points whose size varied from $n = 1,000$ to $n = 20,000$ and where each data set consisted of 50 Gaussian clusters. The standard deviation was fixed at 0.10 and the algorithms were run with $k = 50$. Again, the searches were terminated when stability was achieved or after 30 stages. As before, we ran each case three times with different starting centers and averaged the results. The CPU times per stage and number of

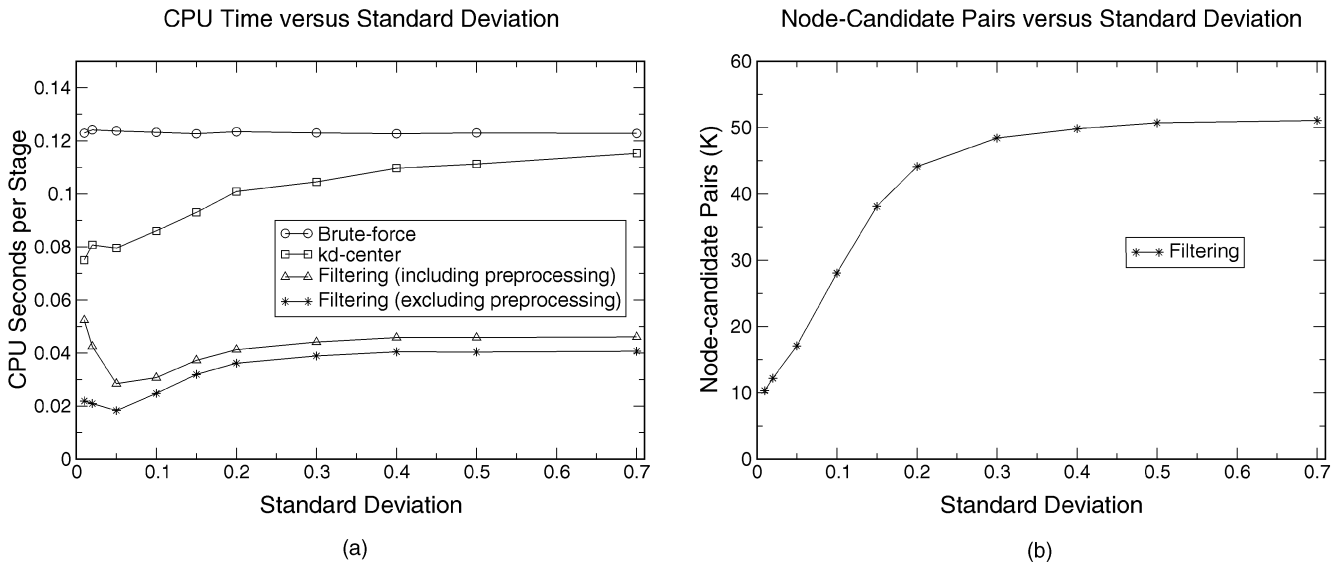


Fig. 5. Average CPU times and node-candidate pairs per stage versus cluster standard deviation for $n = 10,000$, $k = 50$.

node-candidate pairs for all three methods are shown in Fig. 7. The results of this experiment show that, for fixed k and large n, all three algorithms have running times that vary linearly with n, with the filtering algorithm doing the best.

The third experiment was designed to test the effect of the dimension on running times of the three k-means algorithms. Theorem 1 predicts that the filtering algorithm's running time increases exponentially with dimension, whereas that of the brute-force algorithm (whose running time is $O(dkn)$) increases only linearly with dimension d. Thus, as the dimension increases, we expect that the filtering algorithm's speed advantage would tend to diminish. This exponential dependence on dimension seems to be characteristic of many algorithms that are based on kd-trees and many variants, such as R-trees.

We repeated an experimental design similar to the one used by Pelleg and Moore [45]. For each dimension d, we created a random data set with 20,000 points. The points were sampled from a clustered Gaussian distribution with 72 clusters and a standard deviation of 0.05 along each coordinate. (The standard deviation is twice that used by Pelleg and Moore because they used a unit hypercube and our hypercube has side length 2.) The three k-means algorithms were run with $k = 40$ centers. In Fig. 8, we plot the average times taken per stage for the three algorithms. We found that, for this setup, the filtering algorithm outperforms the brute-force and kd-center algorithms for dimensions ranging up to the mid 20s. These results confirm the general trend reported in [45], but the filtering algorithm's performance in moderate dimensions 10 to 20 is considerably better.

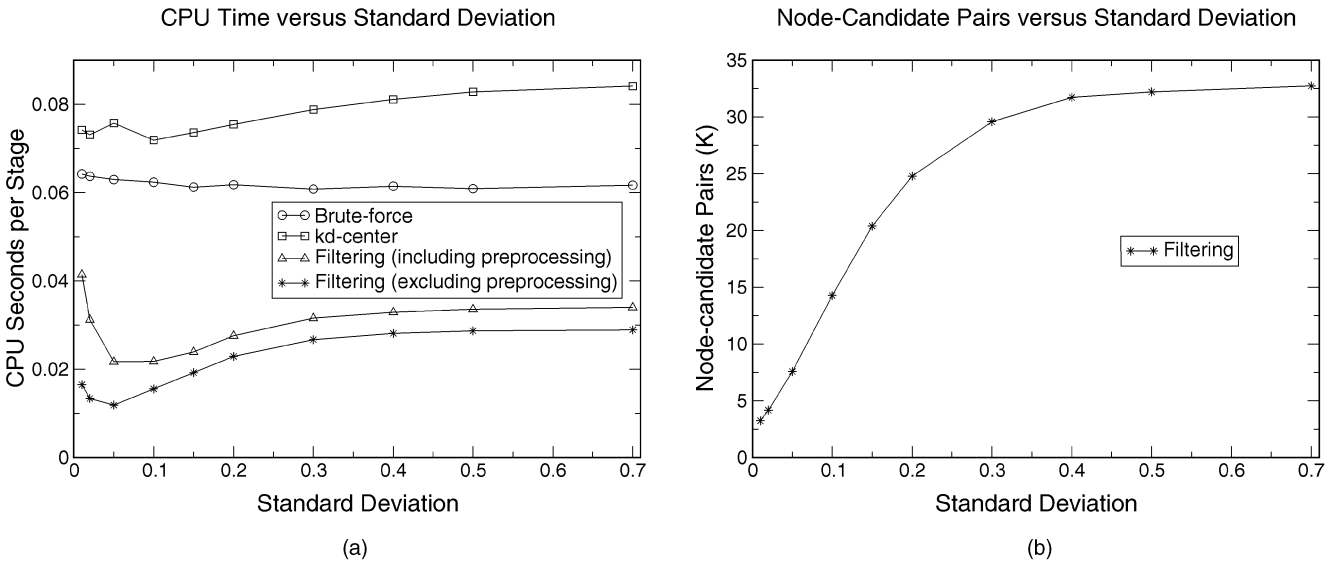


Fig. 6. Average CPU times and node-candidate pairs per stage versus cluster standard deviation for $n = 10,000$, $k = 20$.

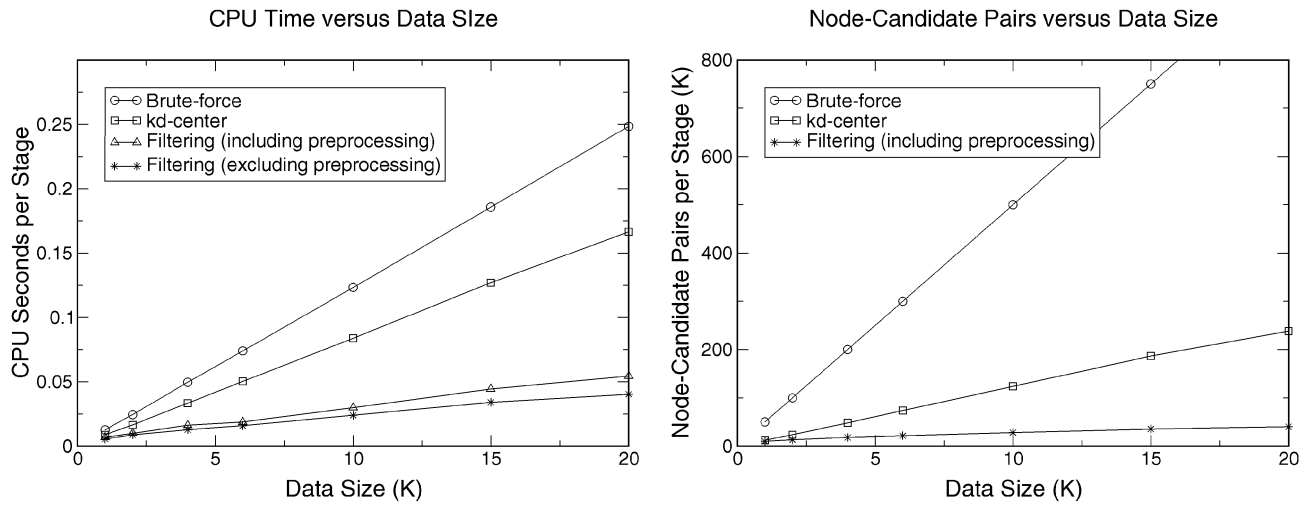


Fig. 7. Running times and node-candidate pairs versus data size for $k = 50$, $\sigma = 0.10$.

4.2 Real Data

To understand the relative efficiency of this algorithm under more practical circumstances, we ran a number of experiments on data sets derived from actual applications in image processing, compression, and segmentation. Each experiment involved solving a k-means problem on a set of points in $\mathbf{R}^d$ for various ds. In each case, we applied all three algorithms: brute force (Brute), kd-center (KDcenter), and filtering (Filter). In the filtering algorithm, we included the preprocessing time in the average. In each case, we computed the average CPU time per stage in seconds (T-) and the average number of node-candidate pairs (NC-). This was repeated for values of k chosen from $\{8, 64, 256\}$. The results are shown in Table 1.

The experiments are grouped into three general categories. The first involves a color quantization application. A color image is given and a number of pixels are sampled from the image (10,000 for this experiment). Each pixel is represented as a triple consisting of red, green, and blue components, each in the range $[0, 255]$ and, hence, is a point in $\mathbf{R}^3$. The input images were chosen from a number of standard images for this application. The images are shown in Fig. 9 and the running time results are given in the upper half of Table 1, opposite the corresponding image names "balls," "kiss," ..., "ball1_s."

The next experiment involved a vector quantization application for data compression. Here, the images are gray-scale images with pixel values in the range $[0, 255]$. Each 2×2 subarray of pixels is selected and mapped to a 4-element vector and the k-means algorithm is run on the resulting set of vectors. The images are shown in Fig. 10 and the results are given in Table 1, opposite the names "couple," "crowd," ..., "woman2."

The final experiment involved image segmentation. A 512×512 Landsat image of Israel consisting of four spectral bands was used. The resulting 4-element vectors in the range $[0, 255]$ were presented to the algorithms. One of the image bands is shown in Fig. 11 and the results are provided in Table 1, opposite the name "Israel."

An inspection of the results reveals that the filtering algorithm significantly outperformed the other two algorithms in all cases. Plots of the underlying point distributions showed that most of these data sets were really not well-clustered. Thus, the filtering algorithm is quite efficient, even when the conditions of Theorem 1 are not satisfied.

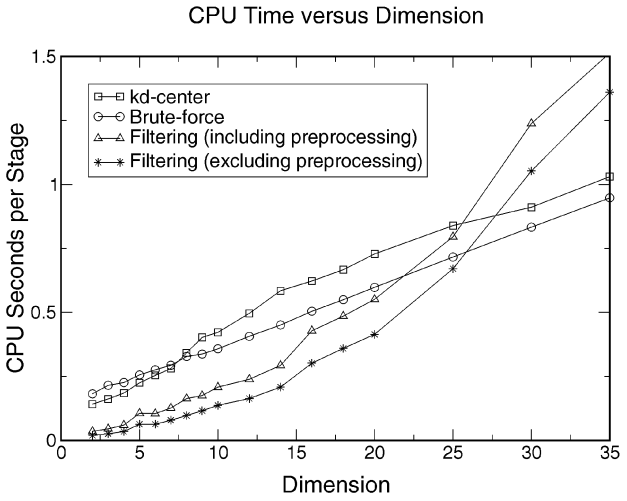


Fig. 8. Average CPU times per stage versus dimension for $n = 20,000$, $\sigma = 0.05$, and $k = 40$.

4.3 Comparison with BIRCH

A comparison between our filtering algorithm and the BIRCH clustering scheme [50] is presented in Table 2. The table shows the distortions produced by both algorithms. The column "BIRCH" reports the distortions obtained with the BIRCH software. The column labeled "Filter" provides the distortions obtained due to the k-means filtering algorithm (with centers initially sampled at random from the data points). Alternatively, one can use the centers obtained by BIRCH as initial centers for our filtering algorithm. This is shown in the column labeled "Combined" of the table. As can be seen, the distortions are always better and considerably better in some of the cases. (This is because k-means cannot increase distortions.) Indeed, as was mentioned in Section 1, k-means may be run to improve the distortion of any clustering algorithm.

TABLE 1
Running Times for Various Test Inputs

Image	Dim	Size	k	T-Brute	T-KDcenter	T-Filter	NC-Brute	NC-KDcenter	NC-Filter
balls	3	10000	8	0.0542857	0.0804762	0.0157143	80	68.13	3.678
			64	0.20875	0.1515	0.01875	640	160.9	17.65
			256	0.742424	0.17875	0.0375758	2560	180	49.77
kiss	3	10000	8	0.053	0.08225	0.0165	80	71.92	10.07
			64	0.209	0.14875	0.0435	640	170.4	41.76
			256	0.7455	0.218	0.089	2560	243.9	111
Lena1	3	10000	8	0.0538889	0.0797222	0.0163889	80	65.18	9.767
			64	0.20925	0.14575	0.046	640	164.7	43.72
			256	0.75025	0.218	0.0915	2560	236.8	118.2
ball1_h	3	10000	8	0.0535	0.0755	0.0095	80	57.27	3.859
			64	0.207	0.119118	0.0245	640	117.8	20.98
			256	0.749231	0.16575	0.0584615	2560	167.2	69.42
ball1_3	3	10000	8	0.0535	0.07625	0.0095	80	58.88	4.073
			64	0.208571	0.11641	0.0251429	640	115.9	20.69
			256	0.748846	0.166364	0.0553846	2560	167.8	65.83
ball1_5	3	10000	8	0.0541667	0.08	0.0225	80	62.72	2.712
			64	0.214688	0.125278	0.029375	640	122.4	23.16
			256	0.74775	0.178	0.059	2560	175.5	71.89
ball1_p	3	10000	8	0.0535714	0.0778571	0.0207143	80	64.37	3.12
			64	0.2095	0.122	0.0285	640	122.2	23.02
			256	0.743043	0.17	0.0608333	2560	174.2	72.93
ball1_s	3	10000	8	0.053	0.0771429	0.015	80	57.95	3.497
			64	0.20875	0.123939	0.0275	640	123.5	24.04
			256	0.745	0.17225	0.059	2560	176	74.3
couple	4	65536	8	0.37475	0.5575	0.15825	524.3	482.3	74.6
			64	1.616	1.21925	0.3535	4194	1452	285.8
			256	5.501	1.732	0.658	1.678e+04	2375	739.6
crowd	4	65536	8	0.36875	0.57375	0.146	524.3	497.7	62.35
			64	1.60975	1.304	0.32175	4194	1512	244.9
			256	5.49975	1.716	0.535	1.678e+04	2347	567.3
lax	4	65536	8	0.36675	0.59225	0.18475	524.3	558.4	99.04
			64	1.6085	1.3725	0.42325	4194	1772	359.7
			256	5.495	1.86175	0.7055	1.678e+04	2776	790.6
Lena2	4	65536	8	0.373	0.55475	0.123	524.3	471.5	50.83
			64	1.615	1.19375	0.34475	4194	1339	266.9
			256	5.50475	1.649	0.618	1.678e+04	2200	677.1
man	4	65536	8	0.3685	0.5505	0.13925	524.3	465.9	63.16
			64	1.609	1.20275	0.35375	4194	1392	278.1
			256	5.49975	1.66125	0.60675	1.678e+04	2275	665.4
woman1	4	65536	8	0.370937	0.563125	0.176875	524.3	483.4	73.65
			64	1.612	1.28275	0.3805	4194	1500	304.2
			256	5.523	1.7945	0.64325	1.678e+04	2450	706.8
woman2	4	65536	8	0.37175	0.544	0.1125	524.3	433.3	35.46
			64	1.615	1.20075	0.30775	4194	1277	214.2
			256	5.51675	1.64675	0.519	1.678e+04	2133	531.4
Israel	4	262144	8	1.6955	3.06775	0.7295	2097	2079	140.4
			64	5.82975	4.82825	1.55225	1.678e+04	5336	558.5
			256	19.975	6.659	2.53775	6.711e+04	8305	1148

5 Concluding Remarks

We have presented an efficient implementation of Lloyd's k-means clustering algorithm, called the filtering algorithm. The algorithm is easy to implement and only requires that a kd-tree be built once for the given data points. Efficiency is achieved because the data points do not vary throughout the computation and, hence, this data structure does not need to be recomputed at each stage. Since there are typically many more data points than "query" points (i.e., centers), the

Fig. 9. Sample of images in the color quantization data set. Each pixel in the RGB color images is represented by a triple with values in the range [0, 255].

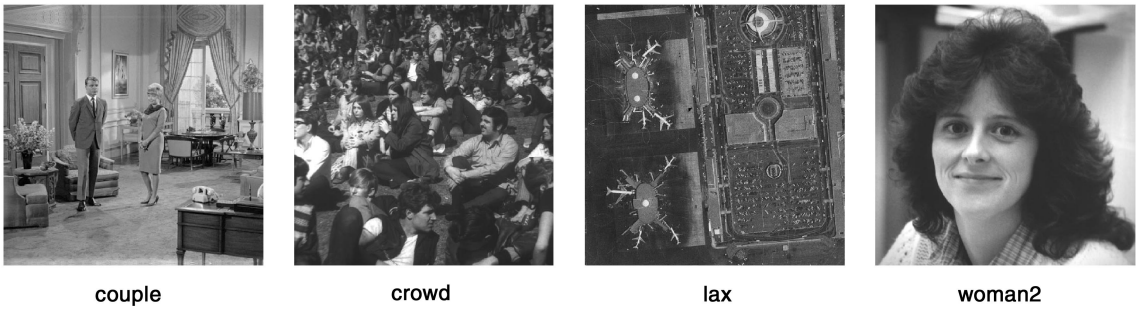


Fig. 10. Sample of images in the image compression dataset. Each pixel in these gray-scale images is represented by 8 bits. 2×2 nonoverlapping blocks were used to create 4-element vectors.

relative advantage provided by preprocessing in the above manner is greater. Our algorithm differs from existing approaches only in how nearest centers are computed, so it could be applied to the many variants of Lloyd's algorithm.

The algorithm has been implemented and the source code is available on request. We have demonstrated the practical efficiency of this algorithm both theoretically, through a data sensitive analysis, and empirically, through experiments on both synthetically generated and real data sets. The data sensitive analysis shows that the more well-separated the clusters, the faster the algorithm runs. This is subject to the assumption that the center points are indeed close to the cluster centers. Although this assumption is rather strong, our empirical analysis on synthetic data indicates that the algorithm's running time does improve dramatically as cluster separation increases. These results for both synthetic and real data sets indicate that the filtering algorithm is significantly more efficient than the other two methods that were tested. Further, they show that the algorithm scales well up to moderately high dimensions (from 10 to 20). Its performance in terms of distortions is competitive with that of the well-known BIRCH software and, by running k-means as a postprocessing step to BIRCH, it is possible to improve distortions significantly.

Fig. 11. Images in satellite image data set. One of four spectral bands of a Landsat image over Israel is shown. The image is 512×512 and each pixel is represented by 8 bits.

A natural question is whether the filtering algorithm can be improved. The most obvious source of inefficiency in the algorithm is that it passes no information from one stage to the next. Presumably, in the later stages of Lloyd's algorithm, as the centers are converging to their final positions, one would expect that the vast majority of the data points have the same closest center from one stage to the next. A good algorithm should exploit this coherence to improve the running time. A kinetic method along these lines was proposed in [31], but this algorithm is quite complex and does not provide significantly faster running time in practice. The development of a simple and practical algorithm which combines the best elements of the kinetic and filtering approaches would make a significant contribution.

ACKNOWLEDGMENTS

The authors would like to thank Azriel Rosenfeld for his comments and Hao Li for his help in running the BIRCH experiments. They also are grateful to the anonymous referees for their many valuable suggestions. This research was funded in part by the US National Science Foundation under Grant CCR-0098151 and by the US Army Research Laboratory and the US Department of Defense under

TABLE 2
Average Distortions for the Filtering Algorithm, BIRCH, and the Two Combined

Image	Dim	BIRCH	Filter	Combined
balls	8	697.10	623.20	434.70
	64	53.22	116.50	38.81
	256	28.54	28.74	11.04
kiss	8	944.50	718.60	687.42
	64	171.20	152.40	142.49
	256	75.33	58.36	53.50
Lena	8	502.60	447.20	406.68
	64	94.78	78.83	75.80
	256	34.76	29.80	27.84
ball1_h	8	531.00	716.70	501.59
	64	59.75	61.32	47.21
	256	32.00	21.05	15.61
ball1_3	8	440.20	663.20	423.89
	64	37.39	39.30	31.62
	256	17.85	11.86	9.43
ball1_5	8	490.40	421.50	469.02
	64	58.21	56.98	46.12
	256	27.53	18.20	14.98
ball1_p	8	574.60	724.20	525.84
	64	79.75	62.92	55.15
	256	47.72	21.99	18.64
ball1_s	8	570.20	535.00	541.79
	64	75.58	70.61	59.74
	256	42.22	25.05	19.69

Image	Dim	BIRCH	Filter	Combined
couple	8	766.70	594.00	631.64
	64	223.80	182.10	178.44
	256	131.20	94.21	89.71
crowd	8	539.70	464.60	458.75
	64	196.10	159.10	137.03
	256	106.20	89.02	65.66
lax	8	935.40	840.50	848.86
	64	426.20	314.10	305.41
	256	288.30	169.00	156.74
Lena4	8	367.90	368.10	340.12
	64	136.70	101.10	103.70
	256	80.89	51.79	52.87
man	8	433.20	455.10	428.93
	64	199.60	152.60	153.08
	256	148.90	79.53	77.18
woman1	8	549.10	509.40	510.47
	64	216.80	169.80	160.63
	256	147.90	88.54	85.73
woman2	8	259.80	252.80	234.11
	64	50.71	46.67	41.90
	256	31.00	24.14	20.90
Israel	8	421.9	397.31	389.06
	64	93.34	84.09	82.72
	256	44.20	34.90	34.29

Contract MDA9049-6C-1250. A preliminary version of this paper appeared in the *Proceedings of the 16th Annual ACM Symposium on Computational Geometry*, pp. 100-109, June 2000.

REFERENCES

[1] P.K. Agarwal and C.M. Procopiuc, "Exact and Approximation Algorithms for Clustering," *Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 658-667, Jan. 1998.

[2] K. Alsabti, S. Ranka, and V. Singh, "An Efficient k-means Clustering Algorithm," *Proc. First Workshop High Performance Data Mining,* Mar. 1998.

[3] S. Arora, P. Raghavan, and S. Rao, "Approximation Schemes for Euclidean k-median and Related Problems," *Proc. 30th Ann. ACM Symp. Theory of Computing,* pp. 106-113, May 1998.

[4] S. Arya and D. M. Mount, "Approximate Range Searching," *Computational Geometry: Theory and Applications,* vol. 17, pp. 135-163, 2000.

[5] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching," *J. ACM,* vol. 45, pp. 891-923, 1998.

[6] G.H. Ball and D.J. Hall, "Some Fundamental Concepts and Synthesis Procedures for Pattern Recognition Preprocessors," *Proc. Int'l Conf. Microwaves, Circuit Theory, and Information Theory,* Sept. 1964.

[7] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Comm. ACM,* vol. 18, pp. 509-517, 1975.

[8] L. Bottou and Y. Bengio, "Convergence Properties of the k-means Algorithms," *Advances in Neural Information Processing Systems 7,* G. Tesauro and D. Touretzky, eds., pp. 585-592. MIT Press, 1995.

[9] P.S. Bradley and U. Fayyad, "Refining Initial Points for K-means Clustering," *Proc. 15th Int'l Conf. Machine Learning,* pp. 91-99, 1998.

[10] P.S. Bradley, U. Fayyad, and C. Reina, "Scaling Clustering Algorithms to Large Databases," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining,* pp. 9-15, 1998.

[11] V. Capoyleas, G. Rote, and G. Woeginger, "Geometric Clusterings," *J. Algorithms,* vol. 12, pp. 341-356, 1991.

[12] J.M. Coggins and A.K. Jain, "A Spatial Filtering Approach to Texture Analysis," *Pattern Recognition Letters,* vol. 3, pp. 195-203, 1985.

[13] S. Dasgupta, "Learning Mixtures of Gaussians," *Proc. 40th IEEE Symp. Foundations of Computer Science,* pp. 634-644, Oct. 1999.

[14] S. Dasgupta and L.J. Shulman, "A Two-Round Variant of EM for Gaussian Mixtures," *Proc. 16th Conf. Uncertainty in Artificial Intelligence (UAI-2000),* pp. 152-159, June 2000.

[15] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi Tesselations: Applications and Algorithms," *SIAM Rev.,* vol. 41, pp. 637-676, 1999.

[16] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* New York: John Wiley & Sons, 1973.

[17] M. Ester, H. Kriegel, and X. Xu, "A Database Interface for Clustering in Large Spatial Databases," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining (KDD-95),* pp. 94-99, 1995.

[18] V. Faber, "Clustering and the Continuous k-means Algorithm," *Los Alamos Science,* vol. 22, pp. 138-144, 1994.

[19] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[20] W. Feller, *An Introduction to Probability Theory and Its Applications,* third ed. New York: John Wiley & Sons, 1968.

[21] E. Forgey, "Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Classification," *Biometrics,* vol. 21, p. 768, 1965.

[22] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* Boston: Academic Press, 1990.

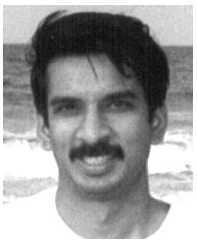
[23] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman, 1979.

[24] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression.* Boston: Kluwer Academic, 1992.

[25] M. Inaba, private communication, 1997.

[26] M. Inaba, H. Imai, and N. Katoh, "Experimental Results of a Randomized Clustering Algorithm," *Proc. 12th Ann. ACM Symp. Computational Geometry,* pp. C1-C2, May 1996.

[27] M. Inaba, N. Katoh, and H. Imai, "Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-clustering," *Proc. 10th Ann. ACM Symp. Computational Geometry*, pp. 332-339, June 1994.
[28] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data.* Englewood Cliffs, N.J.: Prentice Hall, 1988.
[29] A.K. Jain, P.W. Duin, and J. Mao, "Statistical Pattern Recognition: A Review," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37, Jan. 2000.
[30] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264-323, 1999.
[31] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, and A.Y. Wu, "Computing Nearest Neighbors for Moving Points and Applications to Clustering," *Proc. 10th Ann. ACM-SIAM Symp. Discrete Algorithms*, pp. S931-S932, Jan. 1999.
[32] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, and A.Y. Wu, "The Analysis of a Simple k-means Clustering Algorithm," Technical Report CAR-TR-937, Center for Automation Research, Univ. of Maryland, College Park, Jan. 2000.
[33] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, and A.Y. Wu, "The Analysis of a Simple k-means Clustering Algorithm," *Proc. 16th Ann. ACM Symp. Computational Geometry*, pp. 100-109, June 2000.
[34] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* New York: John Wiley & Sons, 1990.
[35] T. Kohonen, *Self-Organization and Associative Memory*, third ed. New York: Springer-Verlag, 1989.
[36] S. Kolliopoulos and S. Rao, "A Nearly Linear-Time Approximation Scheme for the Euclidean k-median Problem," *Proc. Seventh Ann. European Symp. Algorithms*, J. Nesetril, ed., pp. 362-371, July 1999.
[37] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Trans. Information Theory*, vol. 28, 129-137, 1982.
[38] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. Fifth Berkeley Symp. Math. Statistics and Probability*, vol. 1, pp. 281-296, 1967.
[39] S. Maneewongvatana and D.M. Mount, "Analysis of Approximate Nearest Neighbor Searching with Clustered Point Sets," *Proc. Workshop Algorithm Eng. and Experiments (ALENEX '99)*, Jan. 1999. Available from: http://ftp.cs.umd.edu-pub/faculty/mount/Papers/dimacs99.ps.gz.
[40] O.L. Mangasarian, "Mathematical Programming in Data Mining," *Data Mining and Knowledge Discovery*, vol. 1, pp. 183-201, 1997.
[41] J. Matousek, "On Approximate Geometric k-clustering," *Discrete and Computational Geometry*, vol. 24, pp. 61-84, 2000.
[42] A. Moore, "Very Fast EM-Based Mixture Model Clustering Using Multiresolution kd-Trees," *Proc. Conf. Neural Information Processing Systems*, 1998.
[43] D.M. Mount and S. Arya, "ANN: A Library for Approximate Nearest Neighbor Searching," *Proc. Center for Geometric Computing Second Ann. Fall Workshop Computational Geometry*, 1997. (available from http://www.cs.umd.edu/~mount/ANN.)
[44] R.T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," *Proc. 20th Int'l Conf. Very Large Databases*, pp. 144-155, Sept. 1994.
[45] D. Pelleg and A. Moore, "Accelerating Exact k-means Algorithms with Geometric Reasoning," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 277-281, Aug. 1999.
[46] D. Pelleg and A. Moore, "x-means: Extending k-means with Efficient Estimation of the Number of Clusters," *Proc. 17th Int'l Conf. Machine Learning*, July 2000.
[47] D. Pollard, "A Centeral Limit Theorem for k-means Clustering," *Annals of Probability*, vol. 10, pp. 919-926, 1982.
[48] F.P. Preparata, M.I. Shamos, *Computational Geometry: An Introduction*, third ed. Springer-Verlag, 1990.
[49] S.Z. Selim and M.A. Ismail, "K-means-type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, pp. 81-87, 1984.
[50] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: A New Data Clustering Algorithm and Its Applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141-182, 1997. (software available from www.cs.wisc.edu/~vganti/birchcode).



Tapas Kanungo received the PhD degree from the University of Washington, Seattle. Currently, he is a research staff member at the IBM Almaden Research Center, San Jose, California. Prior to working at IBM, he served as a codirector of the Language and Media Processing Lab at the University of Maryland, College Park. His current interests are in information extraction and retrieval and document analysis. He is senior member of the IEEE and the IEEE Computer Society.

David M. Mount received the PhD degree from Purdue University in computer science in 1983. He is a professor in the Department of Computer Science at the University of Maryland with a joint appointment at the Institute for Advanced Computer Studies. His primary research interests include the design, analysis, and implementation of data structures and algorithms for geometric problems, particularly problems with applications in image processing, pattern recognition, information retrieval, and computer graphics. He is an associate editor for *Pattern Recognition*. He is a member of the IEEE.

Nathan S. Netanyahu received BSc and MSc degrees in electrical engineering from the Technion, Israel Institute of Technology, and the MSc and PhD degrees in computer science from the University of Maryland, College Park. He is currently a senior lecturer in the Department of Mathematics and Computer Science at Bar-Ilan University and is also affiliated with the Center for Automation Research, University of Maryland, College Park. Dr. Netanyahu's main research interests are in the areas of algorithm design and analysis, computational geometry, image processing, pattern recognition, remote sensing, and robust statistical estimation. He currently serves as an associate editor for *Pattern Recognition*. He is a member of the IEEE.

Christine D. Piatko received the BA degree in computer science and mathematics from New York University in 1986 and the MS and PhD degrees from Cornell University in 1989 and 1993, respectively. She is currently a computer science researcher at The Johns Hopkins University Applied Physics Laboratory. Her research interests include computational geometry and information retrieval.

Ruth Silverman received the PhD degree in mathematics from the University of Washington in 1970. She is a retired professor from the Department of Computer Science at the University of the District of Columbia and she is a visiting professor at the Center for Automation Research at the University of Maryland, College Park.

Angela Y. Wu received the PhD degree in computer science from the University of Maryland at College Park in 1978. She is a professor in the Department of Computer Science and Information Systems, American University, Washington D.C., where she has taught since 1980. She is a senior member of the IEEE.

A Density-Based Algorithm for Discovering Clusters
A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
{ester | kriegel | sander | xwxu}@informatik.uni-muenchen.de

Abstract

Clustering algorithms are attractive for the task of class identification in spatial databases. However, the application to large spatial databases rises the following requirements for clustering algorithms: minimal requirements of domain knowledge to determine the input parameters, discovery of clusters with arbitrary shape and good efficiency on large databases. The well-known clustering algorithms offer no solution to the combination of these requirements. In this paper, we present the new clustering algorithm DBSCAN relying on a density-based notion of clusters which is designed to discover clusters of arbitrary shape. DBSCAN requires only one input parameter and supports the user in determining an appropriate value for it. We performed an experimental evaluation of the effectiveness and efficiency of DBSCAN using synthetic data and real data of the SEQUOIA 2000 benchmark. The results of our experiments demonstrate that (1) DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS, and that (2) DBSCAN outperforms CLARANS by a factor of more than 100 in terms of efficiency.

Keywords: Clustering Algorithms, Arbitrary Shape of Clusters, Efficiency on Large Spatial Databases, Handling Nlj4-275oise.

1. Introduction

Numerous applications require the management of *spatial* data, i.e. data related to space. *Spatial Database Systems (SDBS)* (Gueting 1994) are database systems for the management of spatial data. Increasingly large amounts of data are obtained from satellite images, X-ray crystallography or other automatic equipment. Therefore, automated knowledge discovery becomes more and more important in spatial databases.

Several tasks of *knowledge discovery in databases* (KDD) have been defined in the literature (Matheus, Chan & Piatetsky-Shapiro 1993). The task considered in this paper is *class identification*, i.e. the grouping of the objects of a database into meaningful subclasses. In an earth observation database, e.g., we might want to discover classes of houses along some river.

Clustering algorithms are attractive for the task of class identification. However, the application to large spatial databases rises the following requirements for clustering algorithms:

(1) Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.

(2) Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.

(3) Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

The well-known clustering algorithms offer no solution to the combination of these requirements. In this paper, we present the new clustering algorithm DBSCAN. It requires only one input parameter and supports the user in determining an appropriate value for it. It discovers clusters of arbitrary shape. Finally, DBSCAN is efficient even for large spatial databases. The rest of the paper is organized as follows. We discuss clustering algorithms in section 2 evaluating them according to the above requirements. In section 3, we present our notion of clusters which is based on the concept of density in the database. Section 4 introduces the algorithm DBSCAN which discovers such clusters in a spatial database. In section 5, we performed an experimental evaluation of the effectiveness and efficiency of DBSCAN using synthetic data and data of the SEQUOIA 2000 benchmark. Section 6 concludes with a summary and some directions for future research.

2. Clustering Algorithms

There are two basic types of clustering algorithms (Kaufman & Rousseeuw 1990): partitioning and hierarchical algorithms. *Partitioning algorithms* construct a partition of a database D of n objects into a set of k clusters. k is an input parameter for these algorithms, i.e some domain knowledge is required which unfortunately is not available for many applications. The partitioning algorithm typically starts with an initial partition of D and then uses an iterative control strategy to optimize an objective function. Each cluster is represented by the gravity center of the cluster *(k-means algorithms)* or by one of the objects of the cluster located near its center *(k-medoid algorithms)*. Consequently, partitioning algorithms use a two-step procedure. First, determine k representatives minimizing the objective function. Second, assign each object to the cluster with its representative "closest" to the considered object. The second step implies that a partition is equivalent to a voronoi diagram and each cluster is contained in one of the voronoi cells. Thus, the shape of all

clusters found by a partitioning algorithm is convex which is very restrictive.

Ng & Han (1994) explore partitioning algorithms for KDD in spatial databases. An algorithm called CLARANS (Clustering Large Applications based on RANdomized Search) is introduced which is an improved k-medoid method. Compared to former k-medoid algorithms, CLARANS is more effective and more efficient. An experimental evaluation indicates that CLARANS runs efficiently on databases of thousands of objects. Ng & Han (1994) also discuss methods to determine the "natural" number k_{nat} of clusters in a database. They propose to run CLARANS once for each k from 2 to n. For each of the discovered clusterings the silhouette coefficient (Kaufman & Rousseeuw 1990) is calculated, and finally, the clustering with the maximum silhouette coefficient is chosen as the "natural" clustering. Unfortunately, the run time of this approach is prohibitive for large n, because it implies O(n) calls of CLARANS.

CLARANS assumes that all objects to be clustered can reside in main memory at the same time which does not hold for large databases. Furthermore, the run time of CLARANS is prohibitive on large databases. Therefore, Ester, Kriegel &Xu (1995) present several focusing techniques which address both of these problems by focusing the clustering process on the relevant parts of the database. First, the focus is small enough to be memory resident and second, the run time of CLARANS on the objects of the focus is significantly less than its run time on the whole database.

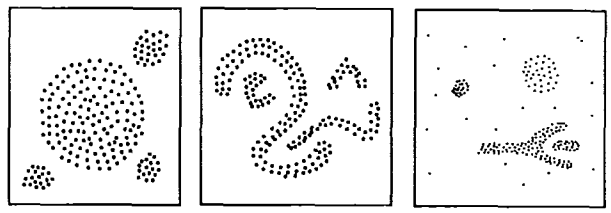
Hierarchical algorithms create a hierarchical decomposition of *D*. The hierarchical decomposition is represented by a *dendrogram*, a tree that iteratively splits *D* into smaller subsets until each subset consists of only one object. In such a hierarchy, each node of the tree represents a cluster of *D*. The dendrogram can either be created from the leaves up to the root (*agglomerative approach*) or from the root down to the leaves (*divisive approach*) by merging or dividing clusters at each step. In contrast to partitioning algorithms, hierarchical algorithms do not need *k* as an input. However, a *termination condition* has to be defined indicating when the merge or division process should be terminated. One example of a termination condition in the agglomerative approach is the critical distance D_{min} between all the clusters of *Q*.

So far, the main problem with hierarchical clustering algorithms has been the difficulty of deriving appropriate parameters for the termination condition, e.g. a value of D_{min} which is small enough to separate all "natural" clusters and, at the same time large enough such that no cluster is split into two parts. Recently, in the area of signal processing the hierarchical algorithm Ejcluster has been presented (García, Fdez-Valdivia, Cortijo & Molina 1994) automatically deriving a termination condition. Its key idea is that two points belong to the same cluster if you can walk from the first point to the second one by a "sufficiently small" step. Ejcluster follows the divisive approach. It does not require any input of domain knowledge. Furthermore, experiments show that it is very effective in discovering non-convex clusters. However, the computational cost of Ejcluster is $O(n^2)$ due to the distance calculation for each pair of points. This is acceptable for applications such as character recognition with moderate values for n, but it is prohibitive for applications on large databases.

Jain (1988) explores a density based approach to identify clusters in k-dimensional point sets. The data set is partitioned into a number of nonoverlapping cells and histograms are constructed. Cells with relatively high frequency counts of points are the potential cluster centers and the boundaries between clusters fall in the "valleys" of the histogram. This method has the capability of identifying clusters of any shape. However, the space and run-time requirements for storing and searching multidimensional histograms can be enormous. Even if the space and run-time requirements are optimized, the performance of such an approach crucially depends on the size of the cells.

3. A Density Based Notion of Clusters

When looking at the sample sets of points depicted in figure 1, we can easily and unambiguously detect clusters of points and noise points not belonging to any of those clusters.



database 1 **database 2** **database 3**

figure 1: Sample databases

The main reason why we recognize the clusters is that within each cluster we have a typical density of points which is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of the clusters.

In the following, we try to formalize this intuitive notion of "clusters" and "noise" in a database *D* of points of some k-dimensional space *S*. Note that both, our notion of clusters and our algorithm DBSCAN, apply as well to 2D or 3D Euclidean space as to some high dimensional feature space. The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, i.e. the density in the neighborhood has to exceed some threshold. The shape of a neighborhood is determined by the choice of a distance function for two points p and q, denoted by *dist(p,q)*. For instance, when using the Manhattan distance in 2D space, the shape of the neighborhood is rectangular. Note, that our approach works with any distance function so that an appropriate function can be chosen for some given application. For the purpose of proper visualization, all examples will be in 2D space using the Euclidean distance.

Definition 1: (Eps-neighborhood of a point) The *Eps-neighborhood* of a point p, denoted by $N_{Eps}(p)$, is defined by $N_{Eps}(p) = \{q \in D \mid dist(p,q) \leq Eps\}$.

A naive approach could require for each point in a cluster that there are at least a minimum number (*MinPts*) of points in an Eps-neighborhood of that point. However, this ap-

proach fails because there are two kinds of points in a cluster, points inside of the cluster (*core points*) and points on the border of the cluster (*border points*). In general, an Eps-neighborhood of a border point contains significantly less points than an Eps-neighborhood of a core point. Therefore, we would have to set the minimum number of points to a relatively low value in order to include all points belonging to the same cluster. This value, however, will not be characteristic for the respective cluster - particularly in the presence of noise. Therefore, we require that for every point p in a cluster C there is a point q in C so that p is inside of the Eps-neighborhood of q and $N_{Eps}(q)$ contains at least MinPts points. This definition is elaborated in the following.

Definition 2: (directly density-reachable) A point p is *directly density-reachable* from a point q wrt. Eps, MinPts if

1) $p \in N_{Eps}(q)$ and

2) $|N_{Eps}(q)| \geq$ MinPts (core point condition).

Obviously, directly density-reachable is symmetric for pairs of core points. In general, however, it is not symmetric if one core point and one border point are involved. Figure 2 shows the asymmetric case.

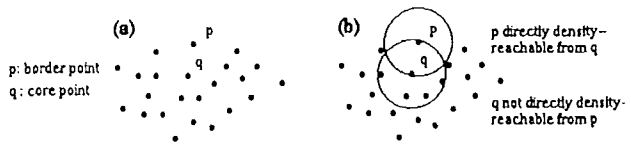


figure 2: core points and border points

Definition 3: (density-reachable) A point p is *density-reachable* from a point q wrt. Eps and MinPts if there is a chain of points $p_1, ..., p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i.

Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, but it is not symmetric. Figure 3 depicts the relations of some sample points and, in particular, the asymmetric case. Although not symmetric in general, it is obvious that density-reachability is symmetric for core points.

Two border points of the same cluster C are possibly not density reachable from each other because the core point condition might not hold for both of them. However, there must be a core point in C from which both border points of C are density-reachable. Therefore, we introduce the notion of density-connectivity which covers this relation of border points.

Definition 4: (density-connected) A point p is *density-connected* to a point q wrt. Eps and MinPts if there is a point o such that both, p and q are density-reachable from o wrt. Eps and MinPts.

Density-connectivity is a symmetric relation. For density reachable points, the relation of density-connectivity is also reflexive (c.f. figure 3).

Now, we are able to define our density-based notion of a cluster. Intuitively, a cluster is defined to be a set of density-connected points which is maximal wrt. density-reachability. Noise will be defined relative to a given set of clusters. Noise is simply the set of points in *D* not belonging to any of its clusters.

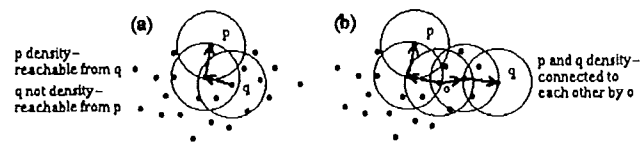


figure 3: density-reachability and density-connectivity

Definition 5: (cluster) Let *D* be a database of points. A *cluster C* wrt. Eps and MinPts is a non-empty subset of *D* satisfying the following conditions:

1) $\forall$ p, q: if $p \in C$ and q is density-reachable from p wrt. Eps and MinPts, then $q \in C$. (Maximality)

2) $\forall$ p, q $\in$ C: p is density-connected to q wrt. EPS and MinPts. (Connectivity)

Definition 6: (noise) Let $C_1, \ldots, C_k$ be the clusters of the database *D* wrt. parameters Eps_i and $MinPts_i$, i = 1, . . ., k. Then we define the *noise* as the set of points in the database *D* not belonging to any cluster C_i, i.e. noise = $\{p \in D \mid \forall i: p \notin C_i\}$.

Note that a cluster C wrt. Eps and MinPts contains at least MinPts points because of the following reasons. Since C contains at least one point p, p must be density-connected to itself via some point o (which may be equal to p). Thus, at least o has to satisfy the core point condition and, consequently, the Eps-Neighborhood of o contains at least MinPts points.

The following lemmata are important for validating the correctness of our clustering algorithm. Intuitively, they state the following. Given the parameters Eps and MinPts, we can discover a cluster in a two-step approach. First, choose an arbitrary point from the database satisfying the core point condition as a seed. Second, retrieve all points that are density-reachable from the seed obtaining the cluster containing the seed.

Lemma 1: Let p be a point in *D* and $|N_{Eps}(p)| \geq$ MinPts. Then the set $O = \{o \mid o \in D$ and o is density-reachable from p wrt. Eps and MinPts$\}$ is a cluster wrt. Eps and MinPts.

It is not obvious that a cluster *C* wrt. Eps and MinPts is uniquely determined by *any* of its core points. However, each point in *C* is density-reachable from any of the core points of *C* and, therefore, a cluster *C* contains exactly the points which are density-reachable from an arbitrary core point of *C*.

Lemma 2: Let *C* be a cluster wrt. Eps and MinPts and let p be any point in *C* with $|N_{Eps}(p)| \geq$ MinPts. Then *C* equals to the set $O = \{o \mid o$ is density-reachable from p wrt. Eps and MinPts$\}$.

4. DBSCAN: Density Based Spatial Clustering of Applications with Noise

In this section, we present the algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) which is designed to discover the clusters and the noise in a spatial database according to definitions 5 and 6. Ideally, we would have to know the appropriate parameters Eps and MinPts of each cluster and at least one point from the respective cluster. Then, we could retrieve all points that are density-reachable from the given point using the correct parameters. But

there is no easy way to get this information in advance for all clusters of the database. However, there is a simple and effective heuristic (presented in section section 4.2) to determine the parameters Eps and MinPts of the "thinnest", i.e. least dense, cluster in the database. Therefore, DBSCAN uses global values for Eps and MinPts, i.e. the same values for all clusters. The density parameters of the "thinnest" cluster are good candidates for these global parameter values specifying the lowest density which is not considered to be noise.

4.1 The Algorithm

To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p wrt. Eps and MinPts. If p is a core point, this procedure yields a cluster wrt. Eps and MinPts (see Lemma 2). If p is a border point, no points are density-reachable from p and DBSCAN visits the next point of the database.

Since we use global values for Eps and MinPts, DBSCAN may merge two clusters according to definition 5 into one cluster, if two clusters of different density are "close" to each other. Let the *distance between two sets of points* S_1 and S_2 be defined as $dist(S_1, S_2) = \min\{dist(p,q) \mid p \in S_1, q \in S_2\}$. Then, two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than Eps. Consequently, a recursive call of DBSCAN may be necessary for the detected clusters with a higher value for MinPts. This is, however, no disadvantage because the recursive application of DBSCAN yields an elegant and very efficient basic algorithm. Furthermore, the recursive clustering of the points of a cluster is only necessary under conditions that can be easily detected.

In the following, we present a basic version of DBSCAN omitting details of data types and generation of additional information about clusters:

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point,
            ClusterId, Eps, MinPts) THEN
        ClusterId := nextId(ClusterId)
      END IF
    END IF
  END FOR
END; // DBSCAN
```

`SetOfPoints` is either the whole database or a discovered cluster from a previous run. `Eps` and `MinPts` are the global density parameters determined either manually or according to the heuristics presented in section 4.2. The function `SetOfPoints.get(i)` returns the i-th element of `SetOfPoints`. The most important function used by DBSCAN is `ExpandCluster` which is presented below:

```
ExpandCluster(SetOfPoints, Point, ClId, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeClId(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeClIds(seeds,ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                        Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId
              IN {UNCLASSIFIED, NOISE} THEN
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP,ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

A call of `SetOfPoints.regionQuery(Point,Eps)` returns the Eps-Neighborhood of `Point` in `SetOfPoints` as a list of points. Region queries can be supported efficiently by spatial access methods such as R*-trees (Beckmann et al. 1990) which are assumed to be available in a SDBS for efficient processing of several types of spatial queries (Brinkhoff et al. 1994). The height of an R*-tree is O(log n) for a database of n points in the worst case and a query with a "small" query region has to traverse only a limited number of paths in the R*-tree. Since the Eps-Neighborhoods are expected to be small compared to the size of the whole data space, the average run time complexity of a single region query is O(log n). For each of the n points of the database, we have at most one region query. Thus, the average run time complexity of DBSCAN is O(n * log n).

The `ClId` (clusterId) of points which have been marked to be `NOISE` may be changed later, if they are density-reachable from some other point of the database. This happens for border points of a cluster. Those points are not added to the `seeds`-list because we already know that a point with a ClId of `NOISE` is not a core point. Adding those points to `seeds` would only result in additional region queries which would yield no new answers.

If two clusters C_1 and C_2 are very close to each other, it might happen that some point p belongs to both, C_1 and C_2. Then p must be a border point in both clusters because otherwise C_1 would be equal to C_2 since we use global parame-

ters. In this case, point p will be assigned to the cluster discovered first. Except from these rare situations, the result of DBSCAN is independent of the order in which the points of the database are visited due to Lemma 2.

4.2 Determining the Parameters Eps and MinPts

In this section, we develop a simple but effective heuristic to determine the parameters Eps and MinPts of the "thinnest" cluster in the database. This heuristic is based on the following observation. Let d be the distance of a point p to its k-th nearest neighbor, then the d-neighborhood of p contains exactly k+1 points for almost all points p. The d-neighborhood of p contains more than k+1 points only if several points have exactly the same distance d from p which is quite unlikely. Furthermore, changing k for a point in a cluster does not result in large changes of d. This only happens if the k-th nearest neighbors of p for k= 1, 2, 3, . . . are located approximately on a straight line which is in general not true for a point in a cluster.

For a given k we define a function *k-dist* from the database *D* to the real numbers, mapping each point to the distance from its k-th nearest neighbor. When sorting the points of the database in descending order of their k-dist values, the graph of this function gives some hints concerning the density distribution in the database. We call this graph the *sorted k-dist graph*. If we choose an arbitrary point p, set the parameter Eps to k-dist(p) and set the parameter MinPts to k, all points with an equal or smaller k-dist value will be core points. If we could find a *threshold point* with the maximal k-dist value in the "thinnest" cluster of *D* we would have the desired parameter values. The threshold point is the first point in the first "valley" of the sorted k-dist graph (see figure 4). All points with a higher k-dist value (left of the threshold) are considered to be noise, all other points (right of the threshold) are assigned to some cluster.

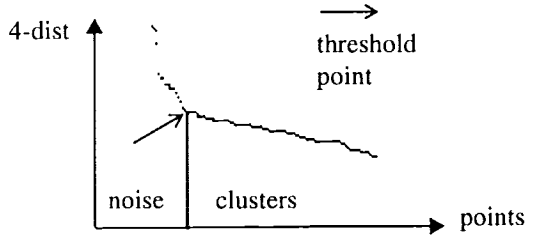


figure 4: sorted 4-dist graph for sample database 3

In general, it is very difficult to detect the first "valley" automatically, but it is relatively simple for a user to see this valley in a graphical representation. Therefore, we propose to follow an interactive approach for determining the threshold point.

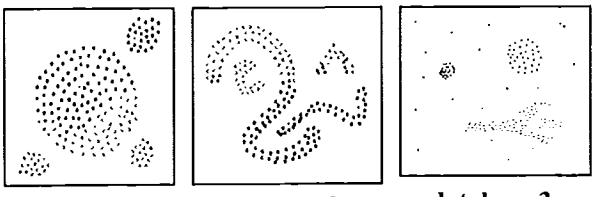
DBSCAN needs two parameters, Eps and MinPts. However, our experiments indicate that the k-dist graphs for $k > 4$ do not significantly differ from the 4-dist graph and, furthermore, they need considerably more computation. Therefore, we eliminate the parameter MinPts by setting it to 4 for all databases (for 2-dimensional data). We propose the following interactive approach for determining the parameter Eps of DBSCAN:

- The system computes and displays the 4-dist graph for the database.
- If the user can estimate the percentage of noise, this percentage is entered and the system derives a proposal for the threshold point from it.
- The user either accepts the proposed threshold or selects another point as the threshold point. The 4-dist value of the threshold point is used as the Eps value for DBSCAN.

5. Performance Evaluation

In this section, we evaluate the performance of DBSCAN. We compare it with the performance of CLARANS because this is the first and only clustering algorithm designed for the purpose of KDD. In our future research, we will perform a comparison with classical density based clustering algorithms. We have implemented DBSCAN in C++ based on an implementation of the R*-tree (Beckmann et al. 1990). All experiments have been run on HP 735 / 100 workstations. We have used both synthetic sample databases and the database of the SEQUOIA 2000 benchmark.

To compare DBSCAN with CLARANS in terms of effectivity (accuracy), we use the three synthetic sample databases which are depicted in figure 1. Since DBSCAN and CLARANS are clustering algorithms of different types, they have no common quantitative measure of the classification accuracy. Therefore, we evaluate the accuracy of both algorithms by visual inspection. In sample database 1, there are four ball-shaped clusters of significantly differing sizes. Sample database 2 contains four clusters of nonconvex shape. In sample database 3, there are four clusters of different shape and size with additional noise. To show the results of both clustering algorithms, we visualize each cluster by a different color (see www availability after section 6). To give CLARANS some advantage, we set the parameter *k* to 4 for these sample databases. The clusterings discovered by CLARANS are depicted in figure 5.

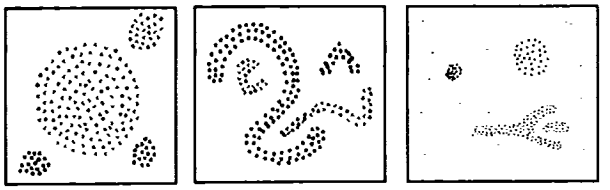


database 1 **database 2** **database 3**

figure 5: Clusterings discovered by CLARANS

For DBSCAN, we set the noise percentage to 0% for sample databases 1 and 2, and to 10% for sample database 3, respectively. The clusterings discovered by DBSCAN are depicted in figure 6.

DBSCAN discovers all clusters (according to definition 5) and detects the noise points (according to definition 6) from all sample databases. CLARANS, however, splits clusters if they are relatively large or if they are close to some other cluster. Furthermore, CLARANS has no explicit notion of noise. Instead, all points are assigned to their closest medoid.



database 1 database 2 database 3

figure 6: Clusterings discovered by DBSCAN

To test the efficiency of DBSCAN and CLARANS, we use the SEQUOIA 2000 benchmark data. The SEQUOIA 2000 benchmark database (Stonebraker et al. 1993) uses real data sets that are representative of Earth Science tasks. There are four types of data in the database: raster data, point data, polygon data and directed graph data. The point data set contains 62,584 Californian names of landmarks, extracted from the US Geological Survey's Geographic Names Information System, together with their location. The point data set occupies about 2.1 M bytes. Since the run time of CLARANS on the whole data set is very high, we have extracted a series of subsets of the SEQUIOA 2000 point data set containing from 2% to 20% representatives of the whole set. The run time comparison of DBSCAN and CLARANS on these databases is shown in table 1.

Table 1: run time in seconds

number of points	1252	2503	3910	5213	6256
DBSCAN	3.1	6.7	11.3	16.0	17.8
CLARANS	758	3026	6845	11745	18029
number of points	7820	8937	10426	12512	
DBSCAN	24.5	28.2	32.7	41.7	
CLARANS	29826	39265	60540	80638	

The results of our experiments show that the run time of DBSCAN is slightly higher than linear in the number of points. The run time of CLARANS, however, is close to quadratic in the number of points. The results show that DBSCAN outperforms CLARANS by a factor of between 250 and 1900 which grows with increasing size of the database.

6. Conclusions

Clustering algorithms are attractive for the task of class identification in spatial databases. However, the well-known algorithms suffer from severe drawbacks when applied to large spatial databases. In this paper, we presented the clustering algorithm DBSCAN which relies on a density-based notion of clusters. It requires only one input parameter and supports the user in determining an appropriate value for it. We performed a performance evaluation on synthetic data and on real data of the SEQUOIA 2000 benchmark. The results of these experiments demonstrate that DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS. Furthermore, the experiments have shown that DBSCAN outperforms CLARANS by a factor of at least 100 in terms of efficiency.

Future research will have to consider the following issues. First, we have only considered point objects. Spatial databases, however, may also contain extended objects such as polygons. We have to develop a definition of the density in an Eps-neighborhood in polygon databases for generalizing DBSCAN. Second, applications of DBSCAN to high dimensional feature spaces should be investigated. In particular, the shape of the k-dist graph in such applications has to be explored.

WWW Availability

A version of this paper in larger font, with large figures and clusterings in color is available under the following URL: `http://www.dbs.informatik.uni-muenchen.de/dbs/project/publikationen/veroeffentlichungen.html`.

References

Beckmann N., Kriegel H.-P., Schneider R, and Seeger B. 1990. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

Brinkhoff T., Kriegel H.-P., Schneider R., and Seeger B. 1994 Efficient Multi-Step Processing of Spatial Joins, Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 1994, pp. 197-208.

Ester M., Kriegel H.-P., and Xu X. 1995. A Database Interface for Clustering in Large Spatial Databases, Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, Montreal, Canada, 1995, AAAI Press, 1995.

García J.A., Fdez-Valdivia J., Cortijo F. J., and Molina R. 1994. A Dynamic Approach for Clustering Data. *Signal Processing*, Vol. 44, No. 2, 1994, pp. 181-196.

Gueting R.H. 1994. An Introduction to Spatial Database Systems. *The VLDB Journal* 3(4): 357-399.

Jain Anil K. 1988. *Algorithms for Clustering Data.* Prentice Hall.

Kaufman L., and Rousseeuw P.J. 1990. *Finding Groups in Data: an Introduction to Cluster Analysis.* John Wiley & Sons.

Matheus C.J.; Chan P.K.; and Piatetsky-Shapiro G. 1993. Systems for Knowledge Discovery in Databases, *IEEE Transactions on Knowledge and Data Engineering* 5(6): 903-913.

Ng R.T., and Han J. 1994. Efficient and Effective Clustering Methods for Spatial Data Mining, Proc. 20th Int. Conf. on Very Large Data Bases, 144-155. Santiago, Chile.

Stonebraker M., Frew J., Gardels K., and Meredith J.1993. The SEQUOIA 2000 Storage Benchmark, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, 1993, pp. 2-11.

BIRCH: An Efficient Data Clustering Method for Very Large Databases

Tian Zhang
Computer Sciences Dept.
Univ. of Wisconsin-Madison
zhang@cs.wisc.edu

Raghu Ramakrishnan
Computer Sciences Dept.
Univ. of Wisconsin-Madison
raghu@cs.wisc.edu

Miron Livny*
Computer Sciences Dept.
Univ. of Wisconsin-Madison
miron@cs.wisc.edu

Abstract

Finding useful patterns in large datasets has attracted considerable interest recently, and one of the most widely studied problems in this area is the identification of *clusters*, or densely populated regions, in a multi-dimensional dataset. Prior work does not adequately address the problem of large datasets and minimization of I/O costs.

This paper presents a data clustering method named *BIRCH* (Balanced Iterative Reducing and Clustering using Hierarchies), and demonstrates that it is especially suitable for very large databases. *BIRCH* incrementally and dynamically clusters incoming multi-dimensional metric data points to try to produce the best quality clustering with the available resources (i.e., available memory and time constraints). *BIRCH* can typically find a good clustering with a single scan of the data, and improve the quality further with a few additional scans. *BIRCH* is also the first clustering algorithm proposed in the database area to handle "noise" (data points that are not part of the underlying pattern) effectively.

We evaluate *BIRCH*'s time/space efficiency, data input order sensitivity, and clustering quality through several experiments. We also present a performance comparisons of *BIRCH* versus *CLARANS*, a clustering method proposed recently for large datasets, and show that *BIRCH* is consistently superior.

1 Introduction

In this paper, we examine *data clustering*, which is a particular kind of data mining problem. Given a large set of multi-dimensional data points, the data space is usually not uniformly occupied. *Data clustering* identifies the sparse and the crowded places, and hence discovers the overall distribution patterns of the dataset. Besides, the derived clusters can be visualized more efficiently and effectively than the original dataset[Lee81, DJ80].

*This research has been supported by NSF Grant IRI-9057562 and NASA Grant 144-EC78.

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

Generally, there are two types of attributes involved in the data to be clustered: *metric* and *nonmetric*[1]. In this paper, we consider metric attributes, as in most of the Statistics literature, where the clustering problem is formalized as follows: *Given the desired number of clusters K and a dataset of N points, and a distance-based measurement function (e.g., the weighted total/average distance between pairs of points in clusters), we are asked to find a partition of the dataset that minimizes the value of the measurement function.* This is a *nonconvex discrete* [KR90] optimization problem. Due to an abundance of local minima, there is typically no way to find a global minimal solution without trying all possible partitions.

We adopt the problem definition used in Statistics, but with an additional, database-oriented constraint: *The amount of memory available is limited (typically, much smaller than the data set size) and we want to minimize the time required for I/O.* A related point is that it is desirable to be able to take into account the amount of *time* that a user is willing to wait for the results of the clustering algorithm.

We present a clustering method named *BIRCH* and demonstrate that it is especially suitable for very large databases. Its I/O cost is linear in the size of the dataset: a *single scan* of the dataset yields a good clustering, and one or more additional passes can (optionally) be used to improve the quality further.

By evaluating *BIRCH*'s time/space efficiency, data input order sensitivity, and clustering quality, and comparing with other existing algorithms through experiments, we argue that *BIRCH* is the best available clustering method for very large databases. *BIRCH*'s architecture also offers opportunities for parallelism, and for interactive or dynamic performance tuning based on knowledge about the dataset, gained over the course of the execution. Finally, *BIRCH* is the first clustering al-

[1] Informally, a *metric* attribute is an attribute whose values satisfy the requirements of *Euclidian* space, i.e., self identity (for any X, $X = X$) and triangular inequality (there exists a distance definition such that for any X_1,X_2,X_3, $d(X_1, X_2) + d(X_2, X_3) \geq d(X_1, X_3)$).

gorithm proposed in the database area that addresses *outliers* (intuitively, data points that should be regarded as "noise") and proposes a plausible solution.

1.1 Outline of Paper

The rest of the paper is organized as follows. Sec. 2 surveys related work and summarizes *BIRCH*'s contributions Sec. 3 presents some background material. Sec. 4 introduces the concepts of clustering feature (CF) and CF tree, which are central to *BIRCH*. The details of *BIRCH* algorithm is described in Sec. 5, and a preliminary performance study of *BIRCH* is presented in Sec. 6. Finally our conclusions and directions for future research are presented in Sec. 7.

2 Summary of Relevant Research

Data clustering has been studied in the Statistics [DH73, DJ80, Lee81, Mur83], Machine Learning [CKS88, Fis87, Fis95, Leb87] and Database [NH94, EKX95a, EKX95b] communities with different methods and different emphases. Previous approaches, probability-based (like most approaches in Machine Learning) or distance-based (like most work in Statistics) , do not adequately consider the case that the dataset can be too large to fit in main memory. In particular, they do not recognize that the problem must be viewed in terms of how to work with a limited resources (e.g., memory that is typically, much smaller than the size of the dataset) to do the clustering as accurately as possible while keeping the I/O costs low.

Probability-based approaches: They typically [Fis87, CKS88] make the assumption that probability distributions on separate attributes are statistically independent of each other. In reality, this is far from true. Correlation between attributes exists, and sometimes this kind of correlation is exactly what we are looking for. The probability representations of clusters make updating and storing the clusters very expensive, especially if the attributes have a large number of values because their complexities are dependent not only on the number of attributes, but also on the number of values for each attribute. A related problem is that often (e.g., [Fis87]), the probability-based tree that is built to identify clusters is not height-balanced. For skewed input data, this may cause the performance to degrade dramatically.

Distance-based approaches: They assume that all data points are given in advance and can be scanned frequently. They totally or partially ignore the fact that not all data points in the dataset are equally important with respect to the clustering purpose, and that data points which are close and dense should be considered collectively instead of individually. They are *global* or *semi-global* methods at the granularity of data points. That is, for each clustering decision, they inspect all data points or all currently existing clusters equally no matter how close or far away they are, and they use global measurements, which require scanning all data points or all currently existing clusters. Hence none of them have linear time scalability with stable quality.

For example, using *exhaustive enumeration (EE)*, there are approximately $K^N/K!$ [DH73] ways of partitioning a set of N data points into K subsets. So in practice, though it can find the global minimum, it is infeasible except when N and K are extremely small. *Iterative optimization (IO)* [DH73, KR90] starts with an initial partition, then tries all possible moving or swapping of data points from one group to another to see if such a moving or swapping improves the value of the measurement function. It can find a local minimum, but the quality of the local minimum is very sensitive to the initially selected partition, and the worst case time complexity is still exponential. *Hierarchical clustering (HC)* [DH73, KR90, Mur83] does not try to find "best" clusters, but keeps merging the closest pair (or splitting the farthest pair) of objects to form clusters. With a reasonable distance measurement, the best time complexity of a practical *HC* algorithm is $O(N^2)$. So it is still unable to scale well with large N.

Clustering has been recognized as a useful spatial data mining method recently. [NH94] presents *CLARANS* that is based on randomized search, and proposes that *CLARANS* outperforms traditional clustering algorithms in Statistics. In *CLARANS*, a cluster is represented by its *medoid*, or the most centrally located data point in the cluster The clustering process is formalized as searching a graph in which each node is a K-partition represented by a set of K medoids, and two nodes are neighbors if they only differ by one medoid. *CLARANS* starts with a randomly selected node. For the current node, it checks at most the *maxneighbor* number of neighbors randomly, and if a better neighbor is found, it moves to the neighbor and continues; otherwise it records the current node as a *local minimum*, and restarts with a new randomly selected node to search for another *local minimum*. *CLARANS* stops after the *numlocal* number of the so-called *local minima* have been found , and returns the best of these.

CLARANS suffers from the same drawbacks as the above *IO* method wrt. efficiency In addition, it may not find a real local minimum due to the searching trimming controlled by *maxneighbor*. Later [EKX95a] and [EKX95b] propose focusing techniques (based on R^*-trees) to improve *CLARANS*'s ability to deal with data objects that may reside on disks by (1) clustering a sample of the dataset that is drawn from each R^*-tree data page; and (2) focusing on relevant data points for distance and quality updates. Their experiments show that the time is improved with a small loss of quality.

2.1 Contributions of *BIRCH*

An important contribution is our formulation of the clustering problem in a way that is appropriate for

very large datasets, by making the time and memory constraints explicit. In addition, *BIRCH* has the following advantages over previous distance-based approaches.

- *BIRCH* is *local* (as opposed to global) in that each clustering decision is made without scanning all data points or all currently existing clusters. It uses measurements that reflect the natural *closeness* of points, and at the same time, can be incrementally maintained during the clustering process.
- *BIRCH* exploits the observation that the data space is usually not uniformly occupied, and hence not every data point is equally important for clustering purposes. A dense region of points is treated collectively as a single *cluster*. Points in sparse regions are treated as *outliers* and removed optionally.
- *BIRCH* makes full use of available memory to derive the finest possible subclusters (to ensure accuracy) while minimizing I/O costs (to ensure efficiency). The clustering and reducing process is organized and characterized by the use of an in-memory, height-balanced and highly-occupied tree structure. Due to these features, its running time is linearly scalable.
- If we omit the optional Phase 4 5, *BIRCH* is an incremental method that does not require the whole dataset in advance, and only scans the dataset once.

3 Background

Assume that readers are familiar with the terminology of vector spaces, we begin by defining centroid, radius and diameter for a cluster. Given N d-dimensional data points in a cluster: $\{\vec{X}_i\}$ where $i = 1, 2, ..., N$, the **centroid** $\vec{X0}$, **radius** R and **diameter** D of the cluster are defined as:

$$\vec{X0} = \frac{\sum_{i=1}^{N} \vec{X}_i}{N} \quad (1)$$

$$R = \left(\frac{\sum_{i=1}^{N} (\vec{X}_i - \vec{X0})^2}{N}\right)^{\frac{1}{2}} \quad (2)$$

$$D = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\vec{X}_i - \vec{X}_j)^2}{N(N-1)}\right)^{\frac{1}{2}} \quad (3)$$

R is the average distance from member points to the centroid. D is the average pairwise distance within a cluster. They are two alternative measures of the tightness of the cluster around the centroid. Next between two clusters, we define 5 alternative distances for measuring their closeness.

Given the centroids of two clusters: $\vec{X0}_1$ and $\vec{X0}_2$, the **centroid Euclidian distance** $D0$ and **centroid Manhattan distance** $D1$ of the two clusters are defined as:

$$D0 = ((\vec{X0}_1 - \vec{X0}_2)^2)^{\frac{1}{2}} \quad (4)$$

$$D1 = |\vec{X0}_1 - \vec{X0}_2| = \sum_{i=1}^{d} |\vec{X0}_1{}^{(i)} - \vec{X0}_2{}^{(i)}| \quad (5)$$

Given N_1 d-dimensional data points in a cluster: $\{\vec{X}_i\}$ where $i = 1, 2, ..., N_1$, and N_2 data points in another cluster: $\{\vec{X}_j\}$ where $j = N_1 + 1, N_1 + 2, ..., N_1 + N_2$, the **average inter-cluster distance** $D2$, **average intra-cluster distance** $D3$ and **variance increase distance** $D4$ of the two clusters are defined as:

$$D2 = \left(\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{N_1 N_2}\right)^{\frac{1}{2}} \quad (6)$$

$$D3 = \left(\frac{\sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{(N_1+N_2)(N_1+N_2-1)}\right)^{\frac{1}{2}} \quad (7)$$

$$D4 = \sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \vec{X}_l}{N_1+N_2}\right)^2 - \sum_{i=1}^{N_1} \left(\vec{X}_i - \frac{\sum_{l=1}^{N_1} \vec{X}_l}{N_1}\right)^2 - \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_2}\right)^2 \quad (8)$$

$D3$ is actually D of the merged cluster. For the sake of clarity, we treat $\vec{X0}$, R and D as properties of a single cluster, and $D0$, $D1$, $D2$, $D3$ and $D4$ as properties between two clusters and state them separately. Users can optionally preprocess data by weighting or shifting along different dimensions without affecting the relative placement.

4 Clustering Feature and CF Tree

The concepts of **Clustering Feature** and **CF** tree are at the core of *BIRCH*'s incremental clustering. A **Clustering Feature** is a triple summarizing the information that we maintain about a cluster.

Definition 4.1 Given N d-dimensional data points in a cluster: $\{\vec{X}_i\}$ where $i = 1, 2, ..., N$, the **Clustering Feature (CF)** vector of the cluster is defined as a triple: $\mathbf{CF} = (N, \vec{LS}, SS)$, where N is the number of data points in the cluster, $\vec{LS}$ is the linear sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{X}_i$, and SS is the square sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{X}_i{}^2$. □

Theorem 4.1 *(**CF Additivity Theorem**): Assume that* $\mathbf{CF_1} = (N_1, \vec{LS}_1, SS_1)$, *and* $\mathbf{CF_2} = (N_2, \vec{LS}_2, SS_2)$ *are the* **CF** *vectors of two disjoint clusters. Then the* **CF** *vector of the cluster that is formed by merging the two disjoint clusters, is:*

$$\mathbf{CF_1} + \mathbf{CF_2} = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, SS_1 + SS_2) \quad (9)$$

The proof consists of straightforward algebra. []

From the **CF** definition and additivity theorem, we know that the **CF** vectors of clusters can be stored and calculated incrementally and accurately as clusters are merged. It is also easy to prove that given the **CF** vectors of clusters, the corresponding $\vec{X0}$, R, D, $D0$, $D1$, $D2$, $D3$ and $D4$, as well as the usual quality metrics (such as weighted total/average diameter of clusters) can all be calculated easily.

One can think of a cluster as a set of data points, but only the **CF** vector stored as summary. This **CF** summary is not only efficient because it stores much less than all the data points in the cluster, but also accurate because it is sufficient for calculating all the measurements that we need for making clustering decisions in *BIRCH*.

4.1 CF Tree

A **CF** tree is a height-balanced tree with two parameters: branching factor B and threshold T. Each nonleaf node contains at most B entries of the form $[\mathbf{CF}_i, child_i]$, where $i = 1, 2, ..., B$, "$child_i$" is a pointer to its i-th child node, and $\mathbf{CF}_i$ is the **CF** of the subcluster represented by this child. So a nonleaf node represents a cluster made up of all the subclusters represented by its entries. A leaf node contains at most L entries, each of the form $[\mathbf{CF}_i]$, where $i = 1, 2, ..., L$. In addition, each leaf node has two pointers, "*prev*" and "*next*" which are used to chain all leaf nodes together for efficient scans. A leaf node also represents a cluster made up of all the subclusters represented by its entries. But all entries in a leaf node must satisfy a *threshold requirement*, with respect to a threshold value T: *the diameter (or radius) has to be less than T.*

The tree size is a function of T. The larger T is, the smaller the tree is. We require a node to fit in a page of size P. Once the dimension d of the data space is given, the sizes of leaf and nonleaf entries are known, then B and L are determined by P. So P can be varied for performance tuning.

Such a **CF** tree will be built dynamically as new data objects are inserted. It is used to guide a new insertion into the correct subcluster for clustering purposes just the same as a B+-tree is used to guide a new insertion into the correct position for sorting purposes. The **CF** tree is a very compact representation of the dataset because each entry in a leaf node is not a single data point but a subcluster (which absorbs many data points with diameter (or radius) under a specific threshold T).

4.2 Insertion into a CF Tree

We now present the algorithm for inserting an entry into a **CF** tree. Given entry "Ent", it proceeds as below:

1. *Identifying the appropriate leaf:* Starting from the root, it recursively descends the **CF** tree by choosing the **closest** child node according to a chosen distance metric: $D0$,$D1$,$D2$,$D3$ or $D4$ as defined in Sec. 3.
2. *Modifying the leaf:* When it reaches a leaf node, it finds the closest leaf entry, say L_i, and then tests whether L_i can "absorb" "Ent" without violating the threshold condition[2]. If so, the **CF** vector for L_i is updated to reflect this. If not, a new entry for "Ent" is added to the leaf. If there is space on the leaf for this new entry, we are done, otherwise we must *split* the leaf node. Node splitting is done by choosing the **farthest** pair of entries as seeds, and redistributing the remaining entries based on the **closest** criteria.
3. *Modifying the path to the leaf:* After inserting "Ent" into a leaf, we must update the **CF** information for each nonleaf entry on the path to the leaf. In the absence of a split, this simply involves adding **CF** vectors to reflect the addition of "Ent". A leaf split requires us to insert a new nonleaf entry into the parent node, to describe the newly created leaf. If the parent has space for this entry, at all higher levels, we only need to update the **CF** vectors to reflect the addition of "Ent". In general, however, we may have to split the parent as well, and so on up to the root. If the root is split, the tree height increases by one.
4. *A Merging Refinement:* Splits are caused by the page size, which is independent of the clustering properties of the data. In the presence of skewed data input order , this can affect the clustering quality, and also reduce space utilization. A simple additional merging step often helps ameliorate these problems: Suppose that there is a leaf split, and the propagation of this split stops at some nonleaf node N_j, i.e., N_j can accommodate the additional entry resulting from the split. We now scan node N_j to find the two **closest** entries. If they are not the pair corresponding to the split, we try to merge them and the corresponding two child nodes. If there are more entries in the two child nodes than one page can hold, we split the merging result again. During the resplitting, in case one of the seed attracts enough merged entries to fill a page, we just put the rest entries with the other seed. In summary, if the merged entries fit on a single page, we free a node space for later use, create one more entry space in node N_j, thereby increasing space utilization and postponing future splits; otherwise we improve the distribution of entries in the closest two children.

Since each node can only hold a limited number of entries due to its size, it does not always correspond to a natural cluster. Occasionally, two subclusters that should have been in one cluster are split across nodes. Depending upon the order of data input and the degree of skew, it is also possible that two subclusters that should not be in one cluster are kept in the same node. These infrequent but undesirable anomalies caused by page size are remedied with a global (or semi-global) algorithm that arranges leaf entries across nodes (Phase 3 discussed in Sec. 5). Another undesirable artifact is that if the same data point is inserted twice, but at different times, the two copies might be entered into distinct leaf entries. Or, in another word, occasionally with a skewed input order, a point might enter a leaf entry that it should not have entered. This problem can be addressed with further refinement passes over the data (Phase 4 discussed in Sec. 5).

5 The BIRCH Clustering Algorithm

Fig. 1 presents the overview of *BIRCH*. The main task of Phase 1 is to scan all data and build an initial in-memory **CF** tree using the given amount of memory

[2] That is, the cluster merged with "Ent" and L_i must satisfy the threshold condition. Note that the **CF** vector of the new cluster can be computed from the **CF** vectors for L_i and "Ent".

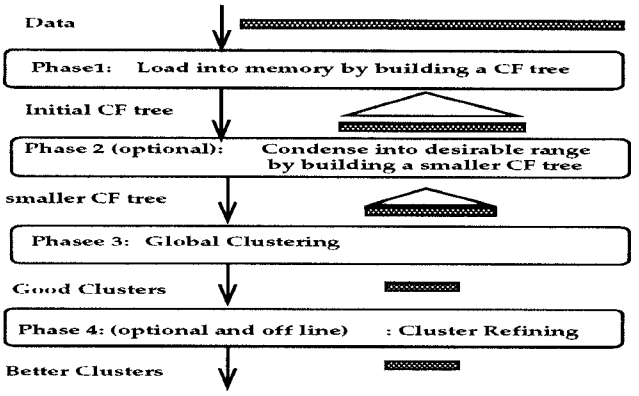


Figure 1: *BIRCH Overview*

and recycling space on disk. This **CF** tree tries to reflect the clustering information of the dataset as fine as possible under the memory limit. With crowded data points grouped as fine subclusters, and sparse data points removed as outliers, this phase creates a in-memory summary of the data. The details of Phase 1 will be discussed in Sec. 5.1. After Phase 1, subsequent computations in later phases will be:

1. fast because (a) no I/O operations are needed, and (b) the problem of clustering the original data is reduced to a smaller problem of clustering the subclusters in the leaf entries;
2. accurate because (a) a lot of outliers are eliminated, and (b) the remaining data is reflected with the finest granularity that can be achieved given the available memory;
3. less order sensitive because the leaf entries of the initial tree form an input order containing better data locality compared with the arbitrary original data input order.

Phase 2 is optional. We have observed that the existing global or semi-global clustering methods applied in Phase 3 have different input size ranges within which they perform well in terms of both speed and quality. So potentially there is a gap between the size of Phase 1 results and the input range of Phase 3. Phase 2 serves as a cushion and bridges this gap: Similar to Phase 1, it scans the leaf entries in the initial **CF** tree to rebuild a smaller **CF** tree, while removing more outliers and grouping crowded subclusters into larger ones.

The undesirable effect of the skewed input order, and splitting triggered by page size (Sec. 4.2) causes us to be unfaithful to the actual clustering patterns in the data. This is remedied in Phase 3 by using a global or semi-global algorithm to cluster all leaf entries. We observe that existing clustering algorithms for a set of data points can be readily adapted to work with a set of subclusters, each described by its **CF** vector. For example, with the **CF** vectors known, (1) naively, by calculating the centroid as the representative of a subcluster, we can treat each subcluster as a single point and use an existing algorithm without modification; (2) or to be a little more sophisticated, we can treat a subcluster of n data points as its centroid repeating n times and modify an existing algorithm slightly to take the counting information into account; (3) or to be general and accurate, we can apply an existing algorithm directly to the subclusters because the information in their **CF** vectors is usually sufficient for calculating most distance and quality metrics.

In this paper, we adapted an agglomerative hierarchical clustering algorithm by applying it directly to the subclusters represented by their **CF** vectors. It uses the accurate distance metric $D2$ or $D4$, which can be calculated from the **CF** vectors, during the whole clustering, and has a complexity of $O(N^2)$. It also provides the flexibility of allowing the user to specify either the desired number of clusters, or the desired diameter (or radius) threshold for clusters.

After Phase 3, we obtain a set of clusters that captures the major distribution pattern in the data. However minor and localized inaccuracies might exist because of the rare misplacement problem mentioned in Sec. 4.2, and the fact that Phase 3 is applied on a coarse summary of the data. Phase 4 is optional and entails the cost of additional passes over the data to correct those inaccuracies and refine the clusters further. Note that up to this point, the original data has only been scanned once, although the tree and outlier information may have been scanned multiple times.

Phase 4 uses the centroids of the clusters produced by Phase 3 as seeds, and redistributes the data points to its closest seed to obtain a set of new clusters. Not only does this allow points belonging to a cluster to migrate, but also it ensures that all copies of a given data point go to the same cluster. Phase 4 can be extended with additional passes if desired by the user, and it has been proved to converge to a minimum [GG92]. As a bonus, during this pass each data point can be labeled with the cluster that it belongs to, if we wish to identify the data points in each cluster. Phase 4 also provides us with the option of discarding outliers. That is, a point which is too far from its closest seed can be treated as an outlier and not included in the result.

5.1 Phase 1 Revisited

Fig. 2 shows the details of Phase 1. It starts with an initial threshold value, scans the data, and inserts points into the tree. If it runs out of memory before it finishes scanning the data, it increases the threshold value, rebuilds a new, *smaller* **CF** tree, by re-inserting the leaf entries of the old tree. After the old leaf entries have been re-inserted, the scanning of the data (and insertion into the new tree) is resumed from the point at which it was interrupted.

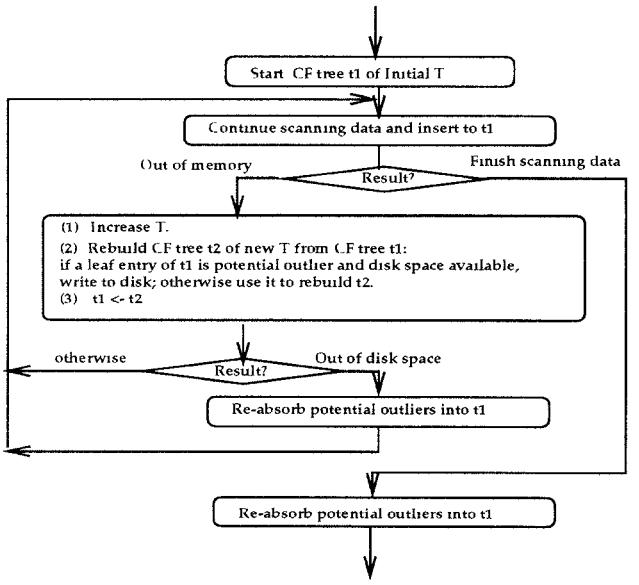


Figure 2: *Control Flow of Phase 1*

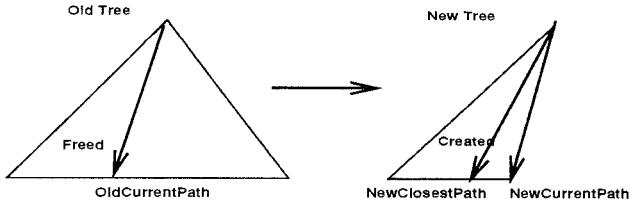


Figure 3: *Rebuilding CF Tree*

5.1.1 Reducibility

Assume t_i is a **CF** tree of threshold T_i. Its height is h, and its size (number of nodes) is S_i. Given $T_{i+1} \geq T_i$, we want to use all the leaf entries of t_i to rebuild a **CF** tree, t_{i+1}, of threshold T_{i+1} such that the size of t_{i+1} should not be larger than S_i. Following is the rebuilding algorithm as well as the consequent reducibility theorem.

Assume within each node of **CF** tree t_i, the entries are labeled contiguously from 0 to $n_k - 1$, where n_k is the number of entries in that node, then a **path** from an entry in the root (level 1) to a leaf node (level h) can be uniquely represented by $(i_1, i_2, ..., i_{h-1})$, where $i_j, j = 1, ..., h-1$ is the label of the j-th level entry on that path. So naturally, path $(i_1^{(1)}, i_2^{(1)}, ..., i_{h-1}^{(1)})$ is **before (or <)** path $(i_1^{(2)}, i_2^{(2)}, ..., i_{h-1}^{(2)})$ if $i_1^{(1)} = i_1^{(2)}$, ..., $i_{j-1}^{(1)} = i_{j-1}^{(2)}$, and $i_j^{(1)} < i_j^{(2)} (0 \leq j \leq h-1)$. It is obvious that a leaf node corresponds to a path uniquely, and we will use path and leaf node interchangeably from now on.

The algorithm is illustrated in Fig. 3. With the natural path order defined above, it scans and frees the old tree path by path, and at the same time, creates the new tree path by path. The new tree starts with NULL, and "OldCurrentPath" starts with the leftmost path in the old tree. For "OldCurrentPath", the algorithm proceeds as below:

1. *Create the corresponding "NewCurrentPath" in the new tree*: nodes are added to the new tree exactly the same as in the old tree, so that there is no chance that the new tree ever becomes larger than the old tree.
2. *Insert leaf entries in "OldCurrentPath" to the new tree*: with the new threshold, each leaf entry in "OldCurrentPath" is tested against the new tree to see if it can fit [3] in the "NewClosestPath" that is found top-down with the **closest** criteria in the new tree. If yes and "NewClosestPath" is **before** "NewCurrentPath", then it is inserted to "NewClosestPath", and the space in "NewCurrentPath" is left available for later use; otherwise it is inserted to "NewCurrentPath" without creating any new node.
3. *Free space in "OldCurrentPath" and "NewCurrentPath"*: Once all leaf entries in "OldCurrentPath" are processed, the un-needed nodes along "OldCurrentPath" can be freed. It is also likely that some nodes along "NewCurrentPath" are empty because leaf entries that originally correspond to this path are now "pushed forward". In this case the empty nodes can be freed too.
4. *"OldCurrentPath" is set to the next path in the old tree if there exists one, and repeat the above steps.*

From the rebuilding steps, old leaf entries are re-inserted, but the new tree can never become larger than the old tree. Since only nodes corresponding to "OldCurrentPath" and "NewCurrentPath" need to exist simultaneously, the maximal extra space needed for the tree transformation is h pages. So by increasing the threshold, we can rebuild a smaller **CF** tree with a limited extra memory.

Theorem 5.1 (Reducibility Theorem:): *Assume we rebuild CF tree t_{i+1} of threshold T_{i+1} from CF tree t_i of threshold T_i by the above algorithm, and let S_i and S_{i+1} be the sizes of t_i and t_{i+1} respectively. If $T_{i+1} \geq T_i$, then $S_{i+1} \leq S_i$, and the transformation from t_i to t_{i+1} needs at most h extra pages of memory, where h is the height of t_i.*

5.1.2 Threshold Values

A good choice of threshold value can greatly reduce the number of rebuilds. Since the initial threshold value T_0 is increased dynamically, we can adjust for its being too low. But if the initial T_0 is too high, we will obtain a less detailed **CF** tree than is feasible with the available memory. So T_0 should be set conservatively. *BIRCH* sets it to zero by default; a knowledgeable user could change this.

[3] Either absorbed by an existing leaf entry, or created as a new leaf entry without splitting.

Suppose that T_i turns out to be too small, and we subsequently run out of memory after N_i data points have been scanned, and C_i leaf entries have been formed (each satisfying the threshold condition wrt. T_i). Based on the portion of the data that we have scanned and the tree that we have built up so far, we need to estimate the next threshold value T_{i+1}. This estimation is a difficult problem, and a full solution is beyond the scope of this paper. Currently, we use the following heuristic approach:

1. We try to choose T_{i+1} so that $N_{i+1} = \text{Min}(2N_i, N)$. That is, whether N is known, we choose to estimate T_{i+1} at most in proportion to the data we have seen thus far.
2. Intuitively, we want to increase threshold based on some measure of *volume*. There are two distinct notions of volume that we use in estimating threshold. The first is *average volume*, which is defined as $V_a = r^d$ where r is the average radius of the root cluster in the **CF** tree, and d is the dimensionality of the space. Intuitively, this is a measure of the space occupied by the portion of the data seen thus far (the "footprint" of seen data). A second notion of volume *packed volume*, which is defined as $V_p = C_i * {T_i}^d$, where C_i is the number of leaf entries and ${T_i}^d$ is the maximal volume of a leaf entry. Intuitively, this is a measure of the actual volume occupied by the leaf clusters. Since C_i is essentially the same whenever we run out of memory (since we work with a fixed amount of memory), we can approximate V_p by ${T_i}^d$.
   We make the assumption that r grows with the number of data points N_i. By maintaining a record of r and the number of points N_i, we can estimate r_{i+1} using least squares linear regression. We define the *expansion factor* $f = Max(1.0, \frac{r_{i+1}}{r_i})$, and use it as a heuristic measure of how the data footprint is growing. The use of *Max* is motivated by our observation that for most large datasets, the observed footprint becomes a constant quite quickly (unless the input order is skewed). Similarly, by making the assumption that V_p grows linearly with N_i, we estimate T_{i+1} using least squares linear regression.
3. We traverse a path from the root to a leaf in the **CF** tree, always going to the child with the most points in a "greedy" attempt to find the most crowded leaf node. We calculate the distance (D_{min}) between the closest two entries on this leaf. If we want to build a more condensed tree, it is reasonable to expect that we should at least increase the threshold value to D_{min}, so that these two entries can be merged.
4. We multiplied the T_{i+1} value obtained through linear regression with the expansion factor f, and adjusted it using D_{min} as follows: $T_{i+1} = Max(D_{min}, f * T_{i+1})$. To ensure that the threshold value grows monotonically, in the very unlikely case that T_{i+1} obtained thus is less than T_i then we choose $T_{i+1} = T_i * (\frac{N_{i+1}}{N_i})^{\frac{1}{d}}$. (This is equivalent to assuming that all data points are uniformly distributed in a d-dimensional sphere, and is really just a crude approximation, however, it is rarely called for.)

5.1.3 Outlier-Handling Option

Optionally, we can use R bytes of disk space for handling *outliers*, which are leaf entries of low density that are judged to be unimportant wrt. the overall clustering pattern. When we rebuild the **CF** tree by re-inserting the old leaf entries, the size of the new tree is reduced in two ways. First, we increase the threshold value, thereby allowing each leaf entry to "absorb" more points. Second, we treat some leaf entries as potential outliers and write them out to disk. An old leaf entry is considered to be a potential outlier if it has "far fewer" data points than the average. "Far fewer", is of course another heuristics.

Periodically, the disk space may run out, and the potential outliers are scanned to see if they can be re-absorbed into the current tree without causing the tree to grow in size. — An increase in the threshold value or a change in the distribution due to the new data read after a potential outlier is written out could well mean that the potential outlier no longer qualifies as an outlier. When all data has been scanned, the potential outliers left in the disk space must be scanned to verify if they are indeed outliers. If a potential outlier can not be absorbed at this last chance, it is very likely a real outlier and can be removed.

Note that the entire cycle — insufficient memory triggering a rebuilding of the tree, insufficient disk space triggering a re-absorbing of outliers, etc. — could be repeated several times before the dataset is fully scanned. This effort must be considered in addition to the cost of scanning the data in order to assess the cost of Phase 1 accurately.

5.1.4 Delay-Split Option

When we run out of main memory, it may well be the case that still more data points can fit in the current **CF** tree, without changing the threshold. However, some of the data points that we read may require us to split a node in the **CF** tree, A simple idea is to write such data points to disk (in a manner similar to how outliers are written), and to proceed reading the data until we run out of disk space as well. The advantage of this approach is that in general, more data points can fit in the tree before we have to rebuild.

6 Performance Studies

We present a complexity analysis, and then discuss the experiments that we have conducted on *BIRCH* (and *CLARANS*) using synthetic as well as real datasets.

6.1 Analysis

First we analyze the cpu cost of Phase 1. The maximal size of the tree is $\frac{M}{P}$. To insert a point, we need to follow a path from root to leaf, touching about $1 + \log_B \frac{M}{P}$ nodes. At each node we must examine B entries, looking for the "closest"; the cost per entry is proportional to the dimension d. So the cost for inserting all data points is $O(d * N * B(1 + \log_B \frac{M}{P}))$. In case we must rebuild the tree, let ES be the **CF** entry size. There are at most $\frac{M}{ES}$ leaf entries to re-insert, so the cost of re-inserting leaf entries is $O(d * \frac{M}{ES} * B(1 + \log_B \frac{M}{P}))$. The number of times we have to re-build the tree depends upon our threshold heuristics. Currently, it is about $\log_2 \frac{N}{N_0}$, where the value 2 arises from the fact that we never estimate farther than twice of the current size, and N_0 is the number of data points loaded into memory with threshold T_0. So the total cpu cost of Phase 1 is $O(d*N*B(1+\log_B \frac{M}{P})+\log_2 \frac{N}{N_0} * d * \frac{M}{ES} * B(1+\log_B \frac{M}{P}))$. The analysis of Phase 2 cpu cost is similar, and hence omitted.

As for I/O, we scan the data once in Phase 1 and not at all in Phase 2. With the outlier-handling and delay-split options on, there is some cost associated with writing out outlier entries to disk and reading them back during a rebuilt. Considering that the amount of disk available for outlier-handling (and delay-split) is not more than M, and that there are about $\log_2 \frac{N}{N_0}$ re-builds, the I/O cost of Phase 1 is not significantly different from the cost of reading in the dataset. Based on the above analysis — which is actually rather pessimistic, in the light of our experimental results — the cost of Phases 1 and 2 should scale linearly with N.

There is no I/O in Phase 3. Since the input to Phase 3 is bounded, the cpu cost of Phase 3 is therefore bounded by a constant that depends upon the maximum input size and the global algorithm chosen for this phase. Phase 4 scans the dataset again and puts each data point into the proper cluster; the time taken is proportional to $N * K$. (However with the newest "nearest neighbor" techniques, it can be improved [GG92] to be almost linear wrt. N.)

6.2 Synthetic Dataset Generator

To study the sensitivity of *BIRCH* to the characteristics of a wide range of input datasets, we have used a collection of synthetic datasets generated by a generator that we have developed. The data generation is controlled by a set of parameters that are summarized in Table 1.

Each dataset consists of K clusters of 2-d data points. A cluster is characterized by the number of data points in it(n), its radius(r), and its center(c). n is in the range of $[n_l,n_h]$, and r is in the range of $[r_l,r_h]$[4]. Once placed, the clusters cover a range of values in each dimension. We refer to these ranges as the "overview" of the dataset.

[4]Note that when $n_l = n_h$ the number of points is fixed and when $r_l = r_h$ the radius is fixed.

Parameter	Values or Ranges
Pattern	grid, sine, random
Number of clusters K	4 .. 256
n_l (Lower n)	0 .. 2500
n_h (Higher n)	50 .. 2500
r_l (Lower r)	0 .. $\sqrt{2}$
r_h (Higher r)	$\sqrt{2}$ .. $\sqrt{32}$
Distance multiplier k_g	4 (grid only)
Number of cycles n_c	4 (sine only)
Noise rate r_n (%)	0 .. 10
Input order o	randomized, ordered

Table 1: *Data Generation Parameters and Their Values or Ranges Experimented*

The location of the center of each cluster is determined by the *pattern* parameter. Three patterns — *grid*, *sine*, and *random* — are currently supported by the generator. When the *grid* pattern is used, the cluster centers are placed on a $\sqrt{K} \times \sqrt{K}$ grid. The distance between the centers of neighboring clusters on the same row/column is controlled by k_g, and is set to $k_g \frac{(r_l+r_h)}{2}$. This leads to an overview of $[0,\sqrt{K}k_g\frac{r_l+r_h}{2}]$ on both dimensions. The *sine* pattern places the cluster centers on a curve of sine function. The K clusters are divided into n_c groups, each of which is placed on a different cycle of the sine function. The x location of the center of cluster i is $2\pi i$ whereas the y location is $\frac{K}{n_c} * sine(2\pi i/(\frac{K}{n_c}))$. The overview of a sine dataset is therefore $[0,2\pi K]$ and $[-\frac{K}{n_c},+\frac{K}{n_c}]$ on the x and y directions respectively. The *random* pattern places the cluster centers randomly. The overview of the dataset is $[0,K]$ on both dimensions since the the x and y locations of the centers are both randomly distributed within the range $[0,K]$.

Once the characteristics of each cluster are determined, the data points for the cluster are generated according to a 2-d independent normal distribution whose mean is the center c, and whose variance in each dimension is $\frac{r^2}{2}$. Note that due to the properties of the normal distribution, the maximum distance between a point in the cluster and the center is unbounded. In other words, a point may be arbitrarily far from its belonging cluster. So a data point that belongs to cluster A may be closer to the center of cluster B than to the center of A, and we refer to such points as "outsiders".

In addition to the clustered data points, noise in the form of data points uniformly distributed throughout the overview of the dataset can be added to the dataset. The parameter r_n controls the percentage of data points in the dataset that are considered noise.

The placement of the data points in the dataset is controlled by the order parameter o. When the randomized option is used, the data points of all clusters and the noise are randomized throughout the entire

Scope	Parameter	Default Value
Global	Memory (M)	80x1024 bytes
	Disk (R)	20% M
	Distance def.	D2
	Quality def.	($\bar{D}$)
	Threshold def.	threshold for $\bar{D}$
Phase1	Initial threshold	0.0
	Delay-split	on
	Page size (P)	1024 bytes
	Outlier-handling	on
	Outlier def.	Leaf entry which contains < 25% of the average number of points per leaf entry
Phase3	Input range	1000
	Algorithm	Adapted HC
Phase4	Refinement pass	1
	Discard-outlier	off
	Outlier def.	Data point whose Euclidian distance to the closest seed is larger than twice of the radius of that cluster

Table 2: *BIRCH Parameters and Their Default Values*

dataset. Whereas when the ordered option is selected, the data points of a cluster are placed together, the clusters are placed in the order they are generated, and the noise is placed at the end.

6.3 Parameters and Default Setting

BIRCH is capable of working under various settings. Table 2 lists the parameters of *BIRCH*, their effecting scopes and their default values. Unless specified explicitly otherwise, an experiments is conducted under this default setting.

M was selected to be 80 kbytes which is about 5% of the dataset size in the base workload used in our experiments. Since disk space (R) is just used for outliers, we assume that $R < M$ and set $R = 20\%$ of M. The experiments on the effects of the 5 distance metrics in the first 3 phases[ZRL95] indicate that (1) using $D3$ in Phases 1 and 2 results in a much higher ending threshold, and hence produces clusters of poorer quality; (2) however, there is no distinctive performance difference among the others. So we decided to choose $D2$ as default. Following Statistics tradition, we choose "weighted average diameter" (denoted as $\bar{D}$) as quality measurement. The smaller $\bar{D}$ is, the better the quality is. The threshold is defined as the threshold for cluster diameter as default.

In Phase 1, the initial threshold is default to 0. Based on a study of how page size affects performance[ZRL95], we selected $P = 1024$. The delay-split option is on so that given a threshold, the **CF** tree accepts more data points and reaches a higher capacity. The outlier-handling option is on so that *BIRCH* can remove outliers and concentrate on the dense places with the given amount of resources. For simplicity, we treat a leaf entry of which the number of data points is less than a quarter of the average as an outlier.

In Phase 3, most global algorithms can handle 1000 objects quite well. So we default the input range as 1000. We have chosen the adapted *HC* algorithm to use here. We decided to let Phase 4 refine the clusters only once with its discard-outlier option off, so that all data points will be counted in the quality measurement for fair comparisons.

6.4 Base Workload Performance

The first set of experiments was to evaluate the ability of BIRCH to cluster various large datasets. All the times are presented in *second* in this paper. Three synthetic datasets, one for each pattern, were used. Table 3 presents the generator settings for them. The weighted average diameters of the actual clusters[5] , $\bar{D}_{act}$ are also included in the table.

Fig. 6 visualizes the actual clusters of DS1 by plotting a cluster as a circle whose center is the centroid, radius is the cluster radius, and label is the number of points in the cluster. The *BIRCH* clusters of DS1 are presented in Fig. 7. We observe that the *BIRCH* clusters are very similar to the actual clusters in terms of location, number of points, and radii. The maximal and average difference between the centroids of an actual cluster and its corresponding *BIRCH* cluster are 0.17 and 0.07 respectively. The number of points in a *BIRCH* cluster is no more than 4% different from the corresponding actual cluster. The radii of the *BIRCH* clusters (ranging from 1.25 to 1.40 with an average of 1.32) are close to, those of the actual clusters (1.41). Note that all the *BIRCH* radii are smaller than the actual radii. This is because *BIRCH* assigns the "outsiders" of an actual clusters to a proper *BIRCH* cluster. Similar conclusions can be reached by analyzing the visual presentations of DS2 and DS3 (but omitted here due to the lack of space).

As summarized in Table 4, it took *BIRCH* less than 50 seconds (on an HP 9000/720 workstation) to cluster 100,000 data points of each dataset. The pattern of the dataset had almost no impact on the clustering time. Table 4 also presents the performance results for three additional datasets – DS1o, DS2o and DS3o – which correspond to DS1, DS2 and DS3, respectively except that the parameter o of the generator is set to *ordered*. As demonstrated in Table 4, changing the order of the data points had almost no impact on the performance of *BIRCH*.

6.5 Sensitivity to Parameters

We studied the sensitivity of *BIRCH*'s performance to the change of the values of some parameters. Due to the lack of space, here we can only present some major conclusions (for details, see [ZRL95]).

[5]From now on, we refer to the clusters generated by the generator as the "actual clusters" whereas the clusters identified by *BIRCH* as "*BIRCH* clusters".

Dataset	Generator Setting	D_{act}
DS1	grid, $K = 100, n_l = n_h = 1000, r_l = r_h = \sqrt{2}, k_g = 4, r_n = 0\%, o = randomized$	2.00
DS2	sine, $K = 100, n_l = n_h = 1000, r_l = r_h = \sqrt{2}, n_c = 4, r_n = 0\%, o = randomized$	2.00
DS3	random, $K = 100, n_l = 0, n_h = 2000, r_l = 0, r_h = 4, r_n = r_n = 0\%, o = randomized$	4.18

Table 3: *Datasets Used as Base Workload*

Initial threshold: (1) *BIRCH*'s performance is stable as long as the initial threshold is not excessively high wrt. the dataset. (2) $T_0 = 0.0$ works well with a little extra running time. (3) If a user does know a good T_0, then she/he can be rewarded by saving up to 10% of the time.

Page Size P: In Phase 1, smaller (larger) P tends to decrease (increase) the running time, requires higher (lower) ending threshold, produces less (more) but "coarser (finer)" leaf entries, and hence degrades (improves) the quality. However with the refinement in Phase 4, the experiments suggest that from $P = 256$ to 4096 , although the qualities at the end of Phase 3 are different, the final qualities after the refinement are almost the same.

Outlier Options: *BIRCH* was tested on "noisy" datasets with all the outlier options *on*, and *off*. The results show that with all the outlier options on, *BIRCH* is not slower but faster, and at the same time, its quality is much better.

Memory Size: In Phase 1, as memory size (or the maximal tree size) increases, the running time increases because of processing a larger tree per rebuilt, but only slightly because it is done in memory; (2) more but finer subclusters are generated to feed the next phase, and hence results in better quality; (3) the inaccuracy caused by insufficient memory can be compensated to some extent by Phase 4 refinements. In another word, *BIRCH* can tradeoff between memory and time to achieve similar final quality.

6.6 Time Scalability

Two distinct ways of increasing the dataset size are used to test the scalability of *BIRCH*.

Increasing the Number of Points per Cluster: For each of DS1, DS2 and DS3, we create a range of datasets by keeping the generator settings the same except for changing n_l and n_h to change n, and hence N. The running time for the first 3 phases, as well as for all 4 phases are plotted against the dataset size N in Fig. 4. Both of them are shown to grow linearly wrt. N consistently for all three patterns.

Increasing the Number of Clusters: For each of DS1, DS2 and DS3, we create a range of datasets by keeping the generator settings the same except for changing K to change N. The running time for the first 3 phases, as well as for all 4 phases are plotted against the dataset size N in Fig. 5. Since both N and K are growing, and Phase 4's complexity is now $O(K*N)$ (can be improved to be almost linear in the future), the total time is not exactly linear wrt. N. However the running time for the first 3 phases is again confirmed to grow linearly wrt. N consistently for all three patterns.

Dataset	Time	$\bar{D}$	Dataset	Time	$\bar{D}$
DS1	47.1	1.87	DS1o	47.4	1.87
DS2	47.5	1.99	DS2o	46.4	1.99
DS3	49.5	3.39	DS3o	48.4	3.26

Table 4: *BIRCH Performance on Base Workload wrt. Time, $\bar{D}$ and Input Order*

Dataset	Time	$\bar{D}$	Dataset	Time	$\bar{D}$
DS1	839.5	2.11	DS1o	1525.7	10.75
DS2	777.5	2.56	DS2o	1405.8	179.23
DS3	1520.2	3.36	DS3o	2390.5	6.93

Table 5: *CLARANS Performance on Base Workload wrt. Time, $\bar{D}$ and Input Order*

6.7 Comparisons of BIRCH and CLARANS

In this experiment we compare the performance of *CLARANS* and *BIRCH* on the base workload. First *CLARANS* assumes that the memory is enough for holding the whole dataset, so it needs much more memory than *BIRCH* does. In order for *CLARANS* to stop after an acceptable running time, we set its *maxneighbor* value to be the larger of 50 (instead of 250) and 1.25% of K(N-K), but no more than 100 (newly enforced upper limit recommended by Ng). Its *numlocal* value is still 2. Fig. 8 visualizes the *CLARANS* clusters for DS1. Comparing them with the actual clusters for DS1 we can observe that: (1) The pattern of the location of the cluster centers is distorted. (2) The number of data points in a *CLARANS* cluster can be as many as 57% different from the number in the actual cluster. (3) The radii of *CLARANS* clusters varies largely from 1.15 to 1.94 with an average of 1.44 (larger than those of the actual clusters, 1.41). Similar behaviors can be observed the visualization of *CLARANS* clusters for DS2 and DS3 (but omitted here due to the lack of space).

Table 5 summarizes the performance of *CLARANS*. For all three datasets of the base workload, (1) *CLARANS* is at least 15 times slower than *BIRCH*, and is sensitive to the pattern of the dataset. (2) The $\bar{D}$ value for the *CLARANS* clusters is much larger than that for the *BIRCH* clusters. (3) The results for DS1o, DS2o, and DS3o show that when the data points are ordered, the time and quality of *CLARANS* degrade dramatically. In conclusion, for the base workload, *BIRCH* uses much less memory, but is faster, more accurate, and less order-sensitive compared with *CLARANS*.

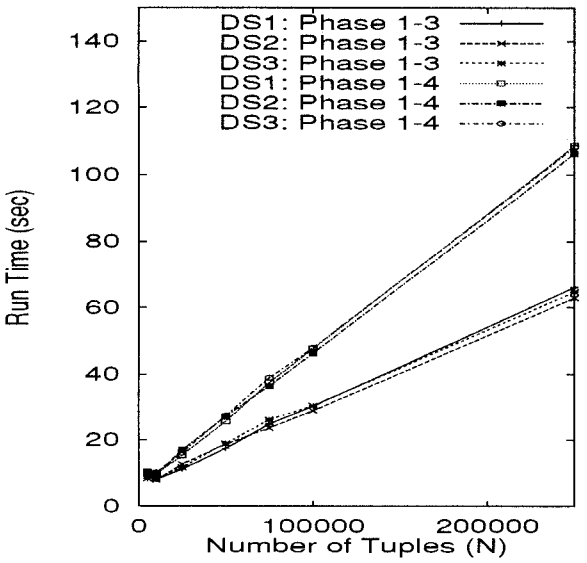


Figure 4: *Scalability wrt. Increasing* n_l, n_h

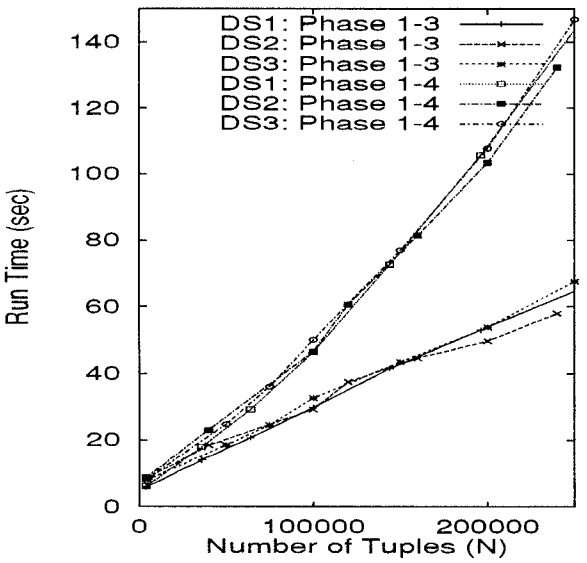


Figure 5: *Scalability wrt. Increasing K*

6.8 Application to Real Datasets

BIRCH has been used for filtering real images. Fig. 9 are two similar images of trees with a partly cloudy sky as the background, taken in two different wavelengths. The top one is in near-infrared band (NIR), and the bottom one is in visible wavelength band (VIS). Each image contains 512x1024 pixels, and each pixel actually has a pair of brightness values corresponding to NIR and VIS. Soil scientists receive hundreds of such image pairs and try to first filter the trees from the background, and then filter the trees into sunlit leaves, shadows and branches for statistical analysis.

We applied *BIRCH* to the (NIR,VIS) value pairs for all pixels in an image (512x1024 2-d tuples) by using 400 kbytes of memory (about 5% of the dataset size) and 80 kbytes of disk space (about 20% of the memory size),

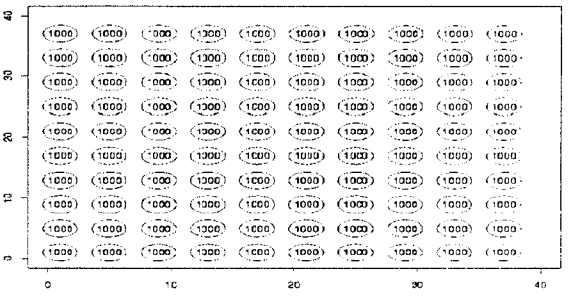


Figure 6: *Actual Clusters of DS1*

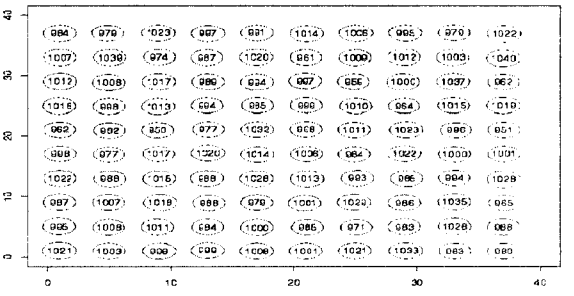


Figure 7: *BIRCH Clusters of DS1*

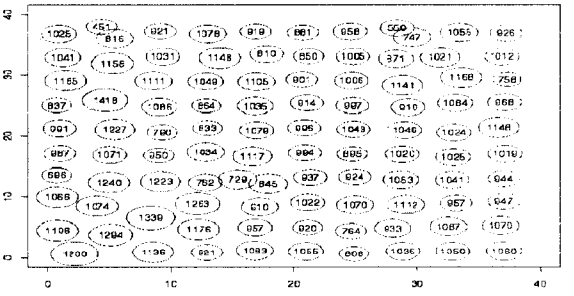


Figure 8: *CLARANS Clusters of DS1*

and weighting NIR and VIS values equally. We obtained 5 clusters that correspond to (1) very bright part of sky, (2) ordinary part of sky, (3) clouds, (4) sunlit leaves (5) tree branches and shadows on the trees. This step took 284 seconds.

However the branches and shadows were too similar to be distinguished from each other, although we could separate them from the other cluster categories. So we pulled out the part of the data corresponding to (5) (146707 2-d tuples) and used *BIRCH* again. But this time, (1) NIR was weighted 10 times heavier than VIS because we observed that branches and shadows were easier to tell apart from the NIR image than from the VIS image; (2) *BIRCH* ended with a finer threshold because it processed a smaller dataset with the same amount of memory. The two clusters corresponding to branches and shadows were obtained with 71 seconds. Fig. 10 shows the parts of image that correspond to

Figure 9: *The images taken in NIR and VIS*

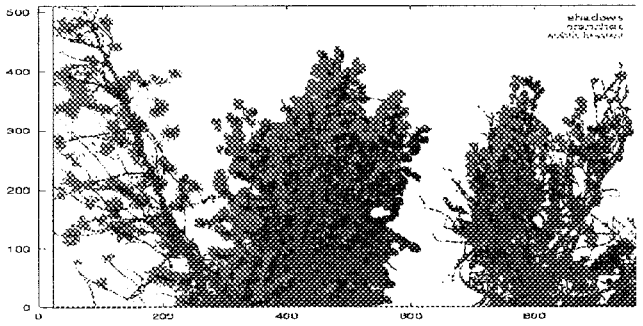


Figure 10: *The sunlit leaves, branches and shadows*

sunlit leaves, tree branches and shadows on the trees, obtained by clustering using *BIRCH*. Visually, we can see that it is a satisfactory filtering of the original image according to the user's intention.

7 Summary and Future Research

BIRCH is a clustering method for very large datasets. It makes a large clustering problem tractable by concentrating on densely occupied portions, and using a compact summary. It utilizes measurements that capture the natural closeness of data. These measurements can be stored and updated incrementally in a height-balanced tree. *BIRCH* can work with any given amount of memory, and the I/O complexity is a little more than one scan of data. Experimentally, *BIRCH* is shown to perform very well on several large datasets, and is significantly superior to *CLARANS* in terms of quality, speed and order-sensitivity.

Proper parameter setting is important to *BIRCH*'s efficiency. In the near future, we will concentrate on studying (1) more reasonable ways of increasing the threshold dynamically, (2) the dynamic adjustment of outlier criteria, (3) more accurate quality measurements, and (4) data parameters that are good indicators of how well *BIRCH* is likely to perform. We will explore *BIRCH*'s architecture for opportunities of parallel executions as well as interactive learnings. As an incremental algorithm, *BIRCH* will be able to read data directly from a tape drive, or from network by matching its clustering speed with the data reading speed. We will also study how to make use of the clustering information obtained to help solve problems such as storage or query optimization, and data compression.

References

[CKS88] Peter Cheeseman, James Kelly, Matthew Self, et al., *AutoClass : A Bayesian Classification System*, Proc. of the 5th Int'l Conf. on Machine Learning, Morgan Kaufman, Jun. 1988.

[DH73] Richard Duda, and Peter E. Hart, *Pattern Classification and Scene Analysis*, Wiley, 1973.

[DJ80] R. Dubes, and A.K. Jain, *Clustering Methodologies in Exploratory Data Analysis* Advances in Computers, Edited by M.C. Yovits, Vol. 19, Academic Press, New York, 1980.

[EKX95a] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu, *A Database Interface for Clustering in Large Spatial Databases*, Proc. of 1st Int'l Conf. on Knowledge Discovery and Data Mining, 1995.

[EKX95b] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu, *Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification*, Proc. of 4th Int'l Symposium on Large Spatial Databases, Portland, Maine, U.S.A., 1995.

[Fis87] Douglas H. Fisher, *Knowledge Acquisition via Incremental Conceptual Clustering*, Machine Learning, 2(2), 1987

[Fis95] Douglas H. Fisher, *Iterative Optimization and Simplification of Hierarchical Clusterings*, Technical Report CS-95-01, Dept. of Computer Science, Vanderbilt University, Nashville, TN 37235.

[GG92] A. Gersho and R. Gray, *Vector quantization and signal compression*, Boston, Ma.: Kluwer Academic Publishers, 1992.

[KR90] Leonard Kaufman, and Peter J. Rousseeuw, *Finding Groups in Data - An Introduction to Cluster Analysis*, Wiley Series in Probability and Mathematical Statistics, 1990.

[Leb87] Michael Lebowitz, *Experiments with Incremental Concept Formation : UNIMEM*, Machine Learning, 1987.

[Lee81] R.C.T.Lee, *Clustering analysis and its applications*, Advances in Information Systems Science, Edited by J.T.Toum, Vol. 8, pp. 169-292, Plenum Press, New York, 1981.

[Mur83] F. Murtagh, *A Survey of Recent Advances in Hierarchical Clustering Algorithms*, The Computer Journal, 1983.

[NH94] Raymond T. Ng and Jiawei Han, *Efficient and Effective Clustering Methods for Spatial Data Mining*, Proc. of VLDB, 1994.

[Ols93] Clark F. Olson, *Parallel Algorithms for Hierarchical Clustering*, Technical Report, Computer Science Division, Univ. of California at Berkeley, Dec.,1993.

[ZRL95] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, Technical Report, Computer Sciences Dept., Univ. of Wisconsin-Madison, 1995.

CLARANS: A method for clustering objects for spatial data mining

Article *in* IEEE Transactions on Knowledge and Data Engineering · October 2002

DOI: 10.1109/TKDE.2002.1033770 · Source: IEEE Xplore

2 authors:

Raymond T. Ng
University of British Columbia - Vancouver
424 PUBLICATIONS **16,914** CITATIONS
SEE PROFILE

Jiawei Han
University of Illinois, Urbana-Champaign
714 PUBLICATIONS **70,928** CITATIONS
SEE PROFILE

CLARANS: A Method for Clustering Objects for Spatial Data Mining

Raymond T. Ng*†
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4,
Canada.

Jiawei Han‡
School of Computing Sciences
Simon Fraser University
Burnaby, B.C., V5A 1S6,
Canada.

Abstract

Spatial data mining is the discovery of interesting relationships and characteristics that may exist implicitly in spatial databases. To this end, this paper has three main contributions. First, we propose a new clustering method called CLARANS, whose aim is to identify spatial structures that may be present in the data. Experimental results indicate that when compared with existing clustering methods, CLARANS is very efficient and effective. Second, we investigate how CLARANS can handle not only points objects, but also polygon objects efficiently. One of the methods considered, called the IR-approximation, is very efficient in clustering convex and non-convex polygon objects. Third, building on top of CLARANS, we develop two spatial data mining algorithms that aim to discover relationships between spatial and non-spatial attributes. Both algorithms can discover knowledge that is difficult to find with existing spatial data mining algorithms.

keywords: spatial data mining, clustering algorithms, randomized search, computational geometry

1 Introduction

Data mining in general is the search for hidden patterns that may exist in large databases. Spatial data mining in particular is the discovery of interesting relationships and characteristics that may exist implicitly in spatial databases. Because of the huge amounts (usually, tera-bytes) of spatial data that may be obtained from satellite images, medical equipments, video cameras, etc., it is costly and often unrealistic for users to examine spatial data in detail. Spatial data mining aims to automate such a knowledge discovery process. Thus, it plays an important role in a) extracting interesting spatial patterns and features; b) capturing intrinsic relationships between spatial and non-spatial data; c) presenting data regularity

*Research partially sponsored by NSERC Grants OGP0138055 and STR0134419, IRIS-3 Grants.
†Person handling correspondence. Email:rng@cs.ubc.ca
‡Research partially supported by NSERC Grant OGP03723 and NCE/IRIS-3 Grants

concisely and at higher conceptual levels; and d) helping to reorganize spatial databases to accommodate data semantics, as well as to achieve better performance.

Many excellent studies on data mining have been conducted, such as those reported in [2, 3, 6, 14, 20, 23, 26]. Agrawal et al. consider the problem of inferring classification functions from samples [2], and study the problem of mining association rules between sets of data items [3]. Han, Cai and Cercone propose an attribute-oriented approach to knowledge discovery [14]. And the book edited by Shapiro and Frawley includes many interesting studies on various issues in knowledge discovery such as finding functional dependencies between attributes [26]. However, most of these studies are concerned with knowledge discovery on non-spatial data, and the work most relevant to our focus here is the one reported in [23]. More specifically, Lu, Han and Ooi propose one spatial dominant and one non-spatial dominant algorithm to extract high-level relationships between spatial and non-spatial data. However, both algorithms suffer from the following problems. First, the user or an expert must provide the algorithms with spatial concept hierarchies, which may not be available in many applications. Second, both algorithms conduct their spatial exploration primarily by merging regions at a certain level of the hierarchy to a larger region at a higher level. Thus, the quality of the results produced by both algorithms relies quite crucially on the appropriateness of the hierarchy to the given data. The problem for most applications is that it is very difficult to know *a priori* which hierarchy will be the most appropriate. Discovering this hierarchy may itself be one of the reasons to apply spatial data mining.

Cluster Analysis is a branch of statistics that in the past three decades has been intensely studied and successfully applied to many applications. To the spatial data mining task at hand, the attractiveness of cluster analysis is its ability to find structures or clusters directly from the given data, without relying on any hierarchies. However, cluster analysis has been applied rather unsuccessfully in the past to general data mining and machine learning. The complaints are that cluster analysis algorithms are ineffective and inefficient. Indeed, for cluster analysis to work effectively, there are the following key issues:

- Whether there exists a natural notion of similarities among the "objects" to be clustered. For spatial data mining, our approach here is to apply cluster analysis only on the spatial attributes. If these attributes correspond to point objects, natural notions of similarities exist (e.g., Euclidean or Manhattan distances). However, if the attributes correspond to polygon objects, the situation is more complicated. More specifically, the similarity (or distance) between two polygon objects may be defined in many ways, some better than others. But more accurate distance measurements may require more effort to compute. The main question then is for the kind of spatial clustering under consideration, which measurement achieves the best balance.

- Whether clustering a large number of objects can be efficiently carried out. Traditional cluster analysis algorithms are not designed for large data sets, with say more than 1000 objects.

In addressing these issues, we report in this paper:

- the development of CLARANS, which aims to use randomized search to facilitate the clustering of a large number of objects; and

- a study on the efficiency and effectiveness of three different approaches to calculate the similarities between polygon objects. They are the approach that calculates the exact separation distance between two polygons, the approach that over-estimates the exact distance by using the minimum distance between vertices, and the approach that under-estimates the exact distance by using the separation distance between the isothetic rectangles of the polygons.

To evaluate our ideas and algorithms, we present results – more often experimental than analytic – showing that:

- CLARANS is more efficient than the existing algorithms PAM and CLARA, both of which motivate the development of CLARANS; and
- calculating the similarity between two polygons by using the separation distance between the isothetic rectangles of the polygons is the most efficient and effective approach.

In [25], we present a preliminary study of CLARANS and the two spatial data mining algorithms. But this paper extends [25] in two major ways. First, CLARANS and the data mining algorithms are generalized to support polygon objects. As motivated above, clustering polygon objects effectively and efficiently is not straightforward at all. Second, this paper presents more detailed analysis and experimental results on the behavior of CLARANS and on the ways to fine tune CLARANS for specific applications.

Since the publication of [25], many clustering methods have been developed, which can be broadly categorized into partitioning methods [7], hierarchical methods [33, 12, 4, 18], density-based methods [11, 15], and grid-based methods [31, 29, 1]. In [7], Bradley et al. proposes an algorithm that follows the basic framework of the K-means algorithm, but that provides scalability by intelligently compressing some regions of the data space. In [33, 12, 4, 18], the proposed hierarchical methods try to detect nested clustering structures, which are prevalent in some applications. In [11, 15], the proposed density-based methods attempt to provide better clustering for elongated clusters; partitioning methods are typically much better suited for spherical clusters. In [31, 29, 1], the developed grid-based methods superimpose a grid structure onto the data space to facilitate clustering.

To compare CLARANS with these works, we make the following general observations:

- Many of the aforementioned techniques require some tree or grid structures to facilitate the clustering. Consequently, these techniques do not scale up well with increasing dimensionality of the datasets. While it is true that the material discussed in this paper is predominantly 2-D, the CLARANS algorithm works the same way for higher dimensional datasets. Because CLARANS is based on randomized search, and does not use any auxiliary structure, CLARANS is much less affected by increasing dimensionality.
- Many of the aforementioned techniques assume that the distance function is Euclidean. CLARANS, being a local search technique, makes no requirement on the nature of the distance function.
- Many of the aforementioned techniques deal with point objects; CLARANS is more general and supports polygonal objects. A considerable portion of this paper is dedicated to handling polygonal objects effectively.

- CLARANS is a main-memory clustering technique, while many of the aforementioned techniques are designed for out-of-core clustering applications. We concede that whenever extensive I/O operations are involved, CLARANS is not as efficient as the others. However, we argue that CLARANS still has considerable applicability. Consider the 2-D objects to be discussed in this paper. Each object is represented by 2 real numbers, occupying a total of 16 bytes. Clustering 1,000,000 objects would require slightly more than 16 Mbytes of main memory. This is an amount easily affordable by a personal computer, let alone computers for data mining. The point here is that given the very low cost of RAM, main-memory clustering algorithms, such as CLARANS, are not completely dominated by out-of-core algorithms for many applications. Finally, on a similar note, although some newly developed clustering methods may find clusters "natural" to the human eye and good for certain applications [4], there are still many applications, such as delivery services, to which partitioning-based clustering, such as CLARANS, is more appropriate.

The paper is organized as follows. Section 2 introduces PAM and CLARA. Section 3 presents our clustering algorithm CLARANS, as well as experimental results comparing the performance of CLARANS, PAM and CLARA. Section 4 studies and evaluates experimentally the three different approaches that compute the similarities between polygon objects. Section 5 concludes the paper with a discussion on ongoing works.

2 Clustering Algorithms based on Partitioning

2.1 Overview

In the past 30 years, cluster analysis has been widely applied to many areas such as medicine (classification of diseases), chemistry (grouping of compounds), social studies (classification of statistical findings), and so on. Its main goal is to identify structures or *clusters* present in the data. Existing clustering algorithms can be classified into two main categories: *hierarchical* methods and *partitioning* methods. Hierarchical methods are either agglomerative or divisive. Given n objects to be clustered, agglomerative methods begin with n clusters (i.e., all objects are apart). In each step, two clusters are chosen and merged. This process continues until all objects are clustered into one group. On the other hand, divisive methods begin by putting all objects in one cluster. In each step, a cluster is chosen and split up into two. This process continues until n clusters are produced. While hierarchical methods have been successfully applied to many biological applications (e.g., for producing taxonomies of animals and plants [19]), they are well known to suffer from the weakness that they can never undo what was done previously. Once an agglomerative method merges two objects, these objects will always be in one cluster. And once a divisive method separates two objects, these objects will never be re-grouped into the same cluster.

In contrast, given the number k of partitions to be found, a partitioning method tries to find the best k partitions [1] of the n objects. It is very often the case that the k clusters found by a partitioning method are of higher quality (i.e., more similar) than the k clusters produced by a hierarchical method. Because of this property, developing partitioning

[1] Partitions here are defined in the usual way: each object is assigned to exactly one group.

methods has been one of the main focuses of cluster analysis research. Indeed, many partitioning methods have been developed, some based on k-means, some on k-medoid, some on fuzzy analysis, etc. Among them, we have chosen the k-medoid methods as the basis of our algorithm for the following reasons. First, unlike many other partitioning methods, the k-medoid methods are very robust to the existence of outliers (i.e., data points that are very far away from the rest of the data points). Second, clusters found by k-medoid methods do not depend on the order in which the objects are examined. Furthermore, they are invariant with respect to translations and orthogonal transformations of data points. Last but not least, experiments have shown that the k-medoid methods described below can handle very large data sets quite efficiently. See [19] for a more detailed comparison of k-medoid methods with other partitioning methods. In the remainder of this section, we present the two best-known k-medoid methods on which our algorithm is based.

2.2 PAM

PAM (Partitioning Around Medoids) was developed by Kaufman and Rousseeuw [19]. To find k clusters, PAM's approach is to determine a representative object for each cluster. This representative object, called a *medoid*, is meant to be the most centrally located object within the cluster. Once the medoids have been selected, each non-selected object is grouped with the medoid to which it is the most similar. More precisely, if O_j is a non-selected object, and O_m is a (selected) medoid, we say that O_j belongs to the cluster represented by O_m, if $d(O_j, O_m) = min_{O_e} d(O_j, O_e)$, where the notation min_{O_e} denotes the minimum over all medoids O_e, and the notation $d(O_1, O_2)$ denotes the dissimilarity or distance between objects O_1 and O_2. All the dissimilarity values are given as inputs to PAM. Finally, the *quality of a clustering* (i.e., the combined quality of the chosen medoids) is measured by the average dissimilarity between an object and the medoid of its cluster. To find the k medoids, PAM begins with an arbitrary selection of k objects. Then in each step, a swap between a selected object O_m and a non-selected object O_p is made, as long as such a swap would result in an improvement of the quality of the clustering.

Before we embark on a formal analysis, let us consider a simple example. Suppose there are 2 medoids: A and B. And we consider replacing A with a new medoid M. Then for all the objects Y that are originally in the cluster represented by A, we need to find the nearest medoid in light of the replacement. There are two cases. In the first case, Y moves to the cluster represented by B, but not to the new one represented by M. In the second case, Y moves to the new cluster represented by M, and the cluster represented by B is not affected. Apart from re-considering all the objects Y that are originally in A's cluster, we also need to consider all the objects Z that are originally in B's cluster. In light of the replacement, Z either stays with B, or moves to the new cluster represented by M. Figure 1 illustrates the four cases.

In the remainder of this paper, we use:

- O_m to denote a current medoid that is to be replaced (e.g., A in Figure 1);
- O_p to denote the new medoid to replace O_m (e.g., M in Figure 1);

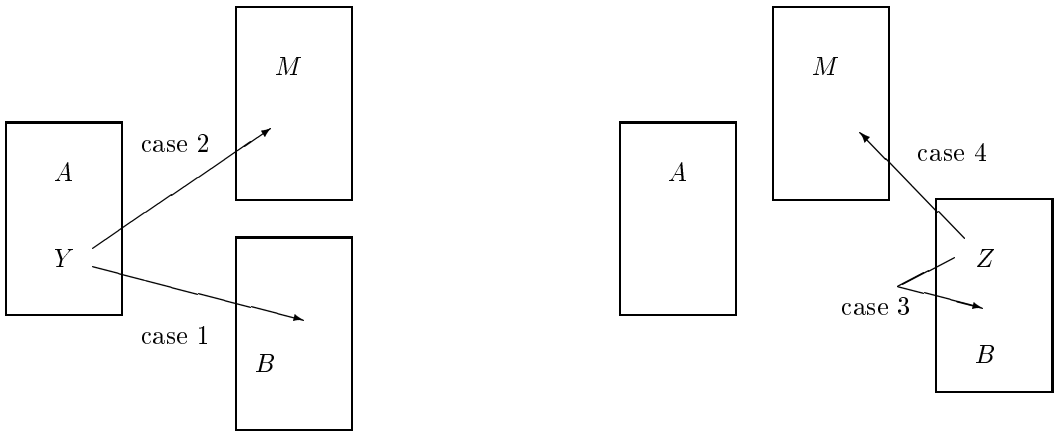


Figure 1: Four cases for Replacing A with M

- O_j to denote other non-medoid objects that may or may not need to be moved (e.g., Y and Z in Figure 1); and
- $O_{j,2}$ to denote a current medoid that is nearest to O_j without A and M (e.g., B in Figure 1).

Now to formalize the effect of a swap between O_m and O_p, PAM computes costs C_{jmp} for all non-medoid objects O_j. Depending on which of the following cases O_j is in, C_{jmp} is defined differently.

Case 1: suppose O_j currently belongs to the cluster represented by O_m. Furthermore, let O_j be more similar to $O_{j,2}$ than to O_p, i.e., $d(O_j, O_p) \geq d(O_j, O_{j,2})$, where $O_{j,2}$ is the second most similar medoid to O_j. Thus, if O_m is replaced by O_p as a medoid, O_j would belong to the cluster represented by $O_{j,2}$ (cf: Case 1 in Figure 1). Hence, the cost of the swap as far as O_j is concerned is:

$$C_{jmp} = d(O_j, O_{j,2}) - d(O_j, O_m). \quad (1)$$

This equation always gives a non-negative C_{jmp}, indicating that there is a non-negative cost incurred in replacing O_m with O_p.

Case 2: O_j currently belongs to the cluster represented by O_m. But this time, O_j is less similar to $O_{j,2}$ than to O_p, i.e., $d(O_j, O_p) < d(O_j, O_{j,2})$. Then, if O_m is replaced by O_p, O_j would belong to the cluster represented by O_p (cf: Figure 1). Thus, the cost for O_j is given by:

$$C_{jmp} = d(O_j, O_p) - d(O_j, O_m). \quad (2)$$

Unlike in Equation (1), C_{jmp} here can be positive or negative, depending on whether O_j is more similar to O_m or to O_p.

Case 3: suppose that O_j currently belongs to a cluster other than the one represented by O_m. Let $O_{j,2}$ be the representative object of that cluster. Furthermore, let O_j be more

similar to $O_{j,2}$ than to O_p. Then even if O_m is replaced by O_p, O_j would stay in the cluster represented by $O_{j,2}$. Thus, the cost is:

$$C_{jmp} = 0. \tag{3}$$

Case 4: O_j currently belongs to the cluster represented by $O_{j,2}$. But O_j is less similar to $O_{j,2}$ than to O_p. Then replacing O_m with O_p would cause O_j to jump to the cluster of O_p from that of $O_{j,2}$ Thus, the cost is:

$$C_{jmp} = d(O_j, O_p) - d(O_j, O_{j,2}), \tag{4}$$

and is always negative. Combining the four cases above, the total cost of replacing O_m with O_p is given by:

$$TC_{mp} = \sum_j C_{jmp} \tag{5}$$

We now present Algorithm PAM.

Algorithm PAM

1. Select k representative objects arbitrarily.
2. Compute TC_{mp} for *all* pairs of objects O_m, O_p where O_m is currently selected, and O_p is not.
3. Select the pair O_m, O_p which corresponds to $min_{O_m,O_p}\ TC_{mp}$. If the minimum TC_{mp} is negative, replace O_m with O_p, and go back to Step (2).
4. Otherwise, for each non-selected object, find the most similar representative object. Halt. □

Experimental results show that PAM works satisfactorily for small data sets (e.g., 100 objects in 5 clusters [19]). But it is not efficient in dealing with medium and large data sets. This is not too surprising if we perform a complexity analysis on PAM. In Steps (2) and (3), there are altogether $k(n-k)$ pairs of O_m, O_p. For each pair, computing TC_{mp} requires the examination of $(n-k)$ non-selected objects. Thus, Steps (2) and (3) combined is of $O(k(n-k)^2)$. And this is the complexity of only one iteration. Thus, it is obvious that PAM becomes too costly for large values of n and k. This analysis motivates the development of CLARA.

2.3 CLARA

Designed by Kaufman and Rousseeuw to handle large data sets, CLARA (Clustering LARge Applications) relies on sampling [19]. Instead of finding representative objects for the entire data set, CLARA draws a sample of the data set, applies PAM on the sample, and finds the medoids of the sample. The point is that if the sample is drawn in a sufficiently random way, the medoids of the sample would approximate the medoids of the entire data set. To come up with better approximations, CLARA draws multiple samples and gives the best

clustering as the output. Here, for accuracy, the quality of a clustering is measured based on the average dissimilarity of all objects in the *entire* data set, and not only of those objects in the samples. Experiments reported in [19] indicate that 5 samples of size $40 + 2k$ give satisfactory results.

Algorithm CLARA

1. For $i = 1$ to 5, repeat the following steps:
2. Draw a sample of $40+2k$ objects randomly from the entire data set [2], and call Algorithm PAM to find k medoids of the sample.
3. For each object O_j in the entire data set, determine which of the k medoids is the most similar to O_j.
4. Calculate the average dissimilarity of the clustering obtained in the previous step. If this value is less than the current minimum, use this value as the current minimum, and retain the k medoids found in Step (2) as the best set of medoids obtained so far.
5. Return to Step (1) to start the next iteration. □

Complementary to PAM, CLARA performs satisfactorily for large data sets (e.g., 1000 objects in 10 clusters). Recall from Section 2.2 that each iteration of PAM is of $O(k(n-k)^2)$. But for CLARA, by applying PAM just to the samples, each iteration is of $O(k(40+k)^2 + k(n-k))$. This explains why CLARA is more efficient than PAM for large values of n.

3 A Clustering Algorithm based on Randomized Search

In this section, we will present our clustering algorithm – CLARANS (Clustering Large Applications based on RANdomized Search). We will first give a graph-theoretic framework within which we can compare PAM and CLARA, and motivate the development of CLARANS. Then after describing the details of the algorithm, we will present experimental results showing how to fine tune CLARANS, and that CLARANS outperforms CLARA and PAM in terms of both efficiency and effectiveness.

3.1 Motivation of CLARANS: a Graph Abstraction

Given n objects, the process described above of finding k medoids can be viewed abstractly as searching through a certain graph. In this graph, denoted by $G_{n,k}$, a node is represented by a set of k objects $\{O_{m_1}, \ldots, O_{m_k}\}$, intuitively indicating that $O_{m_1}, \ldots, O_{m_k}$ are the selected medoids. The set of nodes in the graph is the set $\{ \{O_{m_1}, \ldots, O_{m_k}\} \mid O_{m_1}, \ldots, O_{m_k}$ are objects in the data set$\}$.

Two nodes are neighbors (i.e., connected by an arc) if their sets differ by only one object. More formally, two nodes $S_1 = \{O_{m_1}, \ldots, O_{m_k}\}$ and $S_2 = \{O_{w_1}, \ldots, O_{w_k}\}$ are neighbors if

[2] [19] reports a useful heuristic to draw samples. Apart from the first sample, subsequent samples include the best set of medoids found so far. In other words, apart from the first iteration, subsequent iterations draw $40 + k$ objects to add on to the best k medoids.

and only if the cardinality of the intersection of S_1, S_2 is $k-1$, i.e., $|S_1 \cap S_2| = k-1$. It is easy to see that each node has $k(n-k)$ neighbors. Since a node represents a collection of k medoids, each node corresponds to a clustering. Thus, each node can be assigned a cost that is defined to be the total dissimilarity between every object and the medoid of its cluster. It is not difficult to see that if objects O_m, O_p are the differences between neighbors S_1 and S_2 (i.e., $O_m, O_p \notin S_1 \cap S_2$, but $O_m \in S_1$ and $O_p \in S_2$), the cost differential between the two neighbors is exactly given by T_{mp} defined in Equation (5).

By now, it is obvious that PAM can be viewed as a search for a minimum on the graph $G_{n,k}$. At each step, all the neighbors of the current node are examined. The current node is then replaced by the neighbor with the deepest descent in costs. And the search continues until a minimum is obtained. For large values of n and k (like $n = 1000$ and $k = 10$), examining all $k(n-k)$ neighbors of a node is time consuming. This accounts for the inefficiency of PAM for large data sets.

On the other hand, CLARA tries to examine fewer neighbors and restricts the search on subgraphs that are much smaller in size than the original graph $G_{n,k}$. However, the problem is that the subgraphs examined are defined entirely by the objects in the samples. Let Sa be the set of objects in a sample. The subgraph $G_{Sa,k}$ consists of all the nodes that are subsets (of cardinalities k) of Sa. Even though CLARA thoroughly examines $G_{Sa,k}$ via PAM, the trouble is that the search is fully confined within $G_{Sa,k}$. If M is the minimum node in the original graph $G_{n,k}$, and if M is not included in $G_{Sa,k}$, M will never be found in the search of $G_{Sa,k}$, regardless of how thorough the search is. To atone for this deficiency, many, many samples would need to be collected and processed.

Like CLARA, our algorithm CLARANS does not check every neighbor of a node. But unlike CLARA, it does not restrict its search to a particular subgraph. In fact, it searches the original graph $G_{n,k}$. One key difference between CLARANS and PAM is that the former only checks a sample of the neighbors of a node. But unlike CLARA, each sample is drawn dynamically in the sense that no nodes corresponding to particular objects are eliminated outright. In other words, while CLARA draws a sample of *nodes* at the beginning of a search, CLARANS draws a sample of *neighbors* in each step of a search. This has the benefit of not confining a search to a localized area. As will be shown in Section 3.3, a search by CLARANS gives higher quality clusterings than CLARA, and CLARANS requires a very small number of searches. We now present the details of Algorithm CLARANS.

3.2 CLARANS

Algorithm CLARANS

1. Input parameters *numlocal* and *maxneighbor*. Initialize i to 1, and *mincost* to a large number.

2. Set *current* to an arbitrary node in $G_{n,k}$.

3. Set j to 1.

4. Consider a random neighbor S of *current*, and based on Equation (5), calculate the cost differential of the two nodes.

5. If S has a lower cost, set *current* to S, and go to Step (3).

6. Otherwise, increment j by 1. If $j \leq maxneighbor$, go to Step (4).

7. Otherwise, when $j > maxneighbor$, compare the cost of *current* with *mincost*. If the former is less than *mincost*, set *mincost* to the cost of *current*, and set *bestnode* to *current*.

8. Increment i by 1. If $i > numlocal$, output *bestnode* and halt. Otherwise, go to Step (2). □

Steps (3) to (6) above search for nodes with progressively lower costs. But if the current node has already been compared with the maximum number of the neighbors of the node (specified by *maxneighbor*) and is still of the lowest cost, the current node is declared to be a "local" minimum. Then in Step (7), the cost of this local minimum is compared with the lowest cost obtained so far. The lower of the two costs above is stored in *mincost*. Algorithm CLARANS then repeats to search for other local minima, until *numlocal* of them have been found.

As shown above, CLARANS has two parameters: the maximum number of neighbors examined (*maxneighbor*), and the number of local minima obtained (*numlocal*). The higher the value of *maxneighbor*, the closer is CLARANS to PAM, and the longer is each search of a local minima. But the quality of such a local minima is higher, and fewer local minima needs to be obtained. Like many applications of randomized search [16, 17], we rely on experiments to determine the appropriate values of these parameters.

3.3 Experimental Results: Tuning CLARANS

3.3.1 Details of Experiments

To observe the behavior and efficiency of CLARANS, we ran CLARANS with generated data sets whose clusters are known. For better generality, we used two kinds of clusters with quite opposite characteristics. The first kind of clusters is rectangular, and the objects within each cluster are randomly generated. More specifically, if such a data set of say 3000 objects in 20 clusters is needed, we first generated 20 "bounding boxes" of the same size. To make the clusters less clear-cut, the north-east corner of the i-th box and the south-west corner of $(i+1)$-th box touch. Since for our application of spatial data mining, CLARANS is used to cluster spatial coordinates, objects in our experiments here are pairs of x-, y- coordinates. For each bounding box, we then randomly generated 150 pairs of coordinates that fall within the box. Similarly, we generated data sets of the same kind but of varying numbers of objects and clusters. In the figures below, the symbol rn-k (e.g. r3000-20) represents a data set of this kind with n points in k clusters.

Unlike the first kind, the second kind of clusters we experimented with does not contain random points. Rather, points within a cluster are ordered in a triangle. For example, the points with coordinates (0,0), (1,0), (0,1), (2,0), (1,1), and (0,2) form such a triangular cluster of size 6. To produce a cluster next to the previous one, we used a translation of the origin (e.g., the points (10,10), (11,10), (10,11), (12,10), (11,11), and (10,12)). In the figures

below, the symbol tn-k (e.g., t3000-20) represents a data set organized in this way with n points in k clusters.

All the experiments reported here were carried out in a time-sharing SPARC-LX workstation. Because of the random nature of CLARANS, all the figures concerning CLARANS are average figures obtained by running the same experiment 10 times (with different seeds of the random number generator).

3.3.2 Determining the Maximum Number of Neighbors

In the first series of experiments, we applied CLARANS with the parameter $maxneighbor$ = 250, 500, 750, 1000, and 10000 on the data sets rn-k and tn-k, where n varies from 100 to 3000 and k varies from 5 to 20. To save space, we only summarize the two major findings that lead to further experiments:

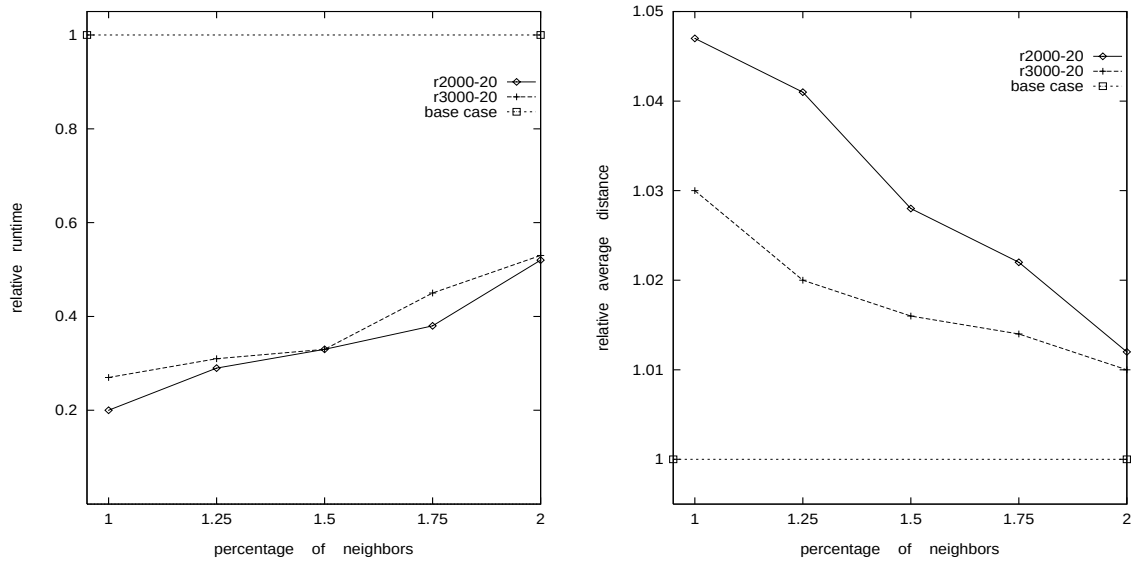
- When the maximum number of neighbors $maxneighbor$ is set to 10000, the quality of the clustering produced by CLARANS is effectively the same as the quality of the clustering produced by PAM (i.e. $maxneighbor = k(n-k)$). While we will explain this phenomenon very shortly, we use the results for $maxneighbor = 10000$ as a yardstick for evaluating other (smaller) values of $maxneighbor$. More specifically, the runtime values of the first graph and the average distance values (i.e. quality of a clustering) of the second graph in Figure 2 below are normalized by those produced by setting $maxneighbor = 10000$. This explains the two horizontal lines at $y-$ value = 1 in both graphs.

- As expected, a lower value of $maxneighbor$ produces a lower quality clustering. A question we ask is then how small can the value of $maxneighbor$ be before the quality of the clustering becomes unacceptable. From the first series of experiments, we find out that these critical values seem to be proportional to the value $k(n-k)$. This motivates us to conduct another series of experiments with the following enhanced formula for determining the value of $maxneighbor$, where $minmaxneighbor$ is a user-defined minimum value for $maxneighhbor$:

  > if $k(n-k) \leq minmaxneighbor$ then $maxneighbor = k(n-k)$; otherwise, $maxneighbor$ equals the the larger value between $p\%$ of $k(n-k)$ and $minmaxneighbor$.

The above formula allows CLARANS to examine all the neighbors as long as the total number of neighbors is below the threshold $minmaxneighbor$. Beyond the threshold, the percentage of neighbors examined gradually drops from 100% to a minimum of $p\%$. The two graphs in Figure 2 show the relative runtime and quality of CLARANS with $minmaxneighbor = 250$ and p varying from 1% to 2%. While the graphs only show the results of the rectangular data sets with 2000 and 3000 points in 20 clusters, these graphs are representative, as the appearances of the graphs for small and medium data sets, and for the triangular data sets are very similar.

Figure 2(a) shows that the lower the value of p, the smaller the amount of runtime CLARANS requires. And as expected, Figure 2(b) shows that a lower value of p produces a lower quality clustering (i.e., higher (relative) average distance). But the very amazing



(a) relative efficiency

(b) relative quality

Figure 2: Determining the Maximum Number of Neighbors

feature shown in Figure 2(b) is that the quality is still within 5% from that produced by setting $maxneighbor = 10000$ (or by PAM). As an example, if a maximum of $p = 1.5\%$ of neighbors are examined, the quality is within 3%, while the runtime is only 40%. What that means is that examining 98.5% more neighbors, while taking much longer, only produces marginally better results. This is consistent with our earlier statement that CLARANS with $maxneigh = 10000$ gives the same quality as PAM, which is effectively the same as setting $maxneighbor = k(n-k) = 20$(3000-20) $= 59600$.

The reason why so few neighbors need to be examined to get good quality clusterings can be best illustrated by the graph abstraction presented in Section 3.1. Recall that each node has $k(n-k)$ neighbors, making the graph very highly connected. Consider two neighbors S_1, S_2 of the current node, and assume that S_1 constitutes a path leading to a certain minimum node S. Even if S_1 is missed by not being examined, and S_2 becomes the current node, there are still numerous paths that connect S_2 to S. Of course, if all such paths are not strictly downward (in cost) paths, and may include "hills" along the way, S will never be reached from S_2. But our experiments seem to indicate that the chance that a hill exists on *every* path is very small.

To keep a good balance between runtime and quality, we believe that a p value between 1.25% and 1.5% is very reasonable. For all our later experiments with CLARANS, we chose the value $p = 1.25\%$.

3.3.3 Determining the Number of Local Minima

Recall that Algorithm CLARANS has two parameters: *maxneighbor* and *numlocal*. Having dealt with the former, here we focus on determining the value of *numlocal*. In this series of experiments, we ran CLARANS with $numlocal = 1, \ldots, 5$ on data sets rn-k and tn-k for

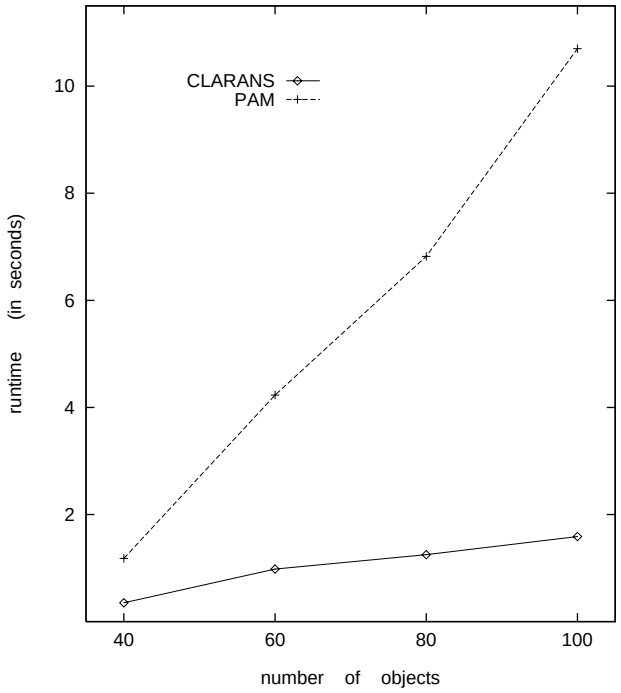


Figure 3: Efficiency: CLARANS vs PAM

small, medium and large values of n and k. For each run, we recorded the runtime and the quality of the clustering. The following table (which is typical of all data sets) shows the relative runtime and quality for the data set r2000-20. Here all the values are normalized by those with $numlocal = 5$.

numlocal	1	2	3	4	5
relative runtime	0.19	0.38	0.6	0.78	1
relative average distance	1.029	1.009	1	1	1

As expected, the runtimes are proportional to the number of local minima obtained. As for the relative quality, there is an improvement from $numlocal = 1$ to $numlocal = 2$. Performing a second search for a local minimum seems to reduce the impact of "unlucky" randomness that may occur in just one search. However, setting *numlocal* larger than 2 is not cost-effective, as there is little increase in quality. This is an indication that a typical local minimum is of very high quality. We believe that this phenomenon is largely due to, as discussed previously, the peculiar nature of the abstract graph representing the operations of CLARANS. For all our later experiments with CLARANS, we used the version that finds two local minima.

3.4 Experimental Results: CLARANS vs PAM

In this series of experiments, we compared CLARANS with PAM. As discussed in Section 3.3.2, for large and medium data sets, it is obvious that CLARANS, while producing clusterings of very comparable quality, is much more efficient than PAM. Thus, our focus here was to compare the two algorithms on small data sets. We applied both algorithms to data sets with 40, 60, 80 and 100 points in 5 clusters. Figure 3 shows the runtime taken by

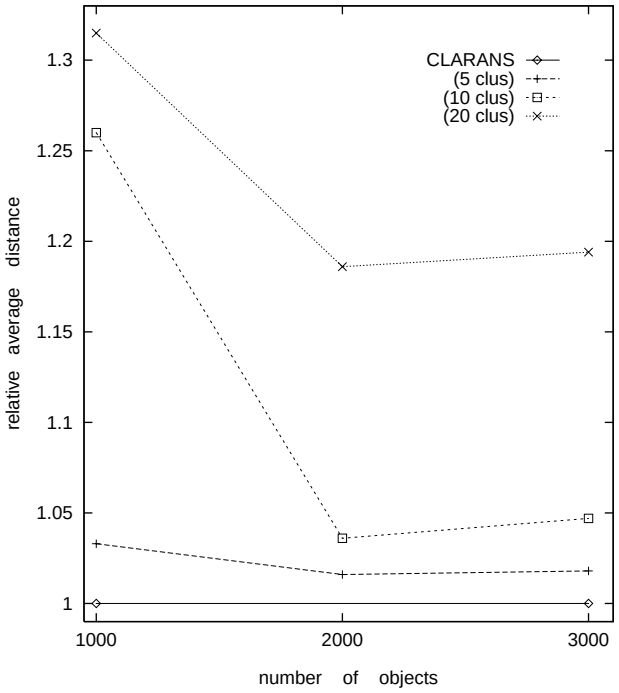


Figure 4: Relative Quality: Same Time for CLARANS and CLARA

both algorithms. Note that for all those data sets, the clusterings produced by both algorithms are of the same quality (i.e., same average distance). Thus, the difference between the two algorithms is determined by their efficiency. It is evident from Figure 3 that even for small data sets, CLARANS outperforms PAM significantly. As expected, the performance gap between the two algorithms grows, as the data set increases in size.

3.5 Experimental Results: CLARANS vs CLARA

In this series of experiments, we compared CLARANS with CLARA. As discussed in Section 2.3, CLARA is not designed for small data sets. Thus, we ran this set of experiments on data sets whose number of objects exceeds 100. And the objects were organized in different number of clusters, as well as in the two types of clusters described in Section 3.3.1.

When we conducted this series of experiments running CLARA and CLARANS as presented earlier, CLARANS is always able to find clusterings of better quality than those found by CLARA. However, in some cases, CLARA may take much less time than CLARANS. Thus, we wondered whether CLARA would produce clusterings of the same quality, if it was given the same amount of time. This leads to the next series of experiments in which we gave both CLARANS and CLARA the same amount of time. Figure 4 shows the quality of the clusterings produced by CLARA, normalized by the corresponding value produced by CLARANS.

Given the same amount of time, CLARANS clearly outperforms CLARA in all cases. The gap between CLARANS and CLARA increases from 4% when k, the number of clusters, is 5 to 20% when k is 20. This widening of the gap as k increases can be best explained by looking at the complexity analyses of CLARA and CLARANS. Recall from Section 2.3 that

each iteration of CLARA is of $O(k^3 + nk)$. On the other hand, recall from Section 3.3.2 that the cost of CLARANS is basically linearly proportional to the number of objects [3]. Thus, an increase in k imposes a much larger cost on CLARA than on CLARANS.

The above complexity comparison also explains why for a fixed number of clusters, the higher the number of objects, the narrower the gap between CLARANS and CLARA is. For example, when the number of objects is 1000, the gap is as high as 30%. The gap drops to around 20% as the number of object increases to 2000. Since each iteration of CLARA is of $O(k^3 + nk)$, the first term k^3 dominates the second term. Thus, for a fixed k, CLARA is relatively less sensitive to an increase in n. On the other hand, since the cost of CLARANS is roughly linearly proportional to n, an increase in n imposes a larger cost on CLARANS than on CLARA. This explains why for a fixed k, the gap narrows as the number of objects increases. Nonetheless, the bottom-line shown in Figure 4 is that CLARANS beats CLARA in all cases.

In sum, we have presented experimental evidence showing that CLARANS is more efficient than PAM and CLARA for small and large data sets. Our experimental results for medium data sets (not included here) lead to the same conclusion.

4 Clustering Convex Polygon Objects

4.1 Motivation

As described in Section 3.3, all the experiments presented so far assume that each object is represented as a point, in which case standard distance metrics such as the Manhattan distance, and the Euclidean distance can be used to calculate the distance between two objects/points. However, in practice, numerous spatial objects that we may want to cluster are polygonal in nature, e.g., shopping malls, parks. The central question then is how to calculate the distance between two polygon objects efficiently and effectively for clustering purposes. One obvious way to approximate polygon objects is to represent each object by a representative point, such as the centroid of the object. However, in general, the objects being clustered may have widely varied sizes and shapes. For instance, a typical house in Vancouver may have a lot size of 200 square meters and a rectangular shape, whereas Stanley Park in Vancouver has a size of about 500,000 square meters and an irregular shape that hugs the shoreline. Simply representing each of these objects by its centroid, or any single point, would easily produce clusterings of poor quality.

Given the above argument, one may wonder whether it is sufficient to represent a polygon object by multiple points in the object, e.g., points on the boundary of the object. But for large objects like Stanley Park, two of its representative points may be 5,000 meters apart

[3] There is a random aspect and a non-random aspect to the execution of CLARANS. The non-random aspect corresponds to the part that finds the cost differential between the current node and its neighbor. This part, as defined in Equation (5) is linearly proportional to the number of objects in the data set. On the other hand, the random aspect corresponds to the part that searches for a local minimum. As the values to plot the graphs are average values of 10 runs, which have the effect of reducing the influence of the random aspect, the runtimes of CLARANS used in our graphs are largely dominated by the non-random aspect of CLARANS.

from each other. If these two representative points are fed to CLARANS as *individual* points/objects, there is no guarantee that they will be in the same cluster. This would result in Stanley Park being assigned to more than one cluster, thus violating the partitioning requirement of the clustering algorithms. [4]

This motivates why in this section, we study how CLARANS (and for that matter, CLARA and PAM) can be augmented to allow convex polygon objects – in their entireties – to be clustered. The key question is how to compute efficiently the distance between two polygons. To answer this question, we study three different approaches. The first one is based on computing the exact separation distance between two convex polygon objects. The second approach uses the minimum distance between vertices to approximate the exact separation distance. The third approach approximates by using the separation distance between isothetic rectangles. We analyze the pros and cons, and complexities of these approaches. Last but not least, we propose a performance optimization that is based on memoizing the computed distances. At the end of this section, we will give experimental results evaluating the usefulness of these ideas.

4.2 Calculating the Exact Separation Distance

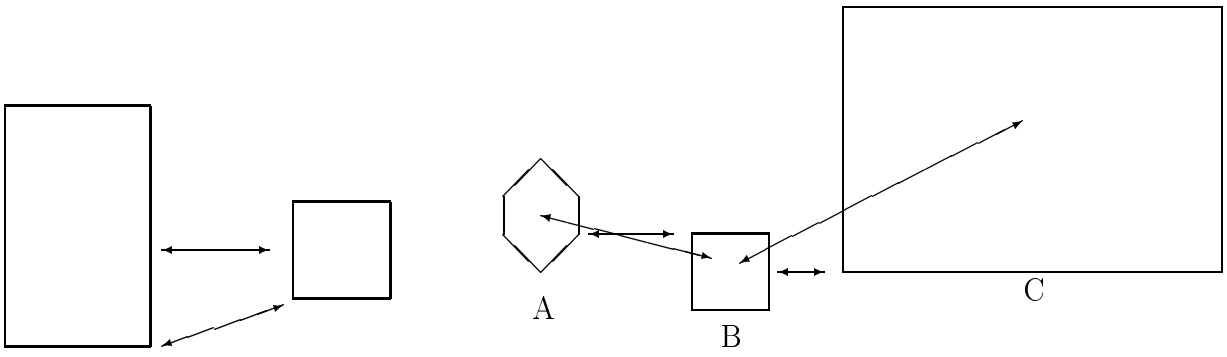
In coordinate geometry, the distance between a point P and a line L is defined as the minimum distance, in this case the perpendicular distance, between the point and the line, i.e., $min\ \{d(P,Q) \mid Q \text{ is a point on } L\}$. Thus, given two polygons A, B, it is natural for us to define the distance between these two polygons to be the minimum distance between any pair of points in A, B, i.e., $min\ \{d(P,Q) \mid P, Q \text{ are points in } A, B \text{ respectively }\}$. This distance is exactly the same as the minimum distance between any pair of points on the boundaries of A, B. This is called the *separation* distance between the two polygons [9, 27].

We need two key steps to compute the separation distance between two convex polygons A, B. First, we need to determine whether A and B have any intersection. This can be done in $O(log\ n\ +\ log\ m)$ time where n, m denotes the number of vertices A and B have [27]. (A vertix of a polygon is the intersection point between two edges of the polygon.) If the two polygons intersect, then the separation distance is zero. Otherwise, we compute the separation distance between the two boundaries. This again can be done in $O(log\, n + log\, m)$ time [9, 21].

While there exist different algorithms that compute the separation distance and that have the same $O(log\ n\ +\ log\ m)$ complexity, we have chosen and implemented one of the most efficient such algorithms that is reported in [21]. Instead of trying all vertices and edges on the boundaries of the two polygons, this algorithm first identifies two chains of vertices and consecutive line segments (one chain from each polygon) that "face" each other in the sense that a vertex in either chain can "see" at least one vertex in the other chain. If P is a vertex in A, and Q a vertex in B, we say that P and Q "see" each other if the line segment joining P and Q does not intersect the interior of either polygon [5]. Thus, by definition, the

[4] Note that there exist clustering algorithms that allow clusters to overlap. However, here we are only concerned with the most standard kind of clusterings, generally known as *crisp* clustering, where each object is assigned to exactly one cluster.

[5] This definition ignores any object that may lie between A and B.



(a) MV-approximation vs separation distance

(b) centroid distance vs separation distance

Figure 5: MV-approximation vs Separation Distance vs Centroid Distance

separation distance of the two polygons must be the minimum distance between any pair of points (not necessarily vertices) on the two chains. By taking a binary search strategy, the algorithm finds the latter distance.

4.3 Approximating by the Minimum Distance between Vertices

One way to approximate the exact separation distance between two polygons is to find the minimum distance between the vertices of the polygons, i.e., $min\ \{d(P,Q) \,|\, P,Q$ are vertices of A,B respectively $\}$. Hereafter, we refer to this approximation as the *MV-approximation*. Obviously, the MV-approximation requires a time complexity of $O(n * m)$. Even though from a complexity point of view, this approximation is inferior to the algorithm described above that computes the exact separation distance, in practice it usually outperforms the exact algorithm, unless when the values of n and m are moderately high, e.g., exceeding 20. For many spatial data mining applications, it is sufficient to represent most spatial objects with fewer than 20 vertices and edges. This justifies using the MV-approximation to optimize performance. Section 4.6.2 will give experimental results on the efficiency of the MV-approximation.

Figure 5(a) shows a simple example demonstrating that the separation distance between two polygons need not be equal to the minimum distance between vertices. However, it is easy to see that the separation distance cannot exceed the minimum distance between vertices. Thus, the MV-approximation always over-estimates the actual separation distance. From the point of view of clustering objects by their MV-approximations, the key question is whether such over-estimations would affect the (quality of the) clusterings.

Recall from Section 4.1 that we argue that using just the centroid to represent an object could produce clusterings of poor quality. And the distance between the centroids of A and B always over-estimates the actual separation distance. At this point, one may wonder whether the over-estimation by centroid distance and the MV-approximation have similar effects. The key difference is that the former, depending largely on the sizes and shapes of the polygons, gives "non-uniform" approximations, whereas the latter is more consistent in

its approximations. In Figure 5(b), B is closer to C than to A based on their exact separation distances. Whereas the centroid distance between A and B is fairly close to the separation distance it approximates, the centroid distance between B and C is many times higher than the actual separation distance between B and C. In fact, by their centroid distances, B is closer to A than to C, reversing the ordering induced by their separation distances. In general, if a collection of objects have widely varied sizes and shapes, the centroid distance approximation would produce poor clusterings (relative to the clusterings produced by using the separation distances). On the other hand, for the example shown in Figure 5(b), the MV-approximation preserves the ordering that B is closer to C than to A. Certainly, an approximation is an approximation, and it is not hard to construct situations where the MV-approximation can be inaccurate. However, by trying the approximation on numerous polygon objects that can be found in real maps, we have verified that the MV-approximation is reasonable, and is much less susceptible to variations in sizes and shapes than the centroid distance approximation is. Section 4.6.4 will give experimental results on the quality of the clusterings produced by the MV-approximation.

4.4 Approximating by the Separation Distance between Isothetic Rectangles

Another way to approximate the exact separation distance between two polygons A, B is to 1) compute isothetic rectangles I_A, I_B, and 2) calculate the separation distance between I_A and I_B. Given a polygon A, the isothetic rectangle I_A is the smallest rectangle that contains A, and whose edges are parallel to either the x- or the y- axes. Hereafter, we refer to this approximation to the exact separation distance as the *IR-approximation*.

For any given polygon A, a minimum bounding rectangle of A is defined to be the smallest rectangle – in area – that contains A. As such, a minimum bounding rectangle need not have its edges parallel to either the x- or the y- axes. Precisely because of this, it is relatively costly to compute a minimum bounding rectangle. In contrast, the isothetic rectangle, while possibly having an area larger than that of a minimum bounding rectangle, can be easily obtained by finding the minimum and maximum of the sets $\{v \mid v$ is the x-coordinate of a vertex of the polygon $\}$, and $\{w \mid w$ is the y-coordinate of a vertex of the polygon $\}$. Thus, Step 1 of the IR-approximation takes a trivial amount of time to compute.

Like computing the exact separation distance between two polygons, as described in Section 4.2, calculating the exact separation distance between two isothetic rectangles requires two steps. In the first step where possible intersection is checked, it only takes constant time for isothetic rectangles, but time logarithmic to the number of vertices for polygons. Similarly, in the next step where the actual separation distance is computed (necessary when the two rectangles or polygons do not intersect), it is constant time for isothetic rectangles, but logarithmic time for polygons. In particular, for isothetic rectangles, it suffices to call repeatedly a procedure that computes the distance between a point and a line segment. Thus, Step 2 of the IR-approximation can be done efficiently. And the IR-approximation can be used to reduce the time otherwise needed to compute the exact separation distance.

From efficiency to effectiveness, just as we evaluate in the previous section how the MV-approximation affects the quality of the clusterings, here we should ask a similar question.

On one hand, the IR-approximation is different from the MV-approximation in that the former always under-estimates the actual separation distance between the original polygons. This is because the isothetic rectangle of a polygon contains the polygon. On the other hand, like the MV-approximation, so long as the IR-approximation under-estimates all polygons being clustered fairly uniformly, the clusterings produced would be of comparable quality to those produced by using the exact separation distances. Section 4.6.4 will give experimental results comparing the quality of the clusterings produced by these two approaches.

So far we have portrayed the IR-approximation (and the MV-approximation) as a performance optimization to computing the exact separation distance. Actually there is another advantage being offered by the IR-approximation (and the MV-approximation). That is, it does not require the original polygon to be convex. As described above, the definition of the isothetic rectangle of a polygon applies equally well to convex and non-convex polygons. Thus, when integrated with the IR-approximation, CLARANS can be used to cluster any polygons. In contrast, the method described in Section 4.2 only works for convex polygons, thereby also restricting CLARANS to convex polygons.

4.5 Memoization of Exact and Approximated Distances

Recall from previous sections that for any pair of objects O_m, O_j, the distance $d(O_m, O_j)$ between the two objects may be referenced numerous times in an execution of CLARANS. When the objects are merely points, this distance can be computed dynamically each time it is needed. This is sufficient because calculating the Manhattan or the Euclidean distance is a simple operation that takes microseconds to complete.

The situation is, however, very different when the objects being clustered are polygons. As will be shown experimentally in Section 4.6.2, calculating the similarity between two polygons, even when the IR-approximation or the MV-approximation is used, still takes milliseconds to complete. Since the distance $d(O_m, O_j)$ may be needed repeatedly, it makes sense to, once computed, memoize the distance. This would ensure that the distance between each pair of polygon objects is computed at most once. Clearly, this memoization strategy trades off space for time efficiency. Section 4.6.5 will give experimental results evaluating whether memoization is valuable, and whether this tradeoff is worthwhile.

4.6 Experimental Evaluation

4.6.1 Details of the Experiments

Recall that both the MV-approximation and the IR-approximation are applicable to calculating the separation distance between non-convex polygons. But because the method described in Section 4.2 only works for convex polygons, all our experiments are restricted to polygon objects that are convex. To generate random polygons of different shapes, sizes, and orientations, we used the convex polygon generator developed by Snoeyink for testing some of the ideas reported in [21]. The generator computes and outputs n evenly-spaced points on an ellipse with center $(ctrx, ctry)$, starting with angle $offset$, and having major and minor axes parallel to the x- and y- axes with radii r_1, r_2 respectively, where $n, ctrx, ctry, offset, r_1, r_2$ are all inputs to the generator. We used a random number generator to create all these

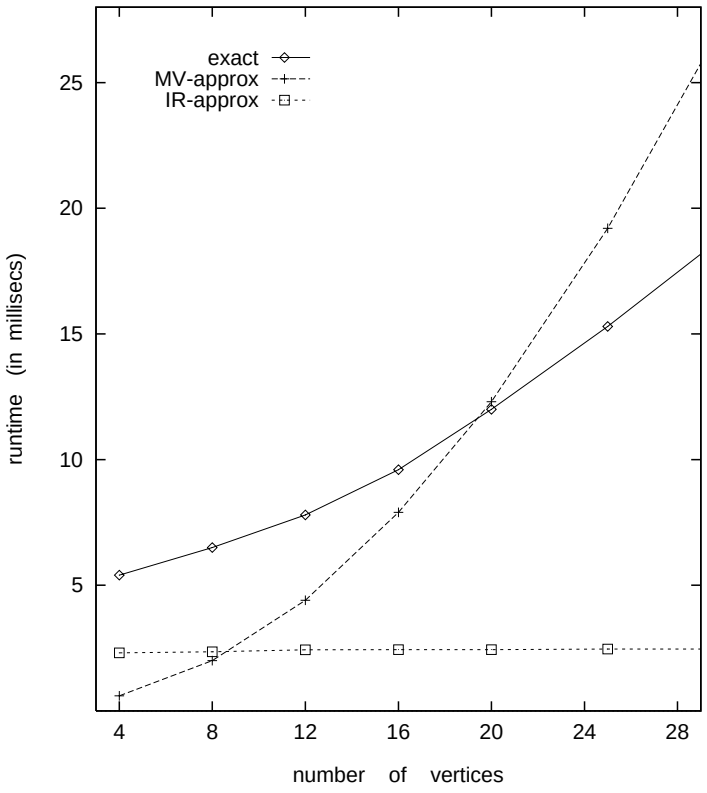


Figure 6: Efficiency of the Approximations

inputs, in such a way that all generated polygons lie in a rectangular region with length l and width w. By varying the values of l and w, we can generate sets of polygons with different densities, and thus with different proportions of polygons that overlap with some other polygons in the set.

All the experiments reported here were carried out in a time-sharing SPARC-LX workstation. Whenever CLARANS was used, all figures concerning CLARANS, due to its random nature, were average figures obtained by running the same experiment 10 times.

4.6.2 Efficiency: Exact Distance vs IR-approximation vs MV-approximation

In this series of experiments, we used polygons with different numbers of vertices, and recorded the average amount of runtime needed to calculate the exact separation distance, its IR-approximation, and its MV-approximation for one pair of polygons. Figure 6 shows the results with the runtimes recorded in milliseconds.

The IR-approximation approach is the clear winner in that it always outperforms the exact separation distance approach, that it beats the MV-approximation approach by a wide margin when the number of vertices is high, and that even when the number of vertices is small, it delivers a performance competitive with that of the MV-approximation approach. Recall from Section 4.4 that two steps are needed to compute the IR-approximation. The first step is to compute the isothetic rectangles, which has a complexity of $O(n)$. The second step is to compute the separation distance between the isothetic rectangles, which has a complexity of $O(1)$. The flatness of the curve for the IR-approximation in Figure 6 clearly

shows that the second step dominates the first one. Thus, as a whole, the IR-approximation does not vary as the number of vertices increases.

In contrast, both the exact separation distance and the MV-approximation require higher runtimes as the number of vertices increases. When the number of vertices is less than 20, calculating the exact separation distance takes more time than the MV-approximation does. But the reverse is true when the number of vertices exceeds 20. In other words, the runtime for the MV-approximation grows faster than that for computing the exact separation distance. This is consistent with the complexity results presented in Sections 4.2 and 4.3.

4.6.3 Clustering Effectiveness: Exact Distance vs IR-approximation vs MV-approximation

While the previous series of experiments do not involve clustering, in this series of experiments, CLARANS was integrated with the three different ways to calculate distances between polygons. Because the number of vertices of polygons is a parameter that affects the performance of the three approaches, we ran CLARANS with sets of n-sided polygons where n varies from 4 to 20, and with a random mix of polygons, each of which has between 4 to 20 edges. For this series of experiments, we focus on both clustering effectiveness and efficiency.

Regarding the effectiveness of the two approximations relative to the exact distance approach, recall that the IR-approximation always underestimates the exact distance, while the MV-approximation always overestimates it. Thus, it is not appropriate to simply measure the effectiveness based on the average approximated distance between a polygon and the medoid of its cluster. Instead, we compare the clusters produced by the exact distance approach and the clusters produced by the two approximations. More precisely, for a particular polygon A, we calculate the ratio:

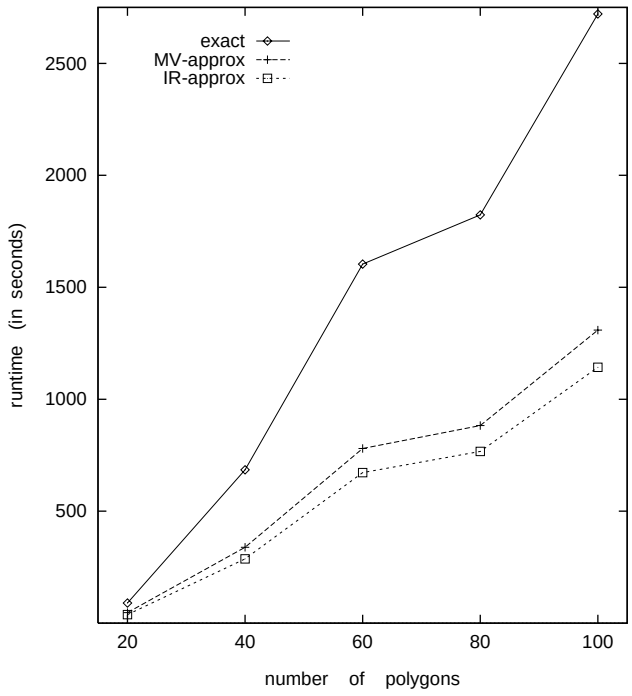
the exact distance between A and M'_A divided by the exact distance between A and M_A,

where M'_A is the medoid of the cluster A is assigned to using the approximation, and M_A is the corresponding medoid using exact distance. The closer to 1 the ratio is, the more accurate the approximation.

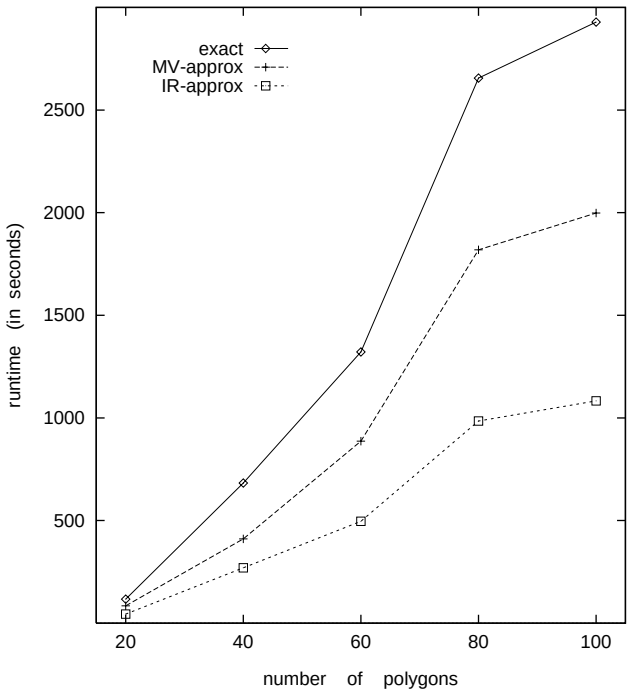
As it turns out for almost every set of polygons we experimented with, over 90% of the polygons satisfy $M'_A = M_A$. That is to say, despite the approximations, the clustering structures are very much preserved. Furthermore, the following table shows the aforementioned ratio average over all polygons A.

Approach	Average Ratio
MV-approximation	1.02
IR-approximation	1.03

The quality of the clusterings produced by the IR-approximation and the MV-approximation is almost identical to the quality of the clustering produced by using the exact separation distance, differing by about 2-3%. This clearly shows that the two approximations are very effective, as they under-estimate or over-estimate the actual distances so consistently that the clusters are preserved. Thus, using the IR-approximation and the MV-approximation as ways to optimize performance is justified.



(a) 10-sided polygons



(b) 4-to-20-sided polygons

Figure 7: Clustering Efficiency of the Approximations

4.6.4 Clustering Efficiency: Exact Distance vs IR-approximation vs MV-approximation

Next we consider the efficiency of the two approximations relative to the exact distance approach. Figure 7 shows the times needed by CLARANS to cluster varying number of polygons that have 10 edges, and varying number of polygons that have between 4 to 20 edges. In both cases, the IR-approximation and the MV-approximation considerably outperform the exact separation distance approach. In particular, the IR-approximation is always the most efficient, requiring only about 30% to 40% the time needed by the exact distance approach.

Other sets of polygons that we experimented with, including some that have varying densities, also give the same conclusion. Thus, given the fact that the IR-approximation is capable of delivering clusterings that are of almost identical quality to those produced by the exact approach, the IR-approximation is the definite choice for CLARANS. Other experiments we conducted indicate that same conclusion can be drawn if PAM and CLARA were to use to cluster polygon objects.

4.6.5 Effectiveness of Memoizing Computed Distances

While the graphs in Figure 7 identify the IR-approximation approach as the clear winner, the performance results of the approximation is disappointing. For example, it takes about 1000 seconds to cluster 100 polygons. Recall from Figure 6 that the IR-approximation for one pair of polygons takes 2 to 3 milliseconds. For 100 polygons, there are 100*100/2 = 5000 pairs of polygons. These 5000 distances take a total of 10 to 15 seconds to compute. Thus, what

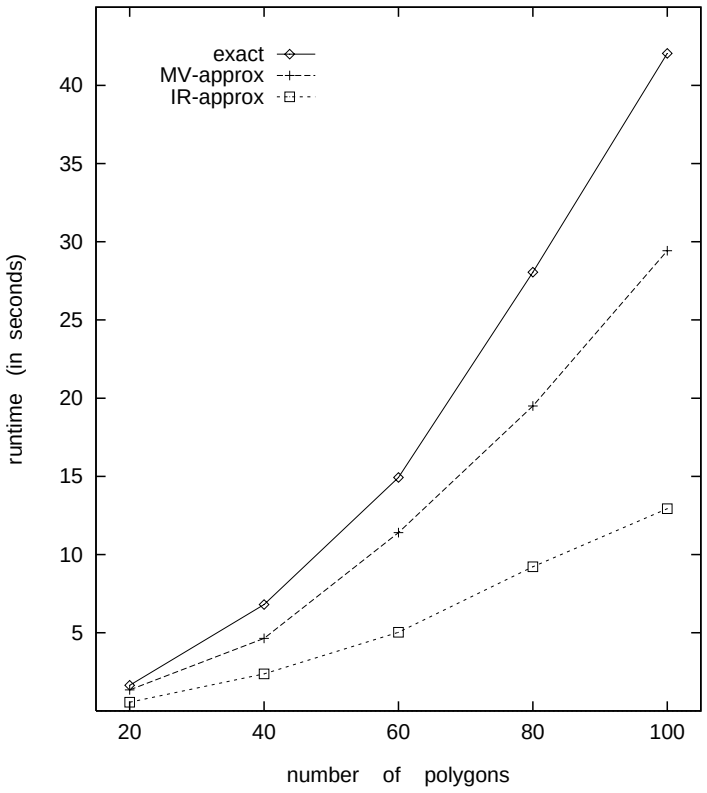


Figure 8: Clustering Efficiency with Distance Memoization

is happening is that the distance between each pair of polygons is computed on the average 60 to 100 times. This argues very strongly why computed distances should be memoized as a performance optimization. Figure 8 shows the results of applying memoization to the same set of polygons used in Figure 7(b). Indeed, with memoization, the time taken to cluster 100 polygons with the IR-approximation drops to about 10 seconds, as estimated above. Similar performance gains are obtained if either the exact separation distance or the MV-approximation are used.

4.7 Summary

Regarding how to compute the similarities between objects, our experimental results clearly indicate that the IR-approximation approach is the choice. While only approximating the exact separation distance, the IR-approximation approach can deliver clusterings of almost identical quality to the clusterings produced by the exact distance approach. The decisive factor is then the efficiency of the IR-approximation approach. It has the desirable property that its computation time does not vary with the number of vertices of the polygon objects, and that it outperforms the exact distance approach typically by 3 to 4 times. Furthermore, it can be used for both convex and non-convex polygon objects.

While the material in this section focuses on collections of objects that are all polygons, the results can be easily generalized to heterogeneous collections of objects, where some objects are polygons and the remaining ones are points. For such collections, the similarity between a point and a polygon can be defined as the distance between the point and the

isothetic rectangle of the polygon. This distance can be computed in constant time.

5 Conclusions

In this paper, we have presented a clustering algorithm called CLARANS which is based on randomized search. For small data sets, CLARANS is a few times faster than PAM; the performance gap for larger data sets is even larger. When compared with CLARA, CLARANS has the advantage that the search space is not localized to a specific subgraph chosen a priori, as in the case of CLARA. Consequently, when given the same amount of run-time, CLARANS can produce clusterings that are of much better quality than those generated by CLARA.

We have also studied how polygon objects can be clustered by CLARANS. We have proposed three different ways to compute the distance between two polygons. Complexity and experimental results indicate that the IR-approximation is a few times faster than the method that computes the exact separation distance. Furthermore, experimental results show that despite the much smaller run-time, the IR-approximation is able to find clusterings that are of quality almost as good as those produced by using the exact separation distances. In other words, the IR-approximation can give significant efficiency gain – but without loss of effectiveness.

In ongoing work, we are developing a package that uses clustering as a basis for providing many operations for mining spatial data. The spatial data mining algorithms SD(CLARANS) and NSD(CLARANS) described in [25] are two examples. There are also operations for mining with maps [32].

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan. (1998) *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications*, Proc. 1998 ACM-SIGMOD, pp. 94–105.

[2] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. (1992) *An Interval Classifier for Database Mining Applications*, Proc. 18th VLDB, pp 560–573.

[3] R. Agrawal, T. Imielinski, and A. Swami. (1993) *Mining Association Rules between Sets of Items in Large Databases*, Proc. 1993 SIGMOD, pp 207–216.

[4] M. Ankerst, M. Breunig, H.-P. Kriegel and J. Sander. (1999) *OPTICS: Ordering Points To Identify the Clustering Structure*, Proc. 1999 ACM-SIGMOD, pp. 49–60.

[5] W. G. Aref and H. Samet. (1991) *Optimization Strategies for Spatial Query Processing*, Proc. 17th VLDB, pp. 81-90.

[6] A. Borgida and R. J. Brachman. (1993) *Loading Data into Description Reasoners*, Proc. 1993 SIGMOD, pp 217–226.

[7] P. Bradley, U. Fayyad and C. Reina. (1998) *Scaling Clustering Algorithms to Large Databases*, Proc. 4th KDD, pp. 9–15.

[8] T. Brinkhoff and H.-P. Kriegel and B. Seeger. (1993) *Efficient Processing of Spatial Joins Using R-trees*, Proc. 1993 SIGMOD, pp 237-246.

[9] D. Dobkin and D. Kirkpatrick. (1985) *A Linear Algorithm for Determining the Separation of Convex Polyhedra*, Journal of Algorithms, 6, 3, pp 381–392.

[10] M. Ester, H. Kriegel and X. Xu. (1995) *Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification*, Proc. 4th Int'l Symp. on Large Spatial Databases (SSD'95), pp 67-82.

[11] M. Ester, H. Kriegel, J. Sander and X. Xu. (1996) *A Density-based Algorithm for Discovering Large Clusters in Large Spatial Databases with Noise*, Proc: 2nd KDD, 1996.

[12] S. Guha, R. Rastogi and K. Shim. (1998) *CURE: An Efficient Clustering Algorithm for Large Databases*, Proc. 1998 ACM-SIGMOD, pp. 73–84.

[13] O. Günther. (1993) *Efficient Computation of Spatial Joins*, Proc. 9th Data Engineering, pp 50-60.

[14] J. Han, Y. Cai and N. Cercone. (1992) *Knowledge Discovery in Databases: an Attribute-Oriented Approach,* Proc. 18th VLDB, pp. 547–559.

[15] A. Hinneburg and D. A. Keim. (1998) *An Efficient Approach to Clustering in Large Multimedia Databases with Noise*, Proc. 1998 KDD, pp. 58–65.

[16] Y. Ioannidis and Y. Kang. (1990) *Randomized Algorithms for Optimizing Large Join Queries*, Proc. 1990 SIGMOD, pp. 312–321.

[17] Y. Ioannidis and E. Wong. (1987) *Query Optimization by Simulated Annealing*, Proc. 1987 SIGMOD, pp. 9–22.

[18] G. Karypis, E.-H. Han and V. Kumar. (1999) *CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling*, IEEE Computer, 32, 8, pp. 68–75.

[19] L. Kaufman and P.J. Rousseeuw. (1990) *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley & Sons.

[20] D. Keim and H. Kriegel and T. Seidl. (1994) *Supporting Data Mining of Large Databases by Visual Feedback Queries*, to appear in Proc. 10th Data Engineering, Houston, TX.

[21] D. Kirkpatrick and J. Snoeyink. (1993) *Tentative Prune-and-search for Computing Fixed-Points with Applications to Geometric Computation*, to appear in Fundamenta Informaticae. Prelim. version in Proc. 9th ACM Symposium on Computational Geometry, pp 133–142.

[22] R. Laurini and D. Thompson. (1992) *Fundamentals of Spatial Information Systems*, Academic Press.

[23] W. Lu, J. Han and B. Ooi. (1993) *Discovery of General Knowledge in Large Spatial Databases,* Proc. Far East Workshop on Geographic Information Systems, Singapore, pp. 275–289.

[24] G. Milligan and M. Cooper. (1985) *An Examination of Procedures for Determining the Number of Clusters in a Data Set,* Psychometrika, 50, pp. 159–179.

[25] R. Ng and J. Han. (1994) *Efficient and Effective Clustering Methods for Spatial Data Mining*, Proc. 20th VLDB, pp. 144-155.

[26] G. Piatetsky-Shapiro and W. J. Frawley. (1991) *Knowledge Discovery in Databases*, AAAI/MIT Press.

[27] F. Preparata and M. Shamos. (1985) *Computational Geometry*, New York, Springer-Verlag.

[28] H. Samet. (1990) *The Design and Analysis of Spatial Data Structures*, Addison-Wesley.

[29] G. Sheikholeslami, S. Chatterjee and A. Zhang. (1998) *WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases*, Proc. 1998 VLDB, pp. 428–439.

[30] H. Spath. (1985) *Cluster Dissection and Analysis: Theory, FORTRAN programs, Examples*, Ellis Horwood Ltd.

[31] W. Wang, J. Yang and R. Muntz. (1997) *STING: a Statistical Information Grid Approach to Spatial Data Mining*, Proc. 23rd VLDB, pp. 186–195.

[32] Y. Yu. (1996) *Finding Strong, Common and Discriminating Characteristics of Clusters from Thematic Maps*, MSc Thesis, Department of Computer Science, University of British Columbia.

[33] T. Zhang, R. Ramakrishnan and M. Livny. (1996) *BIRCH: an Efficient Data Clustering Method for Very Large Databases*, Proc. '96 SIGMOD, pp. 103–114.

Biographies

Raymond Ng received his PhD degree in computer science from the University of Maryland, College Park in 1992. Since then, he has been an assistant and associate professor at the University of British Columbia. His current research interests include data mining, bioinformatics and multimedia database systems. He has published over 80 journal and conference papers, and has served in many program committees of ACM-SIGMOD, VLDB and SIG-KDD.

Jiawei Han received the PhD degree in computer sciences from the University of Wisconsin at Madison in 1985. He is currently the director of Intelligent Database Systems Research Laboratory and professor in the School of Computing Science, Simon Fraser University, Canada.

He has conducted research in the areas of database systems, data mining, data warehousing, geo-spatial data mining, web mining, deductive and object-oriented database systems, and artificial intelligence, with more than 150 journal or conference publications. He is currently a project leader of the Canada Networks of Centres of Excellence (NCE)/IRIS-3 project "Building, querying, analyzing and mining data warehouses on the internet" (1998–2002), and the chief architect of the DBMiner system. He has served or is currently serving in the program committees for more than 50 international conferences and workshops. He has also been serving on the editorial boards of IEEE Transactions on Knowledge and Data Engineering, Data Mining and Knowledge Discovery, and the Journal of Intelligent Information Systems. He is a member of the IEEE Computer Society, the ACM, the ACM-SIGMOD, ACM-SIGKDD, and AAAI.

Computers & Geosciences Vol. 10, No. 2-3, pp. 191-203, 1984.
Printed in the U.S.A.

0098-3004/84 $3.00 + .00

FCM: THE FUZZY *c*-MEANS CLUSTERING ALGORITHM

JAMES C. BEZDEK
Mathematics Department, Utah State University, Logan, UT 84322, U.S.A.

ROBERT EHRLICH
Geology Department, University of South Carolina, Columbia, SC 29208, U.S.A.

WILLIAM FULL
Geology Department, Wichita State University, Wichita, KS 67208, U.S.A.

(*Received* 6 *May* 1982; *revised* 16 *May* 1983)

Abstract—This paper transmits a FORTRAN-IV coding of the fuzzy *c*-means (FCM) clustering program. The FCM program is applicable to a wide variety of geostatistical data analysis problems. This program generates fuzzy partitions and prototypes for any set of numerical data. These partitions are useful for corroborating known substructures or suggesting substructure in unexplored data. The clustering criterion used to aggregate subsets is a generalized least-squares objective function. Features of this program include a choice of three norms (Euclidean, Diagonal, or Mahalonobis), an adjustable weighting factor that essentially controls sensitivity to noise, acceptance of variable numbers of clusters, and outputs that include several measures of cluster validity.

Key Words: Cluster analysis, Cluster validity, Fuzzy clustering, Fuzzy QMODEL, Least-squared errors.

INTRODUCTION

In general, cluster analysis refers to a broad spectrum of methods which try to subdivide a data set X into c subsets (clusters) which are pairwise disjoint, all nonempty, and reproduce X. via union. The clusters then are termed a hard (i.e., nonfuzzy) c-partition of X. Many algorithms, each with its own mathematical clustering criterion for identifying "optimal" clusters, are discussed in the excellent monograph of Duda and Hart (1973). A significant fact about this type of algorithm is the defect in the underlying axiomatic model that each point in X is unequivocally grouped with other members of "its" cluster, and thus bears no apparent similarity to other members of X. One such manner to characterize an individual point's similarity to all the clusters was introduced in 1965 by Zadeh (1965). The key to Zadeh's idea is to represent the similarity a point shares with each cluster with a function (termed the membership function) whose values (called memberships) are between zero and one. Each sample will have a membership in every cluster, memberships close to unity signify a high degree of similarity between the sample and a cluster while memberships close to zero imply little similarity between the sample and that cluster. The history, philosophy, and derivation of such mathematical systems are documented in Bezdek (1981). The net effect of such a function for clustering is to produce fuzzy c-partitions of a given data set. A fuzzy c-partition of X is one which characterizes the membership of each sample point in all the clusters by a membership function which ranges between zero and one. Additionally, the sum of the memberships for each sample point must be unity.

Let $Y = \{y_1, y_2, \ldots, y_N\}$ be a sample of N observations in $\mathbf{R}^n$ (n-dimensional Euclidean space); y_k is the k-th feature vector; y_{kj} the j-th feature of y_k. If c is an integer, $2 \leqslant c < n$, a conventional (or "hard") c-partition of Y is a c-tuple $(Y_1, Y_2, \ldots, Y_c)$ of subsets of Y that satisfies three conditions:

$$Y_i \neq \phi' \quad 1 \leqslant i \leqslant c; \tag{1a}$$

$$Y_i \cap Y_j = \phi; \quad i \neq j \tag{1b}$$

$$\bigcup_{i=1}^{c} Y_i = Y \tag{1c}$$

In these equations, ϕ stands for the empty set, and $(\cap, \cup)$ are respectively, intersection, and union.

In the context discussed later, the sets $\{Y_i\}$ are termed "clusters in Y. Clusters analysis (or simply clustering) in Y refers to the identification of a distinguished c-partition $\{\hat{Y}_i\}$ of Y whose subsets contain points which have high intracluster resemblance; and, simultaneously, low intercluster similarity. The mathematical criterion of resemblance used to define an "optimal" c-partion is termed a cluster criterion. One hopes that the substructure of Y represented by $\{\hat{Y}_i\}$ suggests a useful division or relationship between the population variables of the real physical process from whence Y was drawn. One of the first questions one might ask is whether Y was drawn. One of the first questions one might ask is whether Y contains any clusters at all. In many

geological analyses, a value for c is known *a priori* on physical grounds. If c is unknown, then determination of an optimal c becomes an important issue. This question is sometimes termed the "cluster validity" problem. Our discussion, in addition to the clustering *a posteriori* measures of cluster validity (or "goodness of fit").

Algorithms for clustering and cluster validity have proliferated due to their promise for sorting out complex interactions between variables in high dimensional data. Excellent surveys of many popular methods for conventional clustering using deterministic and statistical clustering criteria are available; for example, consult the books by Duda and Hart (1973), Tou and Gonzalez (1974), or Hartigan (1975). The conventional methodologies discussed in these references include factor analytic techniques, which occupy an important place in the analysis of geoscientific data. The principal algorithms in this last category are embodied in the works of Klovan and Imbrie (1971), Klovan and Miesch (1976), and Miesch (1976a, 1976b). These algorithms for the factor analytical analysis of geoscientific data are known as the QMODEL algorithms (Miesch, 1976a).

In several recent studies, the inadequacy of the QMODEL algorithms for linear unmixing when confronted with certain geometrical configurations in grain shape data has been established numerically (Full, Ehrlich, and Klovan, 1981; Full, Ehrlich, and Bezdek, 1982; Bezdek, and others, 1982. The problem is caused by the presence of outliers. Aberrant points may be real outliers, noise, or simply due to measurement errors; however, peculiarities of this type can cause difficulties for QMODEL that cannot be resolved by standard approaches. The existence of this dilemma led the authors to consider fuzzy clustering methods as an adjunct procedure which might circumvent the problems caused by data of this type. Because fuzzy clustering is most readily understood in terms of the axioms underlying its rationale, we next give a brief description of the basic ideas involved in this model.

FUZZY CLUSTERING

The FCM algorithms are best described by recasting conditions (equation 1) in matrix-theoretic terms. Towards this end, let U be a real $c \times N$ matrix, $U = [u_{ik}]$. U is the matrix representation of the partition $\{Y_i\}$ in equation (1) in the situation

$$u_i(y_k) = u_{ik} = \begin{cases} 1; & y_k \in Y_i \\ 0; & \text{otherwise} \end{cases} \tag{2a}$$

$$\sum_{i=1}^{N} u_{ik} > 0 \quad \text{for all } i; \tag{2b}$$

$$\sum_{i=1}^{N} u_{ik} = 1 \quad \text{for all } k. \tag{2c}$$

In equation (2), u_i is a function; u_i: $Y \rightarrow \{0, 1\}$. In conventional models, u_i is the characteristic function of Y_i: in fact, u_i and Y_i determine one another, so there is no harm in labelling u_i the ith hard subset of the partition (it is unusual, of course, but is important in terms of understanding the term "fuzzy set"). Conditions of equations (1) and (2) are equivalent, so U is termed a hard c-partition of Y. Generalizing this idea, we refer to U as a fuzzy c-partition of Y when the elements of U are numbers in the unit interval [0, 1] that continue to satisfy both equations (2b) and (2c). The basis for this definition are c functions u_i: $Y \rightarrow [0, 1]$ whose values $u_i(y_k) \in [0, 1]$ are interpreted as the grades of membership of the y_ks in the "fuzzy subsets" u_i of Y. This notion is due to Zadeh (1965), who conceived the idea of the fuzzy set as a means for modelling physical systems that exhibit nonstatistical uncertainties. Detailed discussions for the rationale and philosophy of fuzzy sets are available in many recent papers and books (e.g., consult Bezdek (1981)).

For the present discussion, it suffices to note that hard partitions of Y are a special type of fuzzy ones, wherein each data point is grouped unequivocally with its intracluster neighbors. This requirement is a particularly harsh one for physical systems that contain mixtures, or hybrids, along with pure or antecedent strains. Outliers (noise or otherwise) generally fall into the category one should like to reserve for "unclassifiable" points. Most conventional models have no natural mechanism for absorbing the effects of undistinctive or aberrant data, this is a direct consequence of equation (1a). Accordingly, the fuzzy set, and, in turn, fuzzy partition, were introduced as a means for altering the basic axioms underlying clustering and classification models with the aim of accomodating this need. By this device, a point y_k may belong entirely to a single cluster, but in general, is able to enjoy partial membership in several fuzzy clusters (e.g., precisely the situation anticipated for hybrids). We denote the sets of all hard and fuzzy c-partitions of Y by:

$$M_c = \{U_{c\times N} | u_{ik} \in [0, 1]; \quad \text{equations (2b), (2c)}\}; \tag{3a}$$

$$M_{fc} = \{U_{c\times N} | u_{ik} \in [0, 1]; \quad \text{equations (2b), (2c)}]. \tag{3b}$$

Note that M_c is imbedded in M_{fc}. This means that fuzzy clustering algorithms can obtain hard c-partitions. On the other hand, hard clustering algorithms cannot determine fuzzy c-partitions of Y. In other words, the fuzzy imbedment enriches (not replaces!) the conventional partitioning model. Given that fuzzy c-partitions have at least intuitive appeal, how does one use the data to determine them? This is the next question we address.

Several clustering criteria have been proposed for identifying optimal fuzzy c-partitions in Y. Of these, the most popular and well studied method to date is

associated with the generalized least-squared errors functional

$$J_m(U, v) = \sum_{k=1}^{N} \sum_{i=1}^{c} (u_{ik})^m \|y_k - v_i\|_A^2 \quad (4)$$

Equation (4) contains a number of variables: these are

$$Y = \{y_1, y_2, \ldots, y_N\} \subset \mathbf{R}^n = \text{the data}, \quad (5a)$$

$$c = \text{number of clusters in } Y; \quad 2 \leqslant c < n, \quad (5b)$$

$$m = \text{weighting exponent}; \quad 1 \leqslant m < \infty, \quad (5c)$$

$$U = \text{fuzzy } c\text{-partition of } Y; \quad U \in M_{fc} \quad (5d)$$

$$v = (v_1, v_2, \ldots, v_c) = \text{vectors of centers}, \quad (5e)$$

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in}) = \text{center of cluster } i, \quad (5f)$$

$$\| \ \|_A = \text{induced } A\text{-norm on } \mathbf{R}^n \quad (5g)$$

$$A = \text{positive-definite } (n \times n) \text{ weight matrix.} \quad (5h)$$

The squared distance between y_k and v_i shown in equation (4) is computed in the A-norm as

$$d_{ik}^2 = \|y_k - v_i\|_A^2 = (y_k - v_i)^T A (y_k - v_i). \quad (6)$$

The weight attached to each squared error is $(u_{ik})^m$, *the* mth power of y_ks membership in cluster i. The vectors $\{v_i\}$ in equation (5f) are viewed as "cluster centers" or centers of mass of the partitioning subsets. If $m = 1$, it can be shown that J_m minimizes only at hard U's $\in M_c$, and corresponding v_is are just the geometric centroids of the Y_is. With these observations, we can decompose J_m into its basic elements to see what property of the points $\{y_k\}$ it measures:

$$d_{ik}^2 = \text{squared } A\text{-distance from point } y_k \text{ to center of mass } v_i. \quad (7a)$$

$$(u_{ik})^m d_{ik}^2 = \text{squared } A\text{-error incurred by representing } y_k \text{ by } v_i \text{ weighted by (a power of) the membership of } y_k \text{ in cluster } i. \quad (7b)$$

$$\sum_{i=1}^{c} (u_{ik})^m d_{ik}^2 = \text{sum of squared } A\text{-errors due to } y_k\text{s partial replacement by all } c \text{ of the centers } \{v_i\}. \quad (7c)$$

$$\sum_{k=1}^{N} \sum_{i=1}^{c} (u_{ik})^m d_{ik}^2 = \text{overall weighted sum of generalized } A\text{-errors due to replacing } Y \text{ by } v. \quad (7d)$$

The role played by most of the variables exhibited in equation (5) is clear. Two of the parameters of J_m, however warrant further discussion, namely, m and A. Weighting exponent m controls the relative weights placed on each of the squared errors d_{ik}^2. As $m \to 1$ from earlier discussion partitions that minimize J_m become increasingly hard (and, as mentioned before, at $m = 1$, are necessarily hard). Conversely, each entry of optimal $\hat{U}$s for J_m approaches $(1/c)$ as $m \to \infty$. Consequently, increasing m tends to degrade (blur, defocus) membership towards the fuzziest state. Each choice for m defines, all other parameters being fixed, one FCM algorithm. No theoretical or computational evidence distinguishes an optimal m. The range of useful values seems to be [1, 30] or so. If a test set is available for the process under investigation, the best strategy for selecting m at present seems to be experimental. For most data, $1.5 \leqslant m \leqslant 3.0$ gives good results.

The other parameter of J_m that deserves special mention is weight matrix A. This matrix controls the shape that optimal clusters assume in $\mathbf{R}^n$. Because every norm on $\mathbf{R}^n$ is inner product induced via the formula

$$\langle x, y \rangle_A = x^T A y, \quad (8)$$

there are infinitely many A-norms available for use in equation (4). In practice, however, only a few of these norms enjoy widespread use. The FCM listing below allows a choice of three norms, each induced by a specific weight matrix. Let

$$c_y = \sum_{k=1}^{N} y_k / N; \quad (9a)$$

$$C_y = \sum_{k=1}^{N} (y_k - c_y)(y_k - c_y)^t, \quad (9b)$$

be the sample mean and sample covariance matrix of data set Y; and let $\{a_i\}$ denote the eigenvalues of C_y; let D_y be the diagonal matrix with diagonal elements $(d_y)_{ii} = a_i$; and finally, let I be the identity matrix. The norms of greatest interest for use with equation (4) correspond to

$$A = I \sim \text{-Euclidean Norm}, \quad (10a)$$

$$A = D_y^{-1} \sim \text{Diagonal Norm}, \quad (10b)$$

$$A = C_y^{-1} \sim \text{Mahalonobis Norm}. \quad (10c)$$

A detailed discussion of the geometric and statistical implications of these choices can be seen in Bezdek (1981). When $A = I$, J_m identifies hyperspherical clusters; for any other A, the clusters are essentially hyperellipsodial, with axes proportional to the eigenvalues of A. When the diagonal norm is used, each dimension is effectively scaled via the eigenvalues. The Euclidean norm is the only choice for which extensive experience with geological data is available.

Optimal fuzzy clusterings of Y are defined as pairs $(\hat{U}, \hat{v})$ that locally minimize J_m. The necessary conditions for $m = 1$ are well known (but hard to use, because M_c is discrete, but large). For $m > 1$, if $y_k \neq \hat{v}_j$ for all j and k, $(\hat{U}, \hat{v})$ may be locally optimal for J_m only if

$$\hat{v}_i = \sum_{k=1}^{N} (\hat{u}_{ik})^m y_k \Big/ \sum_{k=1}^{N} (\hat{u}_{ik})^m; \quad 1 \leqslant i \leqslant c; \quad (11a)$$

$$\hat{u}_{ik} = \left(\sum_{j=1}^{c} \left(\frac{\hat{d}_{ik}}{\hat{d}_{jk}} \right)^{2/(m-1)} \right)^{-1}; \quad 1 \leqslant k \leqslant N; \ 1 \leqslant i \leqslant c \quad (11b)$$

where $\hat{d}_{ik} = \|y_k - \hat{v}_i\|_A$. Conditions expressed in equations (11) are necessary, but not sufficient; they provide means for optimizing J_m via simple Picard iteration, by looping back and forth from equation (11a) to (11b) until the iterate sequence shows but small changes in successive entries of $\hat{U}$ or $\hat{v}$. We formalize the general procedure as follows:

Fuzzy c-Means (FCM) Algorithms

(A1) Fix c, m, A, $\|k\|_A$. Choose an initial matrix $U^{(o)} \in M_{fc}$. Then at step k, $k = 0, 1, \ldots, LMAX$.

(A) Compute means $\hat{v}^{(k)}$, $i = 1, 2, \ldots, c$ with equation (11)a.

(A3) Compute an updated membership matrix $\hat{U}^{(k+1)} = [\hat{u}_{ik}^{(k+1)}]$ with equation (11b).

(A4) Compare $\hat{U}^{k+1}$ to $\hat{U}^{(k)}$ in any convenient matrix norm. If $\|\hat{U}^{(k+1)} - \hat{U}^{(k)}\| < \in$, stop. Otherwise, set $\hat{U}^{(k)} = \hat{U}^{k+1}$ and return to (A2).

(A1)–(A4) is the basic algorithmic strategy for the FCM algorithms.

Individual control parameters, tie-breaking rules, and computing protocols are discussed in conjunction with the appended FORTRAN listing in Appendix 1.

Theoretical convergence of the sequence $\{\hat{U}^{(k)}, \hat{v}^{(k)}, k = 0, 1, \cdots\}$ generated by (A1)–(A4) has been studied (by Bezdek, 1981). Practically speaking, no difficulties have ever been encountered, and numerical convergence is usually achieved in 10–25 iterations. Whether local minima of J_m are good clusterings of Y is another matter, for it is easy to obtain data sets upon which J_m minimizes globally with visually unappealing substructure. To mitigate this difficulty, several types of cluster validity functionals are usually calculated on each $\hat{U}$ produced by FCM. Among the most popular are the partition coefficient and entropy of $\hat{U} \in M_{fc}$:

$$F_c(\hat{U}) = \sum_{k=1}^{N} \sum_{i=1}^{c} (\hat{u}_{ik})^2/N; \tag{12a}$$

$$H_c(\hat{U}) = -\sum_{k=1}^{N} \sum_{i=1}^{c} (\hat{u}_{ik} \log_a(\hat{u}_{ik}))/N. \tag{12b}$$

In equation (12b), logarithmic base $a \in (1, \infty)$. Properties of F_c and H_c utilized for validity checks are:

$$F_c = 1 \Leftrightarrow H_c = 0 \Leftrightarrow \hat{U} \in M_c \text{ is hard;} \tag{13a}$$

$$F_c = 1/c \Leftrightarrow H_c = \log_a(c) \Leftrightarrow \hat{U} = [1/c]; \tag{13b}$$

$$\frac{1}{c} \leqslant F_c \leqslant 1; \quad 0 \leqslant H_c \leqslant \log_a(c). \tag{13c}$$

Entropy H is a bit more sensitive than F to local changes in partition quality. The FCM program listed below calculates F, H, and $(1 - F)$, the latter quantity owing to the inequality $(1 - F) < H$ for $\hat{U} \not\in M_c$ (when $a = e = 2.71\cdots$).

Finally, we observe that generalizations of J_m which can accommodate a much wider variety of data shapes than FCM are now well known (see Bezdek (1981) for a detailed account). Nonetheless, the basic FCM algorithm remains one of the most useful general purpose fuzzy clustering routines, and is the one utilized in the FUZZY QMODEL algorithms discussed by Full, Ehrlich, and Bezdek (1982). Having given a brief account of the generalities, we now turn to computing protocols for the FCM listing accompanying this paper.

ALGORITHMIC PROTOCOLS

The listing of FCM appended below has some features not detailed in (A1)–(A4). Our description of the listing corresponds to the blocks as documented.

Input Variables. FCM arrays are listing documented. Symbolic dimensions are

NS = number of vectors in $Y = N$.

ND = number of features in $y_k = n$.

Present dimensions will accomodate up to $c = 20$ clusters, $N = 500$ data points, and $n = 20$ features. Input variables ICON specifies the weight matrix A as in equation (10):

ICON = $1 \Rightarrow A = I$

ICON = $2 \Rightarrow A = D_y^{-1}$.

ICON = $3 \Rightarrow A = C_y^{-1}$.

Other parameters read are:

QQ = Weighting exponent m: $1 <$ QQ.

KBEGIN = Initial number of clusters:

$2 \leqslant$ KBEGIN $\leqslant$ DCEASE.

KCEASE = Final number of clusters:

KCEASE $<$ NS.

At any step NCLUS $= C$ is the operating number of clusters. FCM iterates over NCLUS from KBEGIN to KCEASE, generating an optimal pair $(\hat{U}, \hat{v})_{\text{NCLUS}}$ for each number of clusters desired. Changes in m and A must be made between runs (although they could easily be made iterate parameters).

Control Parameters

EPS = Termination criterion $\in$ in (A4).

LMAX = Maximum number iterations at each c in (A1).

Current values of EPS and LMAX are 0.01 and 50. Lowering EPS almost always results in more iterations to termination.

Input Y

Compute Feature Means. Vector FM(ND) is the mean vector c_y of equation (9a).

Compute Scaling Matrix. Matrix CC(ND, ND) is matrix A of equation (10), depending upon the choice made for ICON. The inverse is constructed in the main to avoid dependence upon peripheral subs. Matrix CM = A^*A^{-1} calculated as a check on the computed inverse, but no residual is calculated; nor does the FCM routine contain a flag if CM is not "close" to I. The construction of weight matrices other than the three choices allowed depends on user definition.

Loop Control. NCLUS = c is the current number of clusters: QQ is the weighting exponent m.

Initial Guess. A pseudo-random initial guess for U_0 is generated in this block at each access.

Cluster Centers. Calculation of current centers V(NC, ND) via equation (11a).

Update Memberships. Calculations with equation (11b); W(NC, ND) is the updated membership matrix. The special situation $m = 1$ is not accounted for here. Many programs are available for this situation for example see Ball (1965). The authors will furnish a listing for hard c-means upon request. Note that this block does not have a transfer in situation $y_k = \hat{v}_i$ for some k and i. This eventuality to our knowledge, has never occurred in nearly 10 years of computing experience. If a check and assignment are desired, the method for assigning $\hat{u}_i$'s in any column k where such a singularity occurs is arbitrary, as long as constraints in equation (2) are satisfied. For example, one may, in this instance, place equal weights (that sum to one) on every row where $y_k = \hat{v}_i$, and zero weights otherwise. This will continue the algorithm, and roundoff error alone should carry the sequence away from such points.

Error Criteria and Cutoffs. The criterion used to terminate iteration at fixed NC is

$$\text{ERRMAX} = \max_{i,k}\{|\hat{u}_{ik}^{(k+1)} - \hat{u}_{ik}^{(k)}|\} < \text{EPS}. \quad (14)$$

Threshold EPS thus controls the accuracy of terminal output. An alternative method to terminate iteration would be to compare components of each $\hat{v}_i^{(k+1)}$ to $\hat{v}_i^{(k)}$. There may be differences in terminal pairs $(\hat{U}, \hat{v})$ obtained using a fixed EPS. Furthermore, there is a tradeoff in CPU time, equation (14) requires (cN) comparisons and $\max_{ij}\{|\hat{v}_{ij}^{(k+1)} - \hat{v}_{(ij)}^{(k)}|\}$ requires (cn) comparisons. Thus, if N is much larger than n, $(N \gg n)$, termination based on the quality of successive cluster centers computed via equation (11a) becomes more attractive. By the same token, this can reduce storage space (for updated centers instead of an updated membership matrix) significantly if $n \ll N$. If equation (14) is never satisfied, iteration at current NC will stop when k = LMAX: a convergence flag is issued, and NC advance to NC + 1. More than 25 iterations are rarely needed for EPS in the 0.001 range.

Cluster Validity Indicants. Values of J_m, F_c, H_c, and $1 - F_c$ are computed, and stored, respectively, in the vectors *VJM*, *F*, *H*, and *DIF*.

Output Block. For the current value of NCLUS, current terminal values of F_c, $1 - F_c$, H_c, J_m, $\{\hat{v}_i\}$, and $\hat{U}$ are printed.

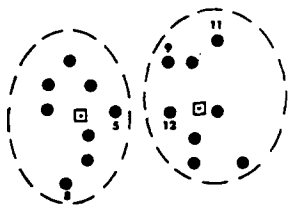
Output Summary. The final block of FCM outputs statistics for the entire run.

The listing provided is a very basic version of FCM: many embellishments are discussed in Bezdek (1981). As an aid for debugging a coded copy of the listing, we present a short example that furnishes a means for checking numerical outputs. This example highlights several of the important features of fuzzy z-partitions in general, and those generated by FCM in particular. Examples of the use of FCM in the context of geological data analysis are presented in Bezdek, and other (1982), and Full, Ehrlich, and Bezdek (1982).

Storage Requirements. The program listed in the appendix can handle 500 data samples with up to 50 variables. It will handle up to 20 clusters. The program, as written, used under 256 K of computer storage. If larger data sets are used, the program is clearly documented as to which parameters to change.

A NUMERICAL EXAMPLE

Figure 1 displays a set Y of 16 points in $\mathbf{R}^2$. This artificial data set was originally published in Sneath



No.	Coordinates y_{k1}	y_{k2}
1	0	4
2	0	3
3	1	5
4	2	4
5	3	3
6	2	2
7	2	1
8	1	0
9	5	5
10	6	5
11	7	6
12	5	3
13	7	3
14	6	2
15	6	1
16	8	1

⊡ = Terminal cluster centers from Table 2, col. 3

) = Terminal maximum membership "boundaries".

Fig. 1. An example: Artificial touching clusters.

and Sokal (1973) in connection with the illustration of a hard clustering algorithm called the unpaired group mean average (UPGMA) method. This data was subsequently studied in Bezdek (1974), where a comparison between the UPGMA and FCM methods was effected. The coordinates of $y_k \in Y$ are listed as columns two and three of the tabular display of Fig. 1. This is a good data set for our purposes because it is easily handled for validation, and further, has some of the geometric properties that necessitate the introduction of fuzzy models. Data of this type might be drawn from a mixture of two bivariate normal distributions. The region of overlap contains several points which might be considered "noise", *viz.* y_5 and y_{12}. Parameters for the outputs to be discussed were as follows:

Table	ICON = A	NCLUS = c	QQ = m	EPS = $\in$
1	1, 2, 3	2	2	0.01
2	2	2	1.25, 2.00	0.01
3	1	2–6	1.25–2.00	0.01

In other words, we illustrate in Tables 1 and 2, respectively, the effects of variation in the norm inducing matrix A, and weighting exponent on $(\hat{U}, \hat{v})$ with all other parameters being fixed; while Table 3 exhibits variations in F_c and H_c due to changes in m and c.

Initial guesses for U_0 were not chosen randomly here, so that users may validate their programs against these tables. Rather, the initial matrix used for all of the outputs discussed later had the following elements:

$$(U_0)_{ii} = \left(\frac{\alpha}{c} + B\right); \quad i = 1, 2, \ldots, c$$

$$(U_0)_{ij} = \left(\frac{\alpha}{c} + \beta\right); \quad j = c + 1, \ldots, n$$

$$(U_0)_{ij} = \left(\frac{\alpha}{c} + \beta\right); \quad \text{otherwise}$$

$$\alpha = 1 - (\sqrt{2}/2);$$

$$\beta = \sqrt{2}/2.$$

The starting value for F_c using this U_0 is always the midpoint of $[1/c, 1)$, the range of F_c, that is $f_c(U_0) = ((1/c) + 1)/2$. In real applications it is, of course, important to run FCM for several different U_0s, as the iteration method used, like all descent methods, is susceptible to local stagnations. If different U_0s result in different $(\hat{U}, \hat{v})$s, one thing is certain: further analysis should be made before one places much confidence in any algorithmically suggested substructure in Y.

Table 1 shows that maximizing F_c is equivalent to minimizing H_c but this behavior is not equivalent to minimizing J_m. Several examples of the general dilemma are documented in Bezdek (1981). Observe that all three partitions of Y are (qualitatively) more or less equivalent. Lower membership generally cor-

Table 1. Variation in $(\hat{U}, \hat{v})$ due to changes in Norm. There are only two clusters, hence $\hat{U}_{2k} = (1 - \hat{U}_{1k})$ as the sum of the $\hat{u}_{ik}$ equals one. Terminal membership U_{ik}

Data Point	ICON = 1 A = I	ICON = 2 A = D_y^{-1}	ICON = 3 A = C_y^{-1}
1	0.92	0.88	0.89
2	0.95	0.93	0.92
3	0.86	0.78	0.82
4	0.91	0.88	0.93
5	0.80	0.84	0.84
6	0.95	0.88	0.82
7	0.86	0.72	0.65
8	0.82	0.67	0.62
9	0.22	0.35	0.43
10	0.12	0.26	0.33
11	0.18	0.32	0.37
12	0.10	0.08	0.09
13	0.02	0.03	0.04
14	0.06	0.09	0.06
15	0.16	0.24	0.19
16	0.15	0.21	0.19
$\hat{v}_{11}$	6.18	5.99	5.96
$\hat{v}_{12}$	3.15	2.95	2.75
$\hat{v}_{21}$	1.44	1.67	1.73
$\hat{v}_{22}$	2.83	3.01	3.19
F_c	0.80	0.71	0.71
H_c	0.35	0.45	0.45
J_m	51.65	13.69	13.69
Iter.	6	6	12

Table 2. Variation in $(\hat{u}, \hat{v})$ due to changes in m (two cluster example). Terminal membership $\hat{u}_{ik}$: $\hat{u}_{2k} = (1 - \hat{u}_{1k})$

Data Point	QQ = m = 1.25	QQ = m = 2.00
1	1.00	0.92
2	1.00	0.95
3	1.00	0.86
4	1.00	0.91
5	1.00	0.80
6	1.00	0.95
7	1.00	0.86
8	1.00	0.82
9	0.00	0.22
10	0.00	0.12
11	0.00	0.18
12	0.00	0.10
13	0.00	0.02
14	0.00	0.06
15	0.00	0.16
16	0.00	0.15
$\hat{v}_{11}$	6.25	6.18
$\hat{v}_{12}$	3.25	3.15
$\hat{v}_{21}$	1.37	1.44
$\hat{v}_{22}$	2.75	2.83
F_c	1.00	0.80
H_c	0.00	0.35
J_m	60.35	51.65
Iter.	4	6

Table 3. Variation in F and H due to changes in m and c.

Weighting Exponent (m)	Number of Clusters (c)	Partition Coefficient (F_c)	Lower Bound ($1-F_c$)	Normalized Entropy (H_c)
1.25	2	0.998	0.002	0.007
	3	0.983	0.017	0.037
	4	0.979	0.021	0.044
	5	0.996	0.004	0.013
1.50	2	0.955	0.045	0.103
	3	0.903	0.097	0.202
	4	0.901	0.099	0.201
	5	0.917	0.083	0.197
1.75	2	0.873	0.127	0.239
	3	0.791	0.209	0.404
	4	0.804	0.196	0.401
	5	0.776	0.224	0.468
2.00	2	0.794	0.206	0.352
	3	0.686	0.314	0.575
	4	0.700	0.300	0.600
	5	0.662	0.338	0.701

responds to points distant from the "core" (i.e. $\hat{v}_i$) of cluster i. Thus, point 8 is clearly signaled an outlier, for example in all three partitions. Notice, however, that the Mahalanobis norm emphasizes this much more heavily than, for example the Euclidean norm. This is because level sets in the former norm are elliptical, and in the latter circular. Thus, the variance of y_8 in the vertical direction weights its influence differently. In all situations, points near cluster centers in the A-norm have higher memberships. Note that $\hat{v}_1$ is more stable to changes in A than v_2: this indicates that points with a high affinity for membership in $\hat{u}_2$ have somewhat more variability than those seeking to associate with $\hat{u}_1$. Table 1 also demonstrates another general fact; the number of iterates needed using the Mahalonobis norm is usually higher than

the number required by other norms. See Bezdek (1981) for more discussion concerning characteristic cluster shapes associated with changes in A.

Table 2 illustrates the usual effect of increasing m lower m's yield harder partitions and higher ones, fuzzier memberships. For $m = 1.25$, $\hat{U}$ is hard (to 2 decimal places). Observe that F_c and H_c mirror this fact, but again, J_m does not, having a higher value at the lower m. Further observe that the cluster centers are rather stable to changes in m. This is not always the situation, and it is an unproven conjecture that the stability of the $\hat{v}_i$'s in the face of severe changes in m is in some sense an indication of cluster validity. Figure 1 exhibits $\hat{v}_1$ and $\hat{v}_2$ for $m = 2 = c$, $A = I$; their geometric positions are at least (visually) appealing.

Table 3 depicts the utility of F_c and H_c for the cluster validity question. For every m, F_c maximizes (and H_c minimizes) at $c = 2$. From this we can infer that the "hardest" substructure detectable in Y occurs are $c = 2$. These values do not, however, have any direct tie to Y. Being computed on algorithmic outputs based on Y rather than any concrete assumptions regarding the distribution of Y somewhat weakens the theoretical plausibility of using F_c and H_c for cluster validity. Nevertheless, they have been demonstrably reliable in many experimental studies, and are, at present, the most reliable indicants of validity for the FCM algorithms.

REFERENCES

Ball, G. 1965, Data analysis in the social sciences: what about the details?: Proc. FJCC, Spartan Books, Washington, D.C., p. 533–560.

Bezdek, J. C. 1974. Mathematical models for systematics and Taxonomy, *in*, Estabrook, G., ed., Proceedings of the 8th International Conference on Numerical Taxonomy: W. H. Freeman, San Francisco, p. 143–166.

Bezdek, J. C., 1980, A convergence theorem for the fuzzy *c*-means clustering algorithms: IEEE Trans. PAMI, PAMI-2(1), p. 1–8.

Bezdek, J. C., 1981, Pattern recognition with fuzzy objective function algorithms: Plenum, New York, 256. p.

Bezdek, J. C., Trivedi, M., Ehrlich, R., and Full, W., 1982, Fuzzy clustering: a new approach for geostatistical analysis: Int. Jour. Sys., Meas., and Decisions.

Duda, R., and Hart, P., 1973, Pattern classification and scene analysis: Wiley-Interscience, New York, 482. p.

Full, W., Ehrlich, R., and Bezdek, J., 1982, FUZZY QMODEL: A new approach for linear unmixing: Jour. Math. Geology.

Full, W., E., Erhlich, R., and Klovan, J. E., 1981, EXTENDED QMODEL—Objective definition of external end members in the analysis of mixtures: Jour. Math. Geology, v. 13, no. 4, p. 331–344.

Hartigan, J., 1975, Clustering algorithms: John Wiley and Sons, New York, 351 p.

Klovan, J., and Imbrie, J., 1971, An algorithm and FORTRAN-IV program for large scale Q-mode factor analysis and calculation of factor Scores: Jour. Math. Geology, v. 3, no. 1, p. 61–76.

Klovan, J. E., and Miesch, A., 1976, EXTENDED CABFAC and QMODEL Computer programs for Q-mode factor analysis of compositional data: Computers & Geosciences, v. 1, no. 3, p. 161–178.

Miesch, A. T., 1976a, Q-mode factor analysis of geochemical and petrologic data matrices with constant row-sums: U.S. Geol. Survey Prof. Paper 574–G, 47 p.

Miesch, A. T., 1976b, Interactive computer programs for petrologic modeling with extended Q-mode factor analysis: Computers & Geosciences, v. 2, no. 2, p. 439–492.

Sneath, P., and Sokal, R., 1973, Numerical taxonomy: Freeman, San Francisco, 573 p.

Tou, J., and Gonzalez, R., 1974, Pattern recognition principles: Addison-Wesley, Reading, Mass., 377 p.

Zadeh, L. A., 1965, Fuzzy sets: Inf. and Cont., v. 8, p. 338–353.

APPENDIX

Listing of fuzzy C-means

```
FILE: KMEANS    FORTRAN  A 03/18/83 11:05  VM/SP CONVERSATIONAL MONITOR SYSTEM


C                                                                       00001000
C                                                                       00002000
C     THIS IS THE FCM (FUZZY C-MEANS) ROUTINE. THIS LISTING IS FOR A    00003000
C     IBM TYPE COMPUTER WITH A FORTRAN IV COMPILER. IT ADAPTS FOR ANY   00004000
C     FORTRAN COMPILER WITH MODIFICATIONS SET AT THE USER SITE.         00005000
C                                                                       00006000
C     REFERENCE: "PATTERN RECOGNITION WITH FUZZY OBJECTIVE FUNCTIONS,"  00007000
C                JAMES BEZDEK, PLENUM, NEW YORK, 1981.                  00008000
C                                                                       00009000
C                                                                       00010000
C     DESCRIPTION OF OPERATING VARIABLES:                               00011000
C     I. INPUT VARIABLES (FROM FILE 5)                                  00012000
C         CARD 1:                                                       00013000
C          TITLE(20).........80 CHARACTER HEADING                       00014000
C         CARD 2:                                                       00015000
C          FMT(20)...........FORTRAN FORMAT (CONTAINED IN PARENTHESIS)  00016000
C                            DESCRIBING THE INPUT FORMAT FOR THE RAW DATA00017000
C                            UP TO 80 CHARACTERS MAY BE USED            00018000
C         CARD 3:                                                       00019000
C          COL 1: ICON......DISTANCE MEASURE TO BE USED. IF:            00020000
C                            ICON=1 USE EUCLIDEAN NORM                  00021000
C                            ICON=2 USE DIAGONAL NORM                   00022000
C                            ICON=3 USE MAHALANOBIS NORM                00023000
C          COLS 2-7: QQ.....WEIGHTING EXPONENT FOR FCM                  00024000
C          COLS 8-9: ND.....NUMBER OF FEATURES PER INPUT VECTOR         00025000
```

```
C          COLS 10-11:KBEGIN.STARTING NUMBER OF CLUSTERS                  00026000
C          COLS 12-13:KCEASE.FINISHING NUMBER OF CLUSTERS (NOTE: KBEGIN   00027000
C                           MUST BE LESS THAN OR EQUAL TO KCEASE)         00028000
C         CARD 4 ON:                                                      00029000
C          Y(NS,ND).........FEATURE VECTORS, INPUT ROW-WISE               00030000
C     II. INTERNAL VARIABLES                                              00031000
C          NS...............NUMBER OF DATA VECTORS                       00032000
C          EPS..............MAXIMUM MEMBERRSHIP ERROR AT CONVERGENCE      00033000
C          NC...............CURRENT NUMBER OF CLUSTERS                    00034000
C          LMAX.............MAXIMUM NUMBER OF ITERATIONS WITHOUT          00035000
C                           CONVERGENCE                                   00036000
C          FM(ND)...........SAMPLE MEAN VECTOR                           00037000
C          FVAR(ND).........VECTOR OF MARGINAL VARIANCES                 00038000
C          CC(ND,ND)........SCALING MATRIX                               00039000
C          AA(ND,ND)........SAMPLE COVARIANCE MATRIX                     00040000
C          AI(ND,ND)........INVERSE OF SAMPLE COVARIANCE MATRIX          00041000
C          BB(ND)...........DUMMY HOLDING MATRIX                         00042000
C          CCC(ND)..........DUMMY HOLDING MATRIX                         00043000
C          ST(ND,ND)........DUMMY HOLDING MATRIX FOR AA                  00044000
C          CM(ND,ND)........CM=AA*(AA INVERSE)                           00045000
C          U(NC,NS).........MEMBERSHIP MATRIX                            00046000
C          W(NC,NS).........UPDATED MEMBERSHIP MATRIX                    00047000
C          V(NC,ND).........CLUSTER CENTERS                              00048000
C          ITT(NC)..........DUMMY HOLDING MATRIX                         00049000
C          H(NC)............ENTROPY MATRIX                               00050000
C          VJM(NC)..........PAYOFF MATRIX                                00051000
C          F(NC)............MATRIX OF PARTITION COEFFICIENTS              00052000
C          DIF(NC)..........MATRIX OF ENTROPY BOUNDS                     00053000
C                                                                         00054000
C                                                                         00055000

FILE: KMEANS   FORTRAN  A 03/18/83 11:05  VM/SP CONVERSATIONAL MONITOR SYSTEM

C*************************************************************************00056000
      DIMENSION FM(50),FVAR(50),F(20)                                     00057000
      DIMENSION BB(50),CCC(50),H(20),DIF(20),ITT(20)                      00058000
      DIMENSION Y(500,2),U(20,500),W(20,500)                              00059000
      DIMENSION AA(50,50),AI(50,50)                                       00060000
      DIMENSION CC(50,50),CM(50,50),ST(50,50)                             00061000
      DIMENSION V(20,50),VJM(20)                                          00062000
      DIMENSION FMT(20),TITLE(20)                                         00063000
      READ(5,1458) (TITLE(I),I=1,20)                                      00064000
1458  FORMAT(20A4)                                                        00065000
      READ(5,12321) (FMT(I),I=1,20)                                       00066000
12321 FORMAT(20A4)                                                        00067000
C-------------------------------------------------------------------------00068000
C     CONTROL PARAMETERS.                                                 00069000
C-------------------------------------------------------------------------00070000
      EPS=.01                                                             00071000
      NS=1                                                                00072000
      LMAX=50                                                             00073000
C-------------------------------------------------------------------------00074000
C     READ FEATURE VECTORS (Y(I,J)).                                      00075000
C-------------------------------------------------------------------------00076000
      READ(5,2021) ICON,QQ,ND,KBEGIN,KCEASE                               00077000
2021  FORMAT(I1,F6.3,3I2)                                                 00078000
      WRITE(6,410)                                                        00079000
410   FORMAT(///1H ,'*** *** BEGIN FUZZY C-MEANS OUTPUT *** ***')         00080000
      WRITE(6,1459) (TITLE(III),III=1,20)                                 00081000
1459  FORMAT(10X,20A4///)                                                 00082000
1     READ(5,399,END=3)(Y(NS,J),J=1,ND)                                   00083000
399   FORMAT (2F1.0)                                                      00084000
      WRITE(6,12738)(Y(NS,J),J=1,ND)                                      00085000
12738 FORMAT(2(10X,10(F7.2,1X)/))                                         00086000
      NS=NS+1                                                             00087000
      GO TO 1                                                             00088000
3     NS=NS-1                                                             00089000
      NDIM=ND                                                             00090000
      NSAMP=NS                                                            00091000
      WRITE(6,11111) NSAMP                                                00092000
11111 FORMAT(10X,'NUMBER OF SAMPLES = ',I5)                               00093000
      ANSAMP=NSAMP                                                        00094000
C-------------------------------------------------------------------------00095000
C     SCALED NORM REQUIRED IN STATEMENTS 31 AND 33.                       00096000
C     CALCULATION OF SCALING MATRIX FOLLOWS.                              00097000
C     FEATURE MEANS.                                                      00098000
C-------------------------------------------------------------------------00099000
```

```
      DO 350 I=1,NDIM                                                  00100000
      FM(I)=0.                                                         00101000
      DO 351 J=1,NSAMP                                                 00102000
351   FM(I)=FM(I)+Y(J,I)                                               00103000
350   FM(I)=FM(I)/ANSAMP                                               00104000
C-----------------------------------------------------------------------00105000
C     FEATURE VARIANCES.                                               00106000
C-----------------------------------------------------------------------00107000
      DO 352 I=1,NDIM                                                  00108000
      FVAR(I)=0.                                                       00109000
      DO 353 J=1,NSAMP                                                 00110000
```

FILE: KMEANS FORTRAN A 03/18/83 11:05 VM/SP CONVERSATIONAL MONITOR SYSTEM

```
353   FVAR(I)=FVAR(I)+((Y(J,I)-FM(I))**2)                              00111000
352   FVAR(I)=FVAR(I)/ANSAMP                                           00112000
      IF(ICON-1)380,380,382                                            00113000
380   DO 381 I=1,NDIM                                                  00114000
      DO 381 J=1,NDIM                                                  00115000
381   CC(I,J)=0.                                                       00116000
      DO 370 I=1,NDIM                                                  00117000
370   CC(I,I)=1.                                                       00118000
      GO TO 390                                                        00119000
382   IF(ICON-2)384,384,386                                            00120000
384   DO 385 I=1,NDIM                                                  00121000
      DO 385 J=1,NDIM                                                  00122000
385   CC(I,J)=0.                                                       00123000
      DO 371 I=1,NDIM                                                  00124000
371   CC(I,I)=1./FM(I)                                                 00125000
      GO TO 390                                                        00126000
386   DO 360 I=1,NDIM                                                  00127000
      DO 360 J=1,NDIM                                                  00128000
      AA(I,J)=0.                                                       00129000
      DO 361 K=1,NSAMP                                                 00130000
361   AA(I,J)=AA(I,J)+((Y(K,I)-FM(I))*(Y(K,J)-FM(J)))                  00131000
360   AA(I,J)=AA(I,J)/ANSAMP                                           00132000
      DO 550 I=1,NDIM                                                  00133000
      DO 550 J=1,NDIM                                                  00134000
550   ST(I,J)=AA(I,J)                                                  00135000
C-----------------------------------------------------------------------00136000
C     INVERSION OF COVARIANCE MATRIX AA TO AI                          00137000
C-----------------------------------------------------------------------00138000
      NN=NDIM-1                                                        00139000
      AA(1,1)=1./AA(1,1)                                               00140000
      DO 500 M=1,NN                                                    00141000
      K=M+1                                                            00142000
      DO 501 I=1,M                                                     00143000
      BB(I)=0.                                                         00144000
      DO 501 J=1,M                                                     00145000
501   BB(I)=BB(I)+AA(I,J)*AA(J,K)                                      00146000
      D=0.                                                             00147000
      DO 502 I=1,M                                                     00148000
502   D=D+AA(K,I)*BB(I)                                                00149000
      D=-D+AA(K,K)                                                     00150000
      AA(K,K)=1./D                                                     00151000
      DO 503 I=1,M                                                     00152000
503   AA(I,K)=-BB(I)*AA(K,K)                                           00153000
      DO 504 J=1,M                                                     00154000
      CCC(J)=0.                                                        00155000
      DO 504 I=1,M                                                     00156000
504   CCC(J)=CCC(J)+AA(K,I)*AA(I,J)                                    00157000
      DO 505 J=1,M                                                     00158000
505   AA(K,J)=-CCC(J)*AA(K,K)                                          00159000
      DO 500 I=1,M                                                     00160000
      DO 500 J=1,M                                                     00161000
500   AA(I,J)=AA(I,J)-BB(I)*AA(K,J)                                    00162000
      DO 520 I=1,NDIM                                                  00163000
      DO 520 J=1,NDIM                                                  00164000
520   AI(I,J)=AA(I,J)                                                  00165000
```

FILE: KMEANS FORTRAN A 03/18/83 11:05 VM/SP CONVERSATIONAL MONITOR SYSTEM

```
      DO 387 I=1,NDIM                                                  00166000
      DO 387 J=1,NDIM                                                  00167000
```

```
387   CC(I,J)=AI(I,J)                                                  00168000
C-----------------------------------------------------------------------00169000
C     CHECK INVERSE AA*AI=I                                            00170000
C-----------------------------------------------------------------------00171000
      DO 530 I=1,NDIM                                                  00172000
      DO 530 J=1,NDIM                                                  00173000
      CM(I,J)=0.                                                       00174000
      DO 530 K=1,NDIM                                                  00175000
530   CM(I,J)=CM(I,J)+ST(I,K)*AI(K,J)                                  00176000
      WRITE(6,531)                                                     00177000
531   FORMAT(' ',//,' CHECK MATRIX AI*AA=1, THE IDENTITY'//)           00178000
      DO 532 I=1,NDIM                                                  00179000
532   WRITE (6,533) (CM(I,J),J=1,NDIM)                                 00180000
533   FORMAT(10X,20F6.2)                                               00181000
390   WRITE(6,1460) (TITLE(III),III=1,20)                              00182000
1460  FORMAT('1',10X,20A4///)                                          00183000
      WRITE(6,420)                                                     00184000
420   FORMAT(' ',///,15X,'SCALING MATRIX CC',///)                      00185000
      DO 421 I=1,NDIM                                                  00186000
421   WRITE(6,422) (CC(I,J),J=1,NDIM)                                  00187000
422   FORMAT(5X,10(F10.1,1X)/5X,10(F10.1,1X)/)                         00188000
      WRITE(6,425)                                                     00189000
425   FORMAT(/////)                                                    00190000
C-----------------------------------------------------------------------00191000
C     QQ IS THE BASIC EXPONENT FOR FUZZY ISODATA.                      00192000
C-----------------------------------------------------------------------00193000
      PP=(1./(QQ-1.))                                                  00194000
      DO 55555 NCLUS=KBEGIN,KCEASE                                     00195000
      WRITE(6,1460) (TITLE(III),III=1,20)                              00196000
      WRITE(6,499) NCLUS,ICON,QQ                                       00197000
499   FORMAT(' ',' NUMBER OF CLUSTERS = ',I3,5X,' ICON = ',I3,5X,      00198000
     C'EXPONENT = ',F4.2,//)                                           00199000
      IT=1                                                             00200000
C-----------------------------------------------------------------------00201000
C     RANDOM INITIAL GUESS FOR U(I,J)                                  00202000
C     THE RANDOM GENERATOR SUBROUTINE RANDU FROM THE IBM SCIENTIFIC    00203000
C     SUBROUTINE PACKAGE (SSP) IS USED AND IS CALLED FROM AN EXTERNAL  00204000
C     LIBRARY. OTHER GENERATORS THAT PRODUCE VALUES ON THE INTERVAL    00205000
C     ZERO TO ONE CAN BE USED.                                         00206000
C-----------------------------------------------------------------------00207000
      RANDOM=.7731                                                     00208000
      IX=1                                                             00209000
      NCLUS1=NCLUS-1                                                   00210000
      DO 1100 K=1,NSAMP                                                00211000
      S=1.0                                                            00212000
      DO 1101 I=1,NCLUS1                                               00213000
C     CALL RANDU(IX,IY,RANDOM)                                         00214000
      RANDOM=RANDOM/2.                                                 00215000
      IX=IY                                                            00216000
      ANC=NCLUS-I                                                      00217000
      U(I,K)=S*(1.0-RANDOM**(1.0/ANC))                                 00218000
1101  S=S-U(I,K)                                                       00219000
1100  U(NCLUS,K)=S                                                     00220000
```

FILE: KMEANS FORTRAN A 03/18/83 11:05 VM/SP CONVERSATIONAL MONITOR SYSTEM

```
C-----------------------------------------------------------------------00221000
C     CALCULATION OF CLUSTER CENTERS V(I).                             00222000
C-----------------------------------------------------------------------00223000
7000  DO 20 I=1,NCLUS                                                  00224000
      DO 20 J=1,NDIM                                                   00225000
      V(I,J)=0.                                                        00226000
      D=0.                                                             00227000
      DO 21 L=1,NSAMP                                                  00228000
      V(I,J)=V(I,J)+((U(I,L)**QQ)*Y(L,J))                              00229000
21    D=D+(U(I,L)**QQ)                                                 00230000
20    V(I,J)=V(I,J)/D                                                  00231000
C-----------------------------------------------------------------------00232000
C     UPDATE MEMBERSHIP FUNCTIONS.                                     00233000
C-----------------------------------------------------------------------00234000
6111  DO 38 I=1,NCLUS                                                  00235000
      DO 38 J=1,NSAMP                                                  00236000
      W(I,J)=0.                                                        00237000
      A=0.                                                             00238000
      DO 31 L=1,NDIM                                                   00239000
      DO 31 M=1,NDIM                                                   00240000
```

```
31    A=A+((Y(J,L)-V(I,L))*CC(L,M)*(Y(J,M)-V(I,M)))                    00241000
      A=1./(A**PP)                                                     00242000
      SUM=0.                                                           00243000
      DO 32 N=1,NCLUS                                                  00244000
      C=0.                                                             00245000
      DO 33 L=1,NDIM                                                   00246000
      DO 33 M=1,NDIM                                                   00247000
33    C=C+((Y(J,L)-V(N,L))*CC(L,M)*(Y(J,M)-V(N,M)))                    00248000
      C=1./(C**PP)                                                     00249000
32    SUM=SUM+C                                                        00250000
      W(I,J)=A/SUM                                                     00251000
38    CONTINUE                                                         00252000
C-----------------------------------------------------------------------00253000
C     ERROR CRITERIA AND CUTOFFS.                                      00254000
C-----------------------------------------------------------------------00255000
9000  ERRMAX=0.                                                        00256000
      DO 40 I=1,NCLUS                                                  00257000
      DO 40 J=1,NSAMP                                                  00258000
      ERR=ABS(U(I,J)-W(I,J))                                           00259000
      IF(ERR.GT.ERRMAX) ERRMAX=ERR                                     00260000
40    CONTINUE                                                         00261000
      WRITE(6,400) IT,ERRMAX,NCLUS                                     00262000
400   FORMAT(1H ,'ITERATION = ',I4,5X,'MAXIMUM ERROR = ',F10.4,        00263000
     110X,'NUMBER OF CLUSTERS = ',I4)                                  00264000
      DO 42 I=1,NCLUS                                                  00265000
      DO 42 J=1,NSAMP                                                  00266000
42    U(I,J)=W(I,J)                                                    00267000
      IF(ERRMAX.LE.EPS) GO TO 6000                                     00268000
43    IT=IT+1                                                          00269000
      IF(IT-LMAX) 7000,7000,6000                                       00270000
C-----------------------------------------------------------------------00271000
C     CALCULATION OF CLUSTER VALIDITY STATISTICS F, H, 1-E             00272000
C-----------------------------------------------------------------------00273000
6000  ITT(NCLUS)=IT                                                    00274000
      F(NCLUS)=0.0                                                     00275000
```

FILE: KMEANS FORTRAN A 03/18/83 11:05 VM/SP CONVERSATIONAL MONITOR SYSTEM

```
      H(NCLUS)=0.0                                                     00276000
      DO 100 I=1,NCLUS                                                 00277000
      DO 100 K=1,NSAMP                                                 00278000
      AU=U(I,K)                                                        00279000
      F(NCLUS)=F(NCLUS)+AU**2/ANSAMP                                   00280000
      IF (AU) 100,100,101                                              00281000
101   H(NCLUS)=H(NCLUS)-AU*ALOG(AU)/ANSAMP                             00282000
100   CONTINUE                                                         00283000
      DIF(NCLUS)=1.0-F(NCLUS)                                          00284000
C-----------------------------------------------------------------------00285000
C     CALCULATION OF OBJECTIVE FUNCTION                                00286000
C-----------------------------------------------------------------------00287000
      A=0.                                                             00288000
      DO 80 I=1,NCLUS                                                  00289000
      DO 80 J=1,NSAMP                                                  00290000
      DIST=0.                                                          00291000
      DO 81 L=1,NDIM                                                   00292000
      DO 81 M=1,NDIM                                                   00293000
81    DIST=DIST+((Y(J,L)-V(I,L))*CC(L,M)*(Y(J,M)-V(I,M)))              00294000
      A=A+((U(I,J)**QQ)*DIST)                                          00295000
80    VJM(NCLUS)=A                                                     00296000
C-----------------------------------------------------------------------00297000
C     OUTPUT BLOCK FOR CURRENT NCLUS                                   00298000
C-----------------------------------------------------------------------00299000
      WRITE(6,401)                                                     00300000
401   FORMAT(' '///' FSTOP',7X,'1-FSTOP',5X,'ENTROPY',5X,'PAYOFF',5X,/) 00301000
      WRITE(6,699) F(NCLUS),DIF(NCLUS),H(NCLUS),VJM(NCLUS)             00302000
699   FORMAT(1H ,2(F6.3,4X),4X,F6.3,5X,E8.3)                           00303000
      WRITE(6,59)                                                      00304000
59    FORMAT(1X,100('-')//)                                            00305000
      WRITE(6,402)                                                     00306000
402   FORMAT(///,15X,'CLUSTER CENTERS V(I,J)',///)                     00307000
      DO 415 I=1,NCLUS                                                 00308000
415   WRITE(6,404) (I,J,V(I,J),J=1,NDIM)                               00309000
404   FORMAT(' I=',I3,3X,'J=',I3,3X,'V(I,J)=',F8.4)                    00310000
405   FORMAT(1H ,7(F6.4,3X))                                           00311000
      WRITE(6,59)                                                      00312000
      WRITE(6,406)                                                     00313000
```

```
406   FORMAT(1H ,///,25X,'MEMBERSHIP FUNCTIONS',///)                    00314000
      DO 407 J=1,NSAMP                                                  00315000
407   WRITE(6,408) J,(U(I,J),I=1,NCLUS)                                 00316000
408   FORMAT(1H ,'J=',I3,5X,8(F6.4,3X))                                 00317000
54444 CONTINUE                                                          00318000
55555 CONTINUE                                                          00319000
C-----------------------------------------------------------------------00320000
C     OUTPUT SUMMARY FOR ALL VALUES OF C                                00321000
C-----------------------------------------------------------------------00322000
      WRITE(6,450)                                                      00323000
450   FORMAT('1',25X,'RUN SUMMARY')                                     00324000
      WRITE(6,460) NSAMP                                                00325000
460   FORMAT(' '///' NUMBER OF SUBJECTS N = ',I4)                       00326000
      WRITE(6,461) NDIM                                                 00327000
461   FORMAT(1HO,'NUMBER OF FEATURES NDIM = ',I4)                       00328000
      WRITE(6,462) EPS                                                  00329000
462   FORMAT(1HO,'MEMBERSHIP DEFECT BOUND EPS = ',F6.4)                 00330000

FILE: KMEANS   FORTRAN  A 03/18/82 11:05  VM/SP CONVERSATIONAL MONITOR SYSTEM

      WRITE(6,464) ICON                                                 00331000
464   FORMAT(1HO,'NORM THIS RUN ICON = ',I1)                            00332000
      WRITE(6,465) QQ                                                   00333000
465   FORMAT(1HO,'WEIGHTING EXPONENT M = ',F4.2)                        00334000
      IF(IT.LE.49) GO TO 476                                            00335000
      WRITE(6,70107)                                                    00336000
70107 FORMAT(' ','CONVERGENCE FLAG: UNABLE TO ACHIEVE SATISFACTORY CLUST00337000
     1ERS AFTER 50 ITERATIONS.')                                        00338000
476   WRITE(6,466)                                                      00339000
466   FORMAT(' '//' NO. OF CLUSTERS',3X,'PART. COEFF.',5X,              00340000
     C'LOWER BOUND',5X,'ENTROPY',5X,'NUMBER OF ITERATIONS')             00341000
      WRITE(6,467)                                                      00342000
467   FORMAT(1HO,6X,'C',17X,'F',15X,'1-F',12X,'H',10X,'IT')             00343000
      DO 468 J=KBEGIN,KCEASE                                            00344000
468   WRITE(6,469) J,F(J),DIF(J),H(J),ITT(J)                            00345000
469   FORMAT(1H ,6X,I2,14X,F6.3,11X,F6.3,7X,F6.3,8X,I4)                 00346000
55556 CONTINUE                                                          00347000
616   WRITE(6,411)                                                      00348000
411   FORMAT(////1H ,'*** *** NORMAL END OF JOB *** ***')               00349000
      STOP                                                              00350000
      END                                                               00351000
```

The Expectation Maximization Algorithm

Frank Dellaert

College of Computing, Georgia Institute of Technology
Technical Report number GIT-GVU-02-20
February 2002

Abstract

This note represents my attempt at explaining the EM algorithm (Hartley, 1958; Dempster et al., 1977; McLachlan and Krishnan, 1997). This is just a slight variation on Tom Minka's tutorial (Minka, 1998), perhaps a little easier (or perhaps not). It includes a graphical example to provide some intuition.

1 Intuitive Explanation of EM

EM is an iterative optimization method to estimate some unknown parameters $\mathbf{\Theta}$, given measurement data $\mathbf{U}$. However, we are not given some "hidden" nuisance variables $\mathbf{J}$, which need to be integrated out. In particular, we want to maximize the posterior probability of the parameters $\mathbf{\Theta}$ given the data $\mathbf{U}$, marginalizing over $\mathbf{J}$:

$$\mathbf{\Theta}^* = \operatorname*{argmax}_{\mathbf{\Theta}} \sum_{\mathbf{J} \in \mathcal{J}^n} P(\mathbf{\Theta}, \mathbf{J}|\mathbf{U}) \tag{1}$$

The intuition behind EM is an old one: alternate between estimating the unknowns $\mathbf{\Theta}$ and the hidden variables $\mathbf{J}$. This idea has been around for a long time. However, instead of finding the best $\mathbf{J} \in \mathcal{J}$ given an estimate $\mathbf{\Theta}$ at each iteration, EM computes a *distribution* over the space $\mathcal{J}$. One of the earliest papers on EM is (Hartley, 1958), but the seminal reference that formalized EM and provided a proof of convergence is the "DLR" paper by Dempster, Laird, and Rubin (Dempster et al., 1977). A recent book devoted entirely to EM and applications is (McLachlan and Krishnan, 1997), whereas (Tanner, 1996) is another popular and very useful reference.

One of the most insightful explanations of EM, that provides a deeper understanding of its operation than the intuition of alternating between variables, is in terms of lower-bound maximization (Neal and Hinton, 1998; Minka, 1998). In this derivation, the E-step can be interpreted as constructing a local lower-bound to the posterior distribution, whereas the M-step optimizes the bound, thereby improving the estimate for the unknowns. This is demonstrated below for a simple example.

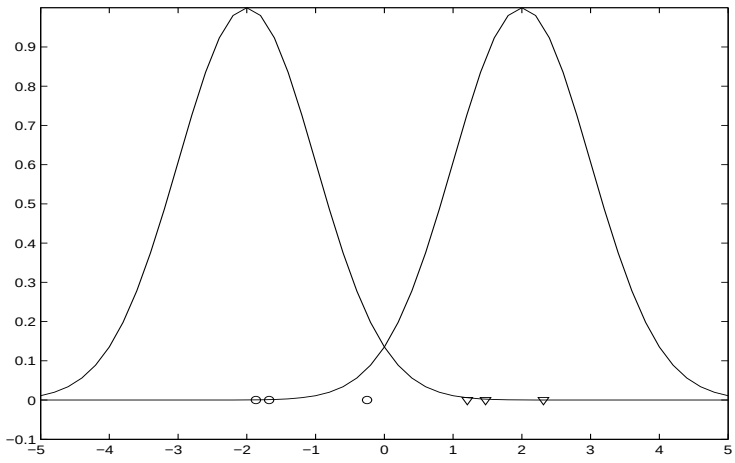


Figure 1: EM example: Mixture components and data. The data consists of three samples drawn from each mixture component, shown above as circles and triangles. The means of the mixture components are -2 and 2, respectively.

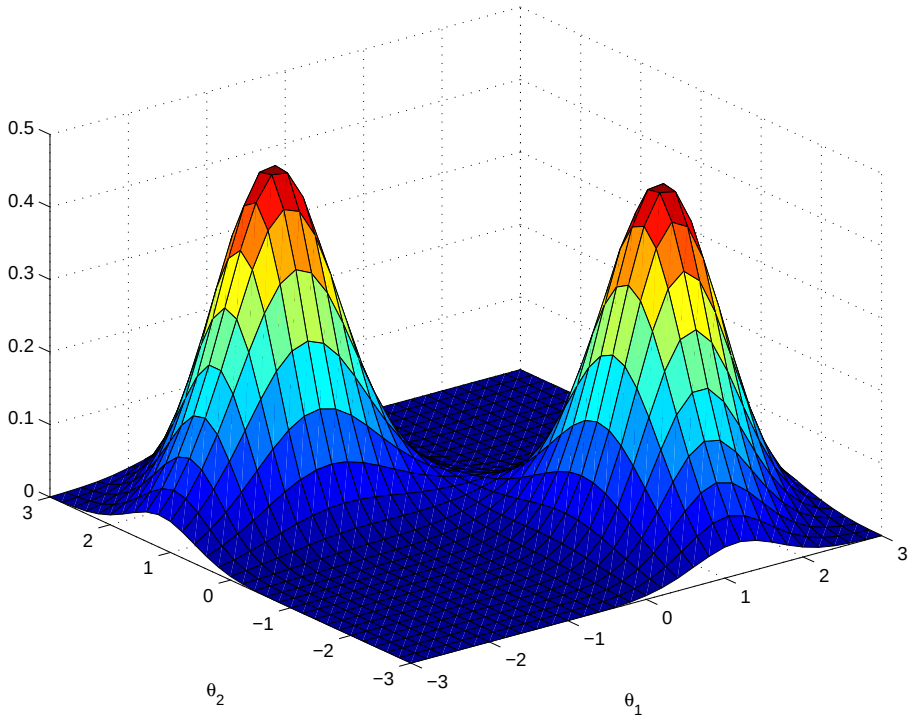


Figure 2: The true likelihood function of the two component means θ_1 and θ_2, given the data in Figure 1.

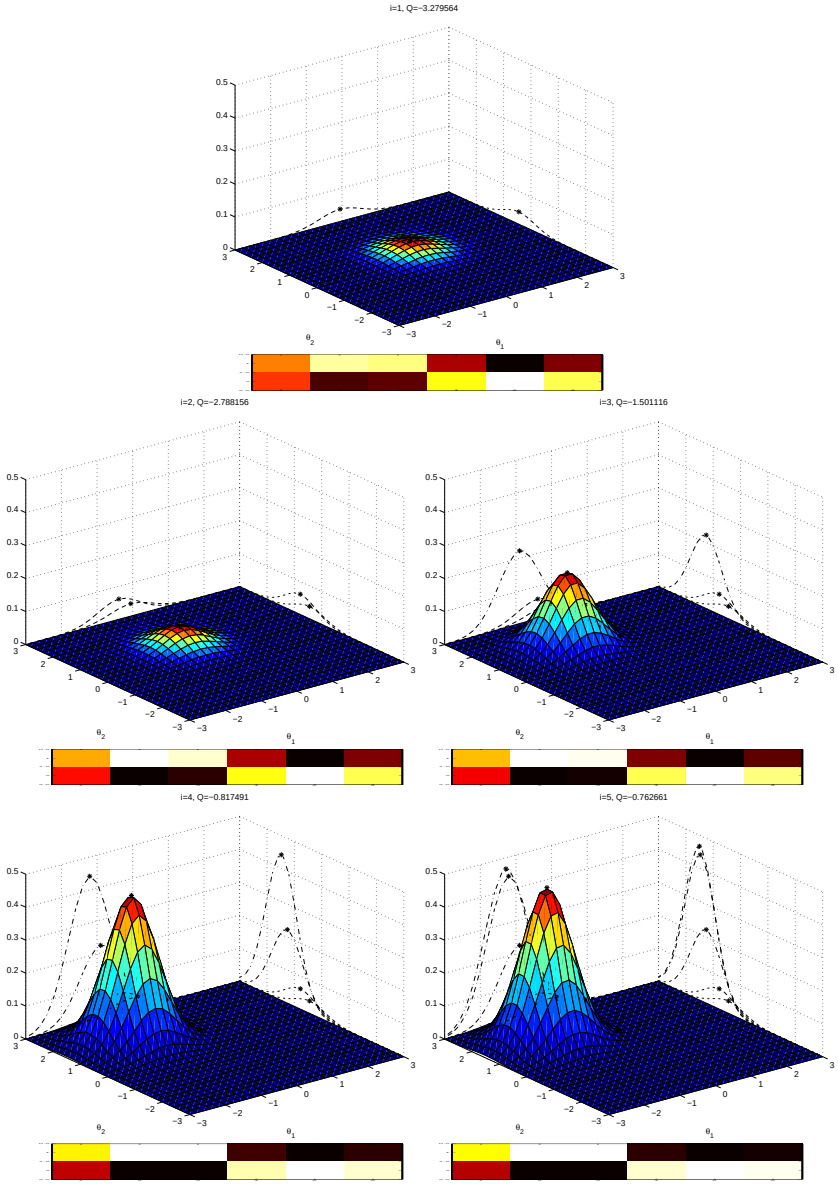


Figure 3: Lower Bounds

Consider the mixture estimation problem shown in Figure 1, where the goal is to estimate the two component means θ_1 and θ_2 given 6 samples drawn from the mixture, but *without knowing from which mixture each sample was drawn*. The state space is two-dimensional, and the true likelihood function is shown in Figure 2. Note that there are two modes, located respectively at $(-2, 2)$ and $(2, -2)$. This makes perfect sense, as we can switch the mixture components without affecting the quality of the solution. Note also that the true likelihood is computed by integrating over all possible data associations, and hence we can find a maximum likelihood solution without solving a correspondence problem. However, even for only 6 samples, this requires summing over the space of 64 possible data-associations.

EM proceeds as follows in this example. In the E-step, a "soft" assignment is computed that assigns a posterior probability to each possible association of each individual sample. In the current example, there are 2 mixtures and 6 samples, so the computed probabilities can be represented in a 2×6 table. Given these probabilities, EM computes a tight lower bound to the true likelihood function of Figure 2. The bound is constructed such that it touches the likelihood function at the current estimate, and it is only close to the true likelihood in the neighborhood of this estimate. The bound and its corresponding probability table is computed in each iteration, as shown in Figure 3. In this case, EM was run for 5 iterations. In the M-step, the lower bound is maximized (shown by a black asterisk in the figure), and the corresponding new estimate (θ_1, θ_2) is guaranteed to lie closer to the location of the nearest local maximum of the likelihood. Each next bound is an increasingly better approximation to the mode of the likelihood, until at convergence the bound touches the likelihood at the local maximum, and progress can no longer be made. This is shown in the last panel of Figure 3.

2 EM as Lower Bound Maximization

EM can be derived in many different ways, one of the most insightful being in terms of lower bound maximization (Neal and Hinton, 1998; Minka, 1998), as illustrated with the example from Section 1. In this section, we derive the EM algorithm on that basis, closely following (Minka, 1998).

The goal is to maximize the posterior probability (1) of the parameters $\mathbf{\Theta}$ given the data $\mathbf{U}$, in the presence of hidden data $\mathbf{J}$. Equivalently, we can maximize the logarithm of the joint distribution (which is proportional to the posterior):

$$\mathbf{\Theta}^* = \operatorname*{argmax}_{\mathbf{\Theta}} \ \log P(\mathbf{U}, \mathbf{\Theta}) = \operatorname*{argmax}_{\mathbf{\Theta}} \ \log \sum_{\mathbf{J} \in \mathcal{J}^n} P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}) \tag{2}$$

The idea behind EM is to start with a guess $\mathbf{\Theta}^t$ for the parameters $\mathbf{\Theta}$, compute an easily computed lower bound $B(\mathbf{\Theta}; \mathbf{\Theta}^t)$ to the function $\log P(\mathbf{\Theta}|\mathbf{U})$, and maximize that bound instead. If iterated, this procedure will converge to a local maximizer $\mathbf{\Theta}^*$ of the objective function, provided the bound improves at each iteration.

To motivate this, note that the key problem with maximizing (2) is that it involves the logarithm of a (big) sum, which is difficult to deal with. Fortunately, we can construct

a tractable lower bound $B(\mathbf{\Theta}; \mathbf{\Theta}^t)$ that instead contains a sum of logarithms. To derive the bound, first trivially rewrite $\log P(\mathbf{U}, \mathbf{\Theta})$ as

$$\log P(\mathbf{U}, \mathbf{\Theta}) = \log \sum_{\mathbf{J} \in \mathcal{J}^n} P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}) = \log \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \frac{P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta})}{f^t(\mathbf{J})}$$

where $f^t(\mathbf{J})$ is an arbitrary probability distribution over the space $\mathcal{J}^n$ of hidden variables $\mathbf{J}$. By Jensen's inequality, we have

$$B(\mathbf{\Theta}; \mathbf{\Theta}^t) \triangleq \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \log \frac{P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta})}{f^t(\mathbf{J})} <= \log \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \frac{P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta})}{f^t(\mathbf{J})}$$

Note that we have transformed a log of sums into a sum of logs, which was the prime motivation.

2.1 Finding an Optimal Bound

EM goes one step further and tries to find the *best* bound, defined as the bound $B(\mathbf{\Theta}; \mathbf{\Theta}^t)$ that touches the objective function $\log P(\mathbf{U}, \mathbf{\Theta})$ at the current guess $\mathbf{\Theta}^t$. Intuitively, finding the best bound at each iteration will guarantee that we obtain an improved estimate $\mathbf{\Theta}^{t+1}$ when we locally maximize the bound with respect to $\mathbf{\Theta}$. Since we know $B(\mathbf{\Theta}; \mathbf{\Theta}^t)$ to be a lower bound, the optimal bound at $\mathbf{\Theta}^t$ can be found by maximizing

$$B(\mathbf{\Theta}^t; \mathbf{\Theta}^t) = \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \log \frac{P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}^t)}{f^t(\mathbf{J})} \tag{3}$$

with respect to the distribution $f^t(\mathbf{J})$. Introducing a Lagrange multiplier λ to enforce the constraint $\sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) = 1$, the objective becomes

$$G(f^t) = \lambda \left[1 - \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \right] + \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \log P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}^t) - \sum_{\mathbf{J} \in \mathcal{J}^n} f^t(\mathbf{J}) \log f^t(\mathbf{J})$$

Taking the derivative

$$\frac{\partial G}{\partial f^t(\mathbf{J})} = -\lambda + \log P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}^t) - \log f^t(\mathbf{J}) - 1$$

and solving for $f^t(\mathbf{J})$ we obtain

$$f^t(\mathbf{J}) = \frac{P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}^t)}{\sum_{\mathbf{J} \in \mathcal{J}^n} P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}^t)} = P(\mathbf{J} | \mathbf{U}, \mathbf{\Theta}^t)$$

By examining the value of the resulting optimal bound at $\mathbf{\Theta}^t$ we see that it indeed touches the objective function:

$$B(\mathbf{\Theta}^t; \mathbf{\Theta}^t) = \sum_{\mathbf{J} \in \mathcal{J}^n} P(\mathbf{J} | \mathbf{U}, \mathbf{\Theta}^t) \log \frac{P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}^t)}{P(\mathbf{J} | \mathbf{U}, \mathbf{\Theta}^t)} = \log P(\mathbf{U}, \mathbf{\Theta}^t)$$

2.2 Maximizing The Bound

In order to maximize $B(\mathbf{\Theta}; \mathbf{\Theta}^t)$ with respect to $\mathbf{\Theta}$, note that we can write it as

$$\begin{aligned} B(\mathbf{\Theta}; \mathbf{\Theta}^t) &\stackrel{\Delta}{=} \langle \log P(\mathbf{U}, \mathbf{J}, \mathbf{\Theta}) \rangle + \mathcal{H} \\ &= \langle \log P(\mathbf{U}, \mathbf{J} | \mathbf{\Theta}) \rangle + \log P(\mathbf{\Theta}) + \mathcal{H} \\ &= Q^t(\mathbf{\Theta}) + \log P(\mathbf{\Theta}) + \mathcal{H} \end{aligned}$$

where $\langle . \rangle$ denotes the expectation with respect to $f^t(\mathbf{J}) \stackrel{\Delta}{=} P(\mathbf{J}|\mathbf{U}, \mathbf{\Theta}^t)$, and

- $Q^t(\mathbf{\Theta})$ is the expected complete log-likelihood, defined as:

$$Q^t(\mathbf{\Theta}) \stackrel{\Delta}{=} \langle \log P(\mathbf{U}, \mathbf{J} | \mathbf{\Theta}) \rangle$$

- $P(\mathbf{\Theta})$ is the prior on the parameters $\mathbf{\Theta}$
- $\mathcal{H} \stackrel{\Delta}{=} - \langle \log f^t(\mathbf{J}) \rangle$ is the entropy of the distribution $f^t(\mathbf{J})$

Since $\mathcal{H}$ does not depend on $\mathbf{\Theta}$, we can maximize the bound with respect to $\mathbf{\Theta}$ using the first two terms only:

$$\mathbf{\Theta}^{t+1} = \underset{\mathbf{\Theta}}{\text{argmax}} \; B(\mathbf{\Theta}; \mathbf{\Theta}^t) = \underset{\mathbf{\Theta}}{\text{argmax}} \; \left[Q^t(\mathbf{\Theta}) + \log P(\mathbf{\Theta}) \right] \tag{4}$$

2.3 The EM Algorithm

At each iteration, the EM algorithm first finds an optimal lower bound $B(\mathbf{\Theta}; \mathbf{\Theta}^t)$ at the current guess $\mathbf{\Theta}^t$ (equation 3), and then maximizes this bound to obtain an improved estimate $\mathbf{\Theta}^{t+1}$ (equation 4). Because the bound is expressed as an expectation, the first step is called the "expectation-step" or E-step, whereas the second step is called the "maximization-step" or M-step. The EM algorithm can thus be conveniently summarized as:

- E-step: calculate $f^t(\mathbf{J}) \stackrel{\Delta}{=} P(\mathbf{J}|\mathbf{U}, \mathbf{\Theta}^t)$
- M-step: $\mathbf{\Theta}^{t+1} = \text{argmax}_{\mathbf{\Theta}} \left[Q^t(\mathbf{\Theta}) + \log P(\mathbf{\Theta}) \right]$

It is important to remember that $Q^t(\mathbf{\Theta})$ is calculated in the E-step by evaluating $f^t(\mathbf{J})$ using the *current guess* $\mathbf{\Theta}^t$ (hence the superscript t), whereas in the M-step we are optimizing $Q^t(\mathbf{\Theta})$ with respect to the *free variable* $\mathbf{\Theta}$ to obtain the new estimate $\mathbf{\Theta}^{t+1}$. It can be proved that the EM algorithm converges to a local maximum of $\log P(\mathbf{U}, \mathbf{\Theta})$, and thus equivalently maximizes the log-posterior $\log P(\mathbf{\Theta}|\mathbf{U})$ (Dempster et al., 1977; McLachlan and Krishnan, 1997).

A Relation to the Expected Log-Posterior

Note that we have chosen to define $Q^t(\mathbf{\Theta})$ as the expected log-likelihood as in (Dempster et al., 1977; McLachlan and Krishnan, 1997), i.e.,

$$Q^t(\mathbf{\Theta}) \triangleq \langle \log P(\mathbf{U}, \mathbf{J}|\mathbf{\Theta}) \rangle$$

An alternative route is to compute the expected log-posterior (Tanner, 1996):

$$\langle \log P(\mathbf{\Theta}|\mathbf{U}, \mathbf{J}) \rangle = \langle \log P(\mathbf{U}, \mathbf{J}|\mathbf{\Theta}) + \log P(\mathbf{\Theta}) - \log P(\mathbf{U}, \mathbf{J}) \rangle \tag{5}$$

Here the second term does not depend on $\mathbf{J}$ and can be taken out of the expectation, and the last term does not depend on $\mathbf{\Theta}$. Hence, maximizing (5) with respect to $\mathbf{\Theta}$ is equivalent to (4):

$$\begin{aligned} \operatorname*{argmax}_{\mathbf{\Theta}} \ \langle \log P(\mathbf{\Theta}|\mathbf{U}, \mathbf{J}) \rangle &= \operatorname*{argmax}_{\mathbf{\Theta}} \ [\langle \log P(\mathbf{U}, \mathbf{J}|\mathbf{\Theta}) \rangle + \log P(\mathbf{\Theta})] \\ &= \operatorname*{argmax}_{\mathbf{\Theta}} \ \left[Q^t(\mathbf{\Theta}) + \log P(\mathbf{\Theta})\right] \end{aligned}$$

This is of course identical to (4).

References

[1] Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

[2] Hartley, H. (1958). Maximum likelihood estimation from incomplete data. *Biometrics*, 14:174–194.

[3] McLachlan, G. and Krishnan, T. (1997). *The EM algorithm and extensions*. Wiley series in probability and statistics. John Wiley & Sons.

[4] Minka, T. (1998). Expectation-Maximization as lower bound maximization. Tutorial published on the web at http://www-white.media.mit.edu/ tpminka/papers/em.html.

[5] Neal, R. and Hinton, G. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M., editor, *Learning in Graphical Models*. Kluwer Academic Press.

[6] Tanner, M. (1996). *Tools for Statistical Inference*. Springer Verlag, New York. Third Edition.

CS838-1 Advanced NLP: The EM Algorithm

Xiaojin Zhu

2007
Send comments to jerryzhu@cs.wisc.edu

"Nice intuitions have nice mathematical explanations."

"Not every iterative procedure can be called EM."

1 Naive Bayes Revisited

Recall in Naive Bayes models, we are given a training set $\{(x, y)_{1:n}\}$, and our goal is to train a classifier that classifies any new document x into one of K classes. In the case of MLE, this is achieved by estimating parameters $\Theta = \{\pi, \theta_1, \ldots, \theta_K\}$, where $p(y = k) = \pi_k$, and $p(x|y = k) = \theta_k$, to maximize the joint log likelihood of the training data:

$$\Theta = \arg \max_{\pi, \theta_1, \ldots, \theta_K} \log p(\{(x, y)_{1:n}\}|\pi, \theta_1, \ldots, \theta_K). \tag{1}$$

The solution is

$$\begin{aligned} \pi_k &= \frac{\sum_{i=1}^n [y_i = k]}{n}, \quad k = 1, \ldots, K \\ \theta_{kw} &= \frac{\sum_{i:y_i=k} x_{iw}}{\sum_{i:y_i=k} \sum_{u=1}^V x_{iu}}, \quad k = 1, \ldots, K. \end{aligned} \tag{2}$$

Note π is a probability vector of length K over classes, and each θ_k is a probability vector of length V over the vocabulary.

Classification is done by computing $p(y|x)$:

$$\hat{y} = \arg \max_k p(y = k|x) \tag{3}$$

$$= \arg \max_k p(y = k)p(x|y = k) \quad ; \text{Bayes rule, ignore constant denominator} \tag{4}$$

$$= \arg \max_k \pi_k \prod_{w=1}^V \theta_{kw}^{x_w} \tag{5}$$

$$= \arg \max_k \log \pi_k + \sum_{w=1}^V x_w \log \theta_{kw} \quad ; \log \text{ is monotonic.} \tag{6}$$

2 K-means Clustering

So far so good. What if the training data is *mostly unlabeled*? For example, let's assume we have only one labeled example per class:

$$\{(x_1, y_1 = 1), (x_2, y_2 = 2), \ldots, (x_K, y_K = K), x_{K+1}, \ldots, x_n\}.$$

We can simply ignore unlabeled data $\{x_{K+1}, \ldots, x_n\}$ and train our Naive Bayes classifier on labeled data. Can we do better?

Here is an iterative procedure, known as *K-means clustering*, which we apply to our classification task:

1. Estimate $\Theta^{(t=0)}$ from labeled examples only.
2. Repeat until things no longer change:
   (a) Classify all examples (labeled and unlabeled) x by its most likely class under the current model: $\hat{y} = \arg\max_k p(y = k|x, \Theta^{(t)})$.
   (b) Now we have a fully labeled dataset $\{(x, \hat{y})_{1:n}\}$. Retrain $\Theta^{(t+1)}$ on it. Let $t = t + 1$.

There are a couple details:

- We re-classify all the labeled points. This is mostly for notational convenience. It is certainly fine (and probably more desirable) to fix their labels during the iterations. The derivation follows similarly, with separate terms for the labeled data.
- We use the K-means clustering algorithm for classification. But it was originally designed for clustering. For example, one can randomly pick $\theta_{1:K}$ to start with, and the algorithm will converge to K final clusters. In that case, we do not have the correspondence between clusters and classes.
- K-means clustering is usually presented on mixture of Gaussian distributions. Here we instead have mixture of multinomial distributions. In either case, θ_k is the 'centroid' of cluster k. The distance is measured differently though [1].

3 The EM Algorithm

In K-means we made a *hard* classification: each x is assigned a unique label $\hat{y}$. A *soft* version would be to use the posterior $p(y|x)$. Intuitively, each x_i is split into K copies, but copy k has weight $\gamma_{ik} = p(y_i = k|x_i)$ instead of one. We now have a dataset with fractional counts, but that poses no difficulty in retraining the parameters. One can show that the MLE is the weighted version of (2)

$$\begin{aligned} \pi_k &= \frac{\sum_{i=1}^n \gamma_{ik}}{n}, \quad k = 1, \ldots, K \\ \theta_{kw} &= \frac{\sum_{i=1}^n \gamma_{ik} x_{iw}}{\sum_{i=1}^n \sum_{u=1}^V \gamma_{ik} x_{iu}}, \quad k = 1, \ldots, K. \end{aligned} \tag{7}$$

The change from hard to soft leads us to the EM (Expectation Maximization) algorithm:

1. Estimate $\Theta^{(t=0)}$ from labeled examples only.
2. Repeat until convergence:
   (a) **E-step**: For $i = 1 \ldots n, k = 1 \ldots K$, compute $\gamma_{ik} = p(y_i = k|x_i, \Theta^{(t)})$.
   (b) **M-step**: Compute $\Theta^{(t+1)}$ from (7). Let $t = t + 1$.

4 Analysis of EM

EM might look like a heuristic method. However, as we show now, it has a rigorous foundation: EM is guaranteed to find a local optimum of data log likelihood.

Recall if we have complete data $\{(x, y)_{1:n}\}$ (as in standard Naive Bayes), the joint log likelihood under parameters Θ is

$$\log p((x, y)_{1:n}|\Theta) = \sum_{i=1}^{n} \log p(y_i|\Theta)p(x_i|y_i, \Theta). \tag{8}$$

However, now $y_{1:n}$ are hidden variables. We instead maximize the *marginal* log likelihood

$$\begin{aligned} \ell(\Theta) &= \log p(x_{1:n}|\Theta) && (9)\\ &= \sum_{i=1}^{n} \log p(x_i|\Theta) && (10)\\ &= \sum_{i=1}^{n} \log \sum_{y=1}^{K} p(x_i, y|\Theta) && (11)\\ &= \sum_{i=1}^{n} \log \sum_{y=1}^{K} p(y|\Theta)p(x_i|y, \Theta). && (12) \end{aligned}$$

We note that there is a summation *inside* the log. This couples the Θ parameters. If we try to maximize the marginal log likelihood by setting the gradient to zero, we will find that there is no longer a nice closed form solution, unlike the joint log likelihood with complete data. The reader is encouraged to attempt this to see the difference.

EM is an iterative procedure to maximize the marginal log likelihood $\ell(\Theta)$. It constructs a concave, easy-to-optimize lower bound $Q(\Theta, \Theta^{(t)})$, where Θ is the variable and $\Theta^{(t)}$ is the previous, fixed, parameter. The lower bound has an interesting property $Q(\Theta^{(t)}, \Theta^{(t)}) = \ell(\Theta^{(t)})$. Therefore the new parameter $\Theta^{(t+1)}$ that maximizes Q is guaranteed to have $Q \geq \ell(\Theta^{(t)})$. Since Q lower bounds ℓ, we have $\ell(\Theta^{(t+1)}) \geq \ell(\Theta^{(t)})$.

The lower bound is obtained via *Jensen's inequality*

$$\log \sum_i p_i f_i \geq \sum_i p_i \log f_i, \tag{13}$$

which holds if the p_i's form a probability distribution (i.e., non-negative and sum to 1). This follows from the concavity of log.

$$\begin{aligned}
\ell(\Theta) &= \sum_{i=1}^{n} \log \sum_{y=1}^{K} p(x_i, y|\Theta) && (14)\\
&= \sum_{i=1}^{n} \log \sum_{y=1}^{K} p(y|x_i, \Theta^{(t)}) \frac{p(x_i, y|\Theta)}{p(y|x_i, \Theta^{(t)})} && (15)\\
&\geq \sum_{i=1}^{n} \sum_{y=1}^{K} p(y|x_i, \Theta^{(t)}) \log \frac{p(x_i, y|\Theta)}{p(y|x_i, \Theta^{(t)})} && (16)\\
&\equiv Q(\Theta, \Theta^{(t)}). && (17)
\end{aligned}$$

Note we introduced a probability distribution $p(y|x_i, \Theta^{(t)}) \equiv \gamma_{iy}$ separately for each example x_i. This is what E-step is computing.

The M-step maximizes the lower bound $Q(\Theta, \Theta^{(t)})$. It is worth noting that now we can set the gradient of Q to zero and obtain a closed form solution. In fact the solution is simply (7), and we call it $\Theta^{(t+1)}$.

It is easy to see that

$$Q(\Theta^{(t)}, \Theta^{(t)}) = \sum_{i=1}^{n} \log p(x_i|\Theta^{(t)}) = \ell(\Theta^{(t)}). \tag{18}$$

Since $\Theta^{(t+1)}$ maximizes Q, we have

$$Q(\Theta^{(t+1)}, \Theta^{(t)}) \geq Q(\Theta^{(t)}, \Theta^{(t)}) = \ell(\Theta^{(t)}). \tag{19}$$

On the other hand, Q lower bounds ℓ. Therefore

$$\ell(\Theta^{(t+1)}) \geq Q(\Theta^{(t+1)}, \Theta^{(t)}) \geq Q(\Theta^{(t)}, \Theta^{(t)}) = \ell(\Theta^{(t)}). \tag{20}$$

This shows that $\Theta^{(t+1)}$ is indeed a better (or no worse) parameter than $\Theta^{(t)}$ in terms of the marginal log likelihood ℓ. By iterating, we arrive at a local maximum of ℓ.

5 Deeper Analysis of EM

You might have noticed that we never referred to the concrete model $p(x|y), p(y)$ (Naive Bayes) in the above analysis, except when we say the solution is simply (7). Does this suggest that EM is more general than Naive Bayes? Besides, where did the particular probability distribution $p(y|x_i, \Theta^{(t)})$ come from in (15)?

The answer to the first question is yes. EM applies to joint probability models where some random variables are missing. It is advantageous when the marginal is hard to optimize, but the joint is. To be general, consider a joint distribution $p(X, Z|\Theta)$, where X is the collection of observed variables, and Z unobserved variables. The quantity we want to maximize is the marginal log likelihood

$$\ell(\Theta) \equiv \log p(X|\Theta) = \log \sum_Z p(X, Z|\Theta), \tag{21}$$

which we assume to be difficult. One can introduce an arbitrary distribution over hidden variables $q(Z)$,

$$\begin{aligned}
\ell(\Theta) &= \sum_Z q(Z) \log P(X|\Theta) && (22)\\
&= \sum_Z q(Z) \log \frac{P(X|\Theta)q(Z)P(X, Z|\Theta)}{P(X, Z|\Theta)q(Z)} && (23)\\
&= \sum_Z q(Z) \log \frac{P(X, Z|\Theta)}{q(Z)} + \sum_Z q(Z) \log \frac{P(X|\Theta)q(Z)}{P(X, Z|\Theta)} && (24)\\
&= \sum_Z q(Z) \log \frac{P(X, Z|\Theta)}{q(Z)} + \sum_Z q(Z) \log \frac{q(Z)}{P(Z|X, \Theta)} && (25)\\
&= F(\Theta, q) + KL(q(Z)\|p(Z|X, \Theta)). && (26)
\end{aligned}$$

Note $F(\Theta, q)$ is the RHS of Jensen's inequality. Since $KL \geq 0$, $F(\Theta, q)$ is a lower bound of $\ell(\Theta)$.

First consider the maximization of F on q with $\Theta^{(t)}$ fixed. $F(\Theta^{(t)}, q)$ is maximized by $q(Z) = p(Z|X, \Theta^{(t)})$ since $\ell(\Theta)$ is fixed and KL attends its minimum zero. This is why we picked the particular distribution $p(Z|X, \Theta^{(t)})$. This is the E-step.

Next consider the maximization of F on Θ with q fixed as above. Note in this case $F(\Theta, q) = Q(\Theta, \Theta^{(t)})$. This is the M-step.

Therefore the EM algorithm can be viewed as coordinate ascent on q and Θ to maximize F, a lower bound of ℓ.

Viewed this way, EM is a particular optimization method. There are several variations of EM:

- Generalized EM (GEM) finds Θ that improves, but not necessarily maximizes, $F(\Theta, q) = Q(\Theta, \Theta^{(t)})$ in the M-step. This is useful when the exact M-step is difficult to carry out. Since this is still coordinate ascent, GEM can find a local optimum.

- Stochastic EM: The E-step is computed with Monte Carlo sampling. This introduces randomness into the optimization, but asymptotically it will converge to a local optimum.

- Variational EM: $q(Z)$ is restricted to some easy-to-compute subset of distributions, for example the fully factorized distributions $q(Z) = \prod_i q(z_i)$.

In general $p(Z|X, \Theta^{(t)})$, which might be intractable to compute, will not be in this subset. There is no longer guarantee that variational EM will find a local optimum.

References

[1] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

CURE: An Efficient Clustering Algorithm for Large Databases

Sudipto Guha*
Stanford University
Stanford, CA 94305
sudipto@cs.stanford.edu

Rajeev Rastogi
Bell Laboratories
Murray Hill, NJ 07974
rastogi@bell-labs.com

Kyuseok Shim
Bell Laboratories
Murray Hill, NJ 07974
shim@bell-labs.com

Abstract

Clustering, in data mining, is useful for discovering groups and identifying interesting distributions in the underlying data. Traditional clustering algorithms either favor clusters with spherical shapes and similar sizes, or are very fragile in the presence of outliers. We propose a new clustering algorithm called CURE that is more robust to outliers, and identifies clusters having non-spherical shapes and wide variances in size. CURE achieves this by representing each cluster by a certain fixed number of points that are generated by selecting *well scattered* points from the cluster and then shrinking them toward the center of the cluster by a specified fraction. Having more than one representative point per cluster allows CURE to adjust well to the geometry of non-spherical shapes and the shrinking helps to dampen the effects of outliers. To handle large databases, CURE employs a combination of *random sampling* and *partitioning*. A random sample drawn from the data set is first partitioned and each partition is *partially* clustered. The partial clusters are then clustered in a second pass to yield the desired clusters. Our experimental results confirm that the quality of clusters produced by CURE is much better than those found by existing algorithms. Furthermore, they demonstrate that random sampling and partitioning enable CURE to not only outperform existing algorithms but also to scale well for large databases without sacrificing clustering quality.

1 Introduction

The wealth of information embedded in huge databases belonging to corporations (e.g., retail, financial, telecom) has spurred a tremendous interest in the areas of *knowledge discovery* and *data mining*. Clustering, in data mining, is a useful technique for discovering interesting data distributions and patterns in the underlying data. The problem of clustering can be defined as follows: given n data points in a d-dimensional metric space, partition the data points into k clusters such that the data points within a cluster are more similar to each other than data points in different clusters.

1.1 Traditional Clustering Algorithms - Drawbacks

Existing clustering algorithms can be broadly classified into *partitional* and *hierarchical* [JD88]. Partitional clustering algorithms attempt to determine k partitions that optimize a certain criterion function. The square-error criterion, defined below, is the most commonly used (m_i is the mean of cluster C_i).

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \|p - m_i\|^2.$$

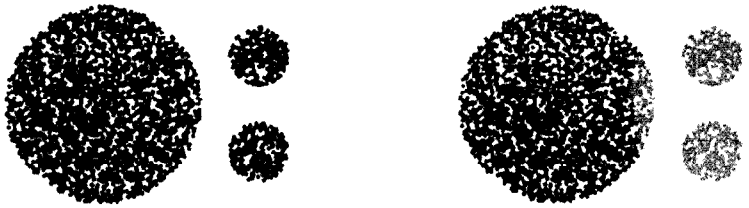
The square-error is a good measure of the within-cluster variation across all the partitions. The objective is to find k partitions that minimize the square-error. Thus, square-error clustering tries to make the k clusters as compact and separated as possible, and works well when clusters are compact clouds that are rather well separated from one another. However, when there are large differences in the sizes or geometries of different clusters, as illustrated in Figure 1, the square-error method could split large clusters to minimize the square-error. In the figure, the square-error is larger for the three separate clusters in (a) than for the three clusters in (b) where the big cluster is split into three portions, one of which is merged with the two smaller clusters. The reduction in square-error for (b) is due to the fact that the slight reduction in square error due to splitting the large cluster is weighted by many data points in the large cluster.

A hierarchical clustering is a sequence of partitions in which each partition is nested into the next partition in the sequence. An *agglomerative* algorithm for hierarchical clustering starts with the disjoint set of clusters, which places each input data point in an individual cluster. Pairs of items or clusters are then successively merged until the number of clusters reduces to k. At each step, the pair of clusters merged are the ones between which the distance is the minimum. The widely used measures for distance between clusters are as follows (m_i is the mean for cluster C_i and n_i is the number of points in C_i).

$$\begin{aligned} d_{mean}(C_i, C_j) &= \|m_i - m_j\| \\ d_{ave}(C_i, C_j) &= 1/(n_i n_j) \sum_{p \in C_i} \sum_{p' \in C_j} \|p - p'\| \\ d_{max}(C_i, C_j) &= \max_{p \in C_i, p' \in C_j} \|p - p'\| \end{aligned}$$

*The work was done while the author was visiting Bell Laboratories.

SIGMOD '98 Seattle, WA, USA



(a) (b)

Figure 1: Splitting of a large cluster by partitional algorithms

$$d_{min}(C_i, C_j) = \min_{p \in C_i, p' \in C_j} \|p - p'\|$$

For example, with d_{mean} as the distance measure, at each step, the pair of clusters whose centroids or means are the closest are merged. On the other hand, with d_{min}, the pair of clusters merged are the ones containing the closest pair of points. All of the above distance measures have a minimum variance flavor and they usually yield the same results if the clusters are compact and well-separated. However, if the clusters are close to one another (even by outliers), or if their shapes and sizes are not hyperspherical and uniform, the results of clustering can vary quite dramatically. For example, with the data set shown in Figure 1(a), using d_{max}, d_{ave} or d_{mean} as the distance measure results in clusters that are similar to those obtained by the square-error method shown in Figure 1(b). Similarly, consider the example data points in Figure 2. The desired elongated clusters are shown in Figure 2(a). However, d_{mean} as the distance measure, causes the elongated clusters to be split and portions belonging to neighboring elongated clusters to be merged. The resulting clusters are as shown in Figure 2(b). On the other hand, with d_{min} as the distance measure, the resulting clusters are as shown in Figure 2(c). The two elongated clusters that are connected by narrow string of points are merged into a single cluster. This "chaining effect" is a drawback of d_{min} – basically, a few points located so as to form a bridge between the two clusters causes points across the clusters to be grouped into a single elongated cluster.

From the above discussion, it follows that neither the centroid-based approach (that uses d_{mean}) nor the all-points approach (based on d_{min}) work well for non-spherical or arbitrary shaped clusters. A shortcoming of the centroid-based approach is that it considers only one point as representative of a cluster – the cluster centroid. For a large or arbitrary shaped cluster, the centroids of its subclusters can be reasonably far apart, thus causing the cluster to be split. The all-points approach, on the other hand, considers all the points within a cluster as representative of the cluster. This other extreme, has its own drawbacks, since it makes the clustering algorithm extremely sensitive to outliers and to slight changes in the position of data points.

When the number N of input data points is large, hierarchical clustering algorithms break down due to their non-linear time complexity (typically, $O(N^2)$) and huge I/O costs. In order to remedy this problem, in [ZRL96], the authors propose a new clustering method named BIRCH, which represents the state of the art for clustering large data sets. BIRCH first performs a *preclustering phase* in which dense regions of points are represented by compact summaries, and then a centroid-based hierarchical algorithm is used to cluster the set of summaries (which is much smaller than the original dataset).

The preclustering algorithm employed by BIRCH to reduce input size is incremental and approximate. During preclustering, the entire database is scanned, and cluster summaries are stored in memory in a data structure called the CF-tree. For each successive data point, the CF-tree is traversed to find the closest cluster to it in the tree, and if the point is within a threshold distance of the closest cluster, it is absorbed into it. Otherwise, it starts its own cluster in the CF-tree.

Once the clusters are generated, a final labeling phase is carried out in which using the centroids of clusters as seeds, each data point is assigned to the cluster with the closest seed. Using only the centroid of a cluster when redistributing the data in the final phase has problems when clusters do not have uniform sizes and shapes as in Figure 3(a). In this case, as illustrated in Figure 3(b), in the final labeling phase, a number of points in the bigger cluster are labeled as belonging to the smaller cluster since they are closer to the centroid of the smaller cluster.

1.2 Our Contributions

In this paper, we propose a new clustering method named CURE (Clustering Using Representatives) whose salient features are described below.

Hierarchical Clustering Algorithm: CURE employs a novel hierarchical clustering algorithm that adopts a middle ground between the centroid-based and the all-point extremes. In CURE, a constant number c of *well scattered* points in a cluster are first chosen. The scattered points capture the shape and extent of the cluster. The chosen scattered points are next shrunk towards the centroid of the cluster by a fraction α. These scattered points after shrinking are used as representatives of the cluster. The clusters with the closest pair of representative points are the clusters that are merged at each step of CURE's hierarchical clustering algorithm.

The scattered points approach employed by CURE alleviates the shortcomings of both the all-points as well as the centroid-based approaches. It enables CURE to correctly identify the clusters in Figure 2(a) – the resulting clusters due to the centroid-based and all-points approaches is as shown in Figures 2(b) and 2(c), respectively. CURE is less sensitive to outliers since shrinking the scattered points toward the mean dampens the adverse effects due to outliers – outliers are typically further away from the mean and are thus shifted a larger distance due to the shrinking. Multiple scattered points also enable CURE to discover non-spherical clusters like the elongated clusters shown in Figure 2(a). For the centroid-based algorithm, the space that constitutes the vicinity of the single centroid for a cluster is spherical. Thus, it favors spherical clusters and as shown in Figure 2(b), splits the elongated clusters. On the other hand, with multiple scattered points as representatives of a cluster, the space

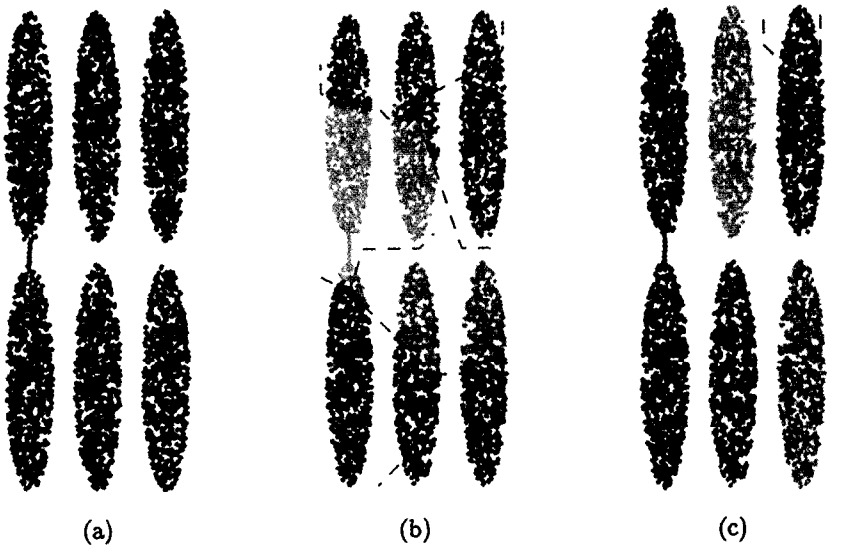


Figure 2: Clusters generated by hierarchical algorithms

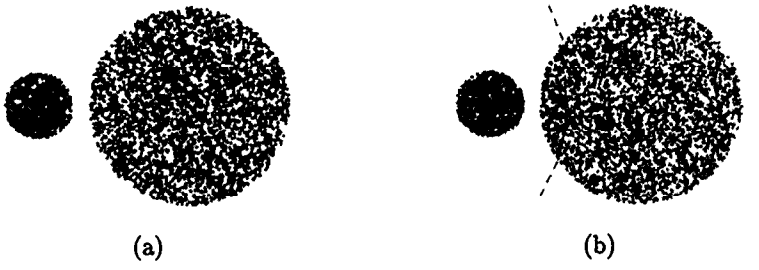


Figure 3: Problem of labeling

that forms the vicinity of the cluster can be non-spherical, and this enables CURE to correctly identify the clusters in Figure 2(a).

Note that the kinds of clusters identified by CURE can be tuned by varying α between 0 and 1. CURE reduces to the centroid-based algorithm if $\alpha = 1$, while for $\alpha = 0$, it becomes similar to the all-points approach. CURE's hierarchical clustering algorithm uses space that is linear in the input size n and has a worst-case time complexity of $O(n^2 \log n)$. For lower dimensions (e.g., two), the complexity can be shown to further reduce to $O(n^2)$. Thus, the time complexity of CURE is no worse than that of the centroid-based hierarchical algorithm.

Random Sampling and Partitioning: CURE's approach to the clustering problem for large data sets differs from BIRCH in two ways. First, instead of preclustering with all the data points, CURE begins by drawing a random sample from the database. We show, both analytically and experimentally, that random samples of moderate sizes preserve information about the geometry of clusters fairly accurately, thus enabling CURE to correctly cluster the input. In particular, assuming that each cluster has a certain minimum size, we use chernoff bounds to calculate the minimum sample size for which the sample contains, with high probability, at least a fraction f of every cluster. Second, in order to further speed up clustering, CURE first partitions the random sample and partially clusters the data points in each partition. After eliminating outliers, the preclustered data in each partition is then clustered in a final pass to generate the final clusters.

Labeling Data on Disk: Once clustering of the random sample is completed, instead of a single centroid, multiple representative points from each cluster are used to label the remainder of the data set. The problems with BIRCH's labeling phase are eliminated by assigning each data point to the cluster containing the closest representative point.

Overview: The steps involved in clustering using CURE are described in Figure 4. Our experimental results confirm that not only does CURE's novel approach to clustering based on scattered points, random sampling and partitioning enable it to find clusters that traditional clustering algorithms fail to find, but it also results in significantly better execution times.

The remainder of the paper is organized as follows. In Section 2, we survey related work on clustering large data sets. We present CURE's hierarchical clustering algorithm that uses representative points, in Section 3. In Section 4, we discuss issues related to sampling, partitioning, outlier handling and labeling in CURE. Finally, in Section 5, we present the results of our experiments which support our claims about CURE's clustering ability and execution times. Concluding remarks are made in Section 6.

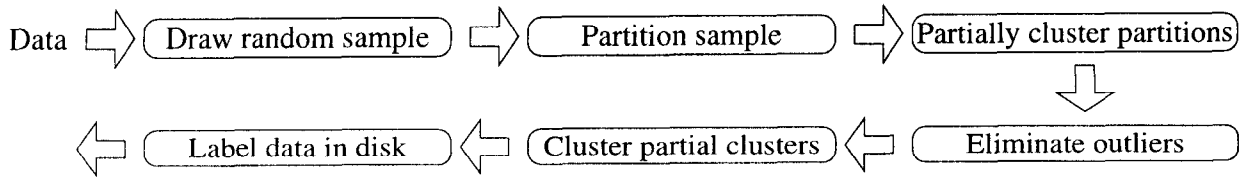


Figure 4: Overview of CURE

2 Related Work

In recent years, a number of clustering algorithms for large databases have been proposed [NH94, ZRL96, EKSX96]. In [NH94], the authors propose a partitional clustering method for large databases which is based on randomized search. Each cluster is represented by its *medoid*, the most centrally located point in the cluster, and the objective is to find the k best medoids that optimize the criterion function. The authors reduce this problem to that of graph search by representing each set of k medoids as a node in the graph, two nodes being adjacent if they have $k-1$ medoids in common. Initially, an arbitrary node is set to be the current node and a fixed number of iterations are performed. In each iteration, a random neighbor of the current node is set to be the current node if it results in better clustering. The computation of the criterion function for the random neighbor requires the entire database to be examined. It is experimentally shown that CLARANS outperforms the traditional *k-medoid* algorithms. However, CLARANS may require several passes over the database, the runtime cost of which could be prohibitive for large databases. Furthermore, like other partitional clustering algorithms, it could converge to a local optimum.

In [EKX95], the authors use the R*-tree[SRF87, BKSS90, Sam89] to improve the I/O efficiency of CLARANS on large databases by (1) drawing samples from leaf pages to reduce the number of data points (since data points are packed in leaf nodes based on spatial locality, a sample point in the leaf page can be a good representative point), and (2) focusing on relevant points when evaluating the "goodness" of a neighbor.

Since multiple I/O scans of the data points is a bottleneck for existing clustering algorithms, in [ZRL96], the authors present a clustering method named BIRCH whose I/O complexity is a little more than one scan of the data. BIRCH first pre-clusters the data into the maximum possible and finest possible subclusters that can fit in main-memory. For the pre-clustering phase, BIRCH employs a CF-tree which is a balanced tree structure similar to the B-tree and R-tree family[Sam89]. After pre-clustering, BIRCH treats each of the subcluster summaries as representative points, and runs a well-known approximation algorithm from [Ols93], which is an agglomerative hierarchical clustering algorithm.

BIRCH and CLARANS work well for convex or spherical clusters of uniform size. However, they are unsuitable when clusters have different sizes (see Figure 1), or when clusters are non-spherical (see Figure 2). For clustering such arbitrary shaped collections of points (e.g., ellipsoid, spiral, cylindrical), a density-based algorithm called DBSCAN was proposed in [EKSX96]. DBSCAN requires the user to specify two parameters that are used to define minimum density for clustering – the radius *Eps* of the neighborhood of a point and the minimum number of points *MinPts* in the neighborhood. Clusters are then found by starting from an arbitrary point and if its neighborhood satisfies the minimum density, including the points in its neighborhood into the cluster. The process is then repeated for the newly added points.

While DBSCAN can find clusters with arbitrary shapes, it suffers from a number of problems. DBSCAN is very sensitive to the parameters *Eps* and *MinPts*, which in turn, are difficult to determine. Furthermore, DBSCAN also suffers from the robustness problems that plague the all-points hierarchical clustering algorithm – in case there is a dense string of points connecting two clusters, DBSCAN could end up merging the two clusters. Also, DBSCAN does not perform any sort of preclustering and executes directly on the entire database. As a result, for large databases, DBSCAN could incur substantial I/O costs. Finally, with density-based algorithms, using random sampling to reduce the input size may not be feasible – the reason for this is that unless sample sizes are large, there could be substantial variations in the density of points within each cluster in the random sample.

3 Hierarchical Clustering Algorithm

In this section, we present CURE's hierarchical clustering algorithm whose salient features are: (1) the clustering algorithm can recognize arbitrarily shaped clusters (e.g., ellipsoidal), (2) the algorithm is robust to the presence of outliers, and (3) the algorithm has linear storage requirements and time complexity of $O(n^2)$ for low-dimensional data. The n data points input to the algorithm are either a sample drawn randomly from the original data points, or a subset of it if partitioning is employed. An analysis of issues related to the size of the random sample and number of partitions is presented in Section 4.

3.1 Intuition and Overview

The clustering algorithm starts with each input point as a separate cluster, and at each successive step merges the closest pair of clusters. In order to compute the distance between a pair of clusters, for each cluster, c representative points are stored. These are determined by first choosing c *well scattered* points within the cluster, and then shrinking them toward the mean of the cluster by a fraction α. The distance between two clusters is then the distance between the closest pair of representative points – one belonging to each of the two clusters. Thus, only the representative points of a cluster are used to compute its distance from other clusters.

The c representative points attempt to capture the physical shape and geometry of the cluster. Furthermore, shrinking the scattered points toward the mean by a factor α gets rid of surface abnormalities and mitigates the effects of outliers. The reason for this is that outliers typically will be further away from the cluster center, and as a result, the shrinking would cause outliers to move more toward the center while the remaining representative points would experience minimal shifts. The larger movements in the outliers would thus reduce their ability to cause the wrong clusters to

```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.     u := extract_min(Q)
5.     v := u.closest
6.     delete(Q, v)
7.     w := merge(u, v)
8.     delete_rep(T, u); delete_rep(T, v); insert_rep(T, w)
9.     w.closest := x /* x is an arbitrary cluster in Q */
10.    for each x ∈ Q do {
11.       if dist(w, x) < dist(w, w.closest)
12.          w.closest := x
13.       if x.closest is either u or v {
14.          if dist(x, x.closest) < dist(x, w)
15.             x.closest := closest_cluster(T, x, dist(x, w))
16.          else
17.             x.closest := w
18.          relocate(Q, x)
19.       }
20.       else if dist(x, x.closest) > dist(x, w) {
21.          x.closest := w
22.          relocate(Q, x)
23.       }
24.    }
25.    insert(Q, w)
26. }
end
```

Figure 5: Clustering algorithm

be merged. The parameter α can also be used to control the shapes of clusters. A smaller value of α shrinks the scattered points very little and thus favors elongated clusters. On the other hand, with larger values of α, the scattered points get located closer to the mean, and clusters tend to be more compact.

3.2 Clustering Algorithm

In this subsection, we describe the details of our clustering algorithm (see Figure 5). The input parameters to our algorithm are the input data set S containing n points in d-dimensional space and the desired number of clusters k. As we mentioned earlier, starting with the individual points as individual clusters, at each step the closest pair of clusters is merged to form a new cluster. The process is repeated until there are only k remaining clusters.

Data Structures: With every cluster is stored all the points in the cluster. Also, for each cluster u, u.mean and u.rep store the mean of the points in the cluster and the set of c representative points for the cluster, respectively. For a pair of points p, q, $dist(p, q)$ denotes the distance between the points. This distance could be any of the L_p metrics like L_1 ("manhattan") or L_2 ("euclidean") metrics. Alternatively, nonmetric *similarity functions* can also be used. The distance between two clusters u and v can then be defined as

$$dist(u, v) = \min_{p \in u.\text{rep}, q \in v.\text{rep}} dist(p, q)$$

For every cluster u, we keep track of the cluster closest to it in u.closest.

```
procedure merge(u, v)
begin
1.  w := u ∪ v
2.  w.mean := (|u|u.mean + |v|v.mean) / (|u| + |v|)
3.  tmpSet := ∅
4.  for i := 1 to c do {
5.     maxDist := 0
6.     foreach point p in cluster w do {
7.        if i = 1
8.           minDist := dist(p, w.mean)
9.        else
10.          minDist := min{dist(p, q) : q ∈ tmpSet}
11.       if (minDist ≥ maxDist){
12.          maxDist := minDist
13.          maxPoint := p
14.       }
15.    }
16.    tmpSet := tmpSet ∪ {maxPoint}
17. }
18. foreach point p in tmpSet do
19.    w.rep := w.rep ∪ {p + α*(w.mean-p) }
20. return w
end
```

Figure 6: Procedure for merging clusters

The algorithm makes extensive use of two data structures – a heap[CLR90] and a k-d tree[Sam90]. Furthermore, corresponding to every cluster, there exists a single entry in the heap – the entries for the various clusters u are arranged in the heap in the increasing order of the distances between u and u.closest. The second data structure is a k-d tree that stores the representative points for every cluster. The k-d tree is a data structure for efficiently storing and retrieving multi-dimensional point data. It is a binary search tree with the distinction that a different key value is tested at each level of the tree to determine the branch to traverse further. For example, for two dimensional data points, the first dimension of the point is tested at even levels (assuming the root node is level 0) while the second dimension is tested at odd levels. When a pair of clusters is merged, the k-d tree is used to compute the closest cluster for clusters that may previously have had one of the merged clusters as the closest cluster.

Clustering procedure: Initially, for each cluster u, the set of representative points u.rep contains only the point in the cluster. Thus, in Step 1, all input data points are inserted into the k-d tree. The procedure build_heap (in Step 2) treats each input point as a separate cluster, computes u.closest for each cluster u and then inserts each cluster into the heap (note that the clusters are arranged in the increasing order of distances between u and $u.closest$).

Once the heap Q and tree T are initialized, in each iteration of the while-loop, until only k clusters remain, the closest pair of clusters is merged. The cluster u at the top of the heap Q is the cluster for which u and $u.closest$ are the closest pair of clusters. Thus, for each step of the while-loop, extract_min (in Step 4) extracts the top element u in Q and also deletes u from Q. The merge procedure (see Figure 6) is then used to merge the closest pair of clusters u and v, and to compute new representative points for the new merged cluster w which are subsequently inserted into

T (in Step 8). The points in cluster w are simply the union of the points in the two clusters u and v that were merged. The merge procedure, in the for-loop (Steps 4-17), first iteratively selects c well-scattered points. In the first iteration, the point farthest from the mean is chosen as the first scattered point. In each subsequent iteration, a point from the cluster w is chosen that is farthest from the previously chosen scattered points. The points are then shrunk toward the mean by a fraction α in Step 19 of the merge procedure.

For the merged cluster w, since the set of representative points for it could have changed (a new set of representative points is computed for it), we need to compute its distance to every other cluster and set w.closest to the cluster closest to it (see Steps 11 and 12 of the cluster procedure). Similarly, for a different cluster x in Q, x.closest may change and x may need to be relocated in Q (depending on the distance between x and x.closest). A brute-force method for determining the closest cluster to x is to compute its distance with every other cluster (including w). However, this would require $O(n)$ steps for each cluster in Q, and could be computationally expensive and inefficient. Instead, we observe that the expensive computation of determining the closest cluster is not required for every cluster x. For the few cases that it is required, we use T to determine this efficiently in $O(\log n)$ steps per case. We can classify the clusters in Q into two groups. The first group of clusters are those who had either u or v as the closest cluster before u and v were merged. The remaining clusters in Q constitute the second group. For a cluster x in the first group, if the distance to w is smaller than its distance to the previously closest cluster (say u), then all we have to do is simply set w to be the closest cluster (see Step 17). The reason for this is that we know that the distance between x and every other cluster is greater than the distance between x and u. The problem arises when the distance between x and w is larger than the distance between x and u. In this case, any of the other clusters could become the new closest cluster to x. The procedure closest_cluster (in Step 15) uses the tree T to determine the closest cluster to cluster x. For every point p in x.rep, T is used to determine the nearest neighbor to p that is not in x.rep. From among the nearest neighbors, the one that is closest to one of x's representative points is determined and the cluster containing it is returned as the closest cluster to x. Since we are not interested in clusters whose distance from x is more than $dist(x, w)$, we pass this as a parameter to closest_cluster which uses it to make the search for nearest neighbors more efficient. Processing a cluster x in the second group is much simpler – x.closest already stores the closest cluster to x from among existing clusters (except w). Thus, if the distance between x and w is smaller than x's distance to it's previously closest cluster, x.closest, then w becomes the closest cluster to x (see Step 21); otherwise, nothing needs to be done. In case x.closest for a cluster x is updated, then since the distance between x and its closest cluster may have changed, x may need to be relocated in the heap Q (see Steps 18 and 22).

An improved merge procedure: In the merge procedure, the overhead of choosing representative points for the merged cluster can be reduced as follows. The merge procedure, in the outer for-loop (Step 4), chooses c scattered points from among all the points in the merged cluster w. Instead, suppose we selected the c scattered points for w from the $2c$ scattered points for the two clusters u and v being merged (the original scattered points for clusters u and v can be obtained by unshrinking their representative points by α). Then, since at most $2c$ points, instead of $O(n)$ points, need to be examined every time a scattered point is chosen, the complexity of the merge procedure reduces to $O(1)$. Furthermore, since the scattered points for w are chosen from the original scattered points for clusters u and v, they can be expected to be fairly well spread out.

3.3 Time and Space Complexity

The worst-case time complexity of our clustering algorithm can be shown to be $O(n^2 \log n)$. In [GRS97], we show that when the dimensionality of data points is small, the time complexity further reduces to $O(n^2)$. Since both the heap and the k-d tree require linear space, it follows that the space complexity of our algorithm is $O(n)$.

4 Enhancements for Large Data Sets

Most hierarchical clustering algorithms, including the one presented in the previous subsection, cannot be directly applied to large data sets due to their quadratic time complexity with respect to the input size. In this section, we present enhancements and optimizations that enable CURE to handle large data sets. We also address the issue of outliers and propose schemes to eliminate them.

4.1 Random Sampling

In order to handle large data sets, we need an effective mechanism for reducing the size of the input to CURE's clustering algorithm. One approach to achieving this is via *random sampling* – the key idea is to apply CURE's clustering algorithm to a random sample drawn from the data set rather than the entire data set. Typically, the random sample will fit in main-memory and will be much smaller than the original data set. Consequently, significant improvements in execution times for CURE can be realized. Also, random sampling can improve the quality of clustering since it has the desirable effect of filtering outliers.

Efficient algorithms for drawing a sample randomly from data in a file in one pass and using constant space are proposed in [Vit85]. As a result, we do not discuss sampling in any further detail, and assume that we employ one of the well-known algorithms for generating the random sample. Also, our experience has been that generally, the overhead of generating a random sample is very small compared to the time for performing clustering on the sample (the random sampling algorithm typically takes less than two seconds to sample a few thousand points from a file containing hundred thousand or more points).

Of course, one can argue that the reduction in input size due to sampling has an associated cost. Since we do not consider the entire data set, information about certain clusters may be missing in the input. As a result, our clustering algorithms may miss out certain clusters or incorrectly identify certain clusters. Even though random sampling does have this tradeoff between accuracy and efficiency, our experimental results indicate that for most of the data sets that we considered, with moderate sized random samples, we were able to obtain very good clusters. In addition, we can use chernoff bounds to analytically derive values for sample sizes for which the probability of missing clusters is low.

We are interested in answering the following question: what should the size s of the random sample be so that the probability of missing clusters is low ? One assumption that we will make is that the probability of missing a cluster u

is low if the sample contains at least $f|u|$ points from the sample, where $0 \leq f \leq 1$. This is a reasonable assumption to make since clusters will usually be densely packed and a subset of the points in the cluster is all that is required for clustering. Furthermore, the value of f depends on the cluster density as well as the intercluster separation – the more well-separated and the more dense clusters become, the smaller is the fraction of the points from each cluster that we need for clustering. *Chernoff bounds* [MR95] can be used to prove the following theorem.

Theorem 4.1: *For a cluster u, if the sample size s satisfies*

$$s \geq fN + \frac{N}{|u|}\log(\frac{1}{\delta}) + \frac{N}{|u|}\sqrt{(\log(\frac{1}{\delta}))^2 + 2f|u|\log(\frac{1}{\delta})} \quad (1)$$

then the probability that the sample contains fewer than $f|u|$ points belonging to cluster u is less than δ, $0 \leq \delta \leq 1$. ∎

Proof: See [GRS97]. ∎

Thus, based on the above equation, we conclude that for the sample to contain at least $f|u|$ points belonging to cluster u (with high probability), we need the sample to contain more than a fraction f of the total number of points – which seems intuitive. Also, suppose u_{min} is the smallest cluster that we are interested in, and s_{min} is the result of substituting $|u_{min}|$ for $|u|$ in the right hand side of Equation (1). It is easy to observe that Equation (1) holds for $s = s_{min}$ and all $|u| \geq |u_{min}|$. Thus, with a sample of size s_{min}, we can guarantee that with a high probability, $1 - \delta$, the sample contains at least $f|u|$ points from an arbitrary cluster u. Also, assuming that there are k clusters, with a sample size of s_{min}, the probability of selecting fewer than $f|u|$ points from any one of the clusters u is bounded above by $k\delta$.

4.2 Partitioning for Speedup

As the separation between clusters decreases and as clusters become less densely packed, samples of larger sizes are required to distinguish them. However, as the input size n grows, the computation that needs to be performed by CURE's clustering algorithm could end up being fairly substantial due to the $O(n^2 \log n)$ time complexity. In this subsection, we propose a simple partitioning scheme for speeding up CURE when input sizes become large.

The basic idea is to partition the sample space into p partitions, each of size $\frac{n}{p}$. We then partially cluster each partition until the final number of clusters in each partition reduces to $\frac{n}{pq}$ for some constant $q > 1$. Alternatively, we could stop merging clusters in a partition if the distance between the closest pair of clusters to be merged next increases above a certain threshold. Once we have generated $\frac{n}{pq}$ clusters for each partition, we then run a second clustering pass on the $\frac{n}{q}$ partial clusters for all the partitions (that resulted from the first pass).

The idea of partially clustering each partition achieves a sort of *preclustering* – schemes for which were also proposed in [ZRL96]. The preclustering algorithm in [ZRL96] is incremental and scans the entire data set. Each successive data point becomes part of the closest existing cluster if it is within some threshold distance τ from it – else, it forms a new cluster. Thus, while [ZRL96] applies an incremental and approximate clustering algorithm to all the points, CURE uses it's hierarchical clustering algorithm only on the points in a partition. In some abstract sense, CURE's partitioning scheme behaves like a sieve working in a bottom-up fashion and filtering out individual points in favor of partial clusters that are then processed during the second pass.

The advantage of partitioning the input in the above mentioned fashion is that we can reduce execution times by a factor of approximately $\frac{q-1}{pq} + \frac{1}{q^2}$. The reason for this is that the complexity of clustering any one partition is $O(\frac{n^2}{p^2}(\frac{q-1}{q})\log\frac{n}{p})$ since the number of points per partition is $\frac{n}{p}$ and the number of merges that need to be performed for the number of clusters to reduce to $\frac{n}{pq}$ is $\frac{n}{p}(\frac{q-1}{q})$. Since there are p such partitions, the complexity of the first pass becomes $O(\frac{n^2}{p}(\frac{q-1}{q})\log\frac{n}{p})$. The time complexity of clustering the $\frac{n}{q}$ clusters in the second pass is $O(\frac{n^2}{q^2}\log\frac{n}{q})$. Thus, the complexity of CURE's partitioning algorithm is $O(\frac{n^2}{p}(\frac{q-1}{q})\log\frac{n}{p} + \frac{n^2}{q^2}\log\frac{n}{q})$, which corresponds to an improvement factor of approximately $\frac{q-1}{pq} + \frac{1}{q^2}$ over clustering without partitioning.

An important point to note is that, in the first pass, the closest points in each partition are merged only until the final number of clusters reduces to $\frac{n}{pq}$. By ensuring that $\frac{n}{pq}$ is sufficiently large compared to the number of desired clusters, k, we can ensure that even though each partition contains fewer points from each cluster, the closest points merged in each partition generally belong to the same cluster and do not span clusters. Thus, we can ensure that partitioning does not adversely impact clustering quality. Consequently, the best values for p and q are those that maximize the improvement factor $\frac{q-1}{pq} + \frac{1}{q^2}$ while ensuring that $\frac{n}{pq}$ is at least 2 or 3 times k.

The partitioning scheme can also be employed to ensure that the input set to the clustering algorithm is always in main-memory even though the random sample itself may not fit in memory. If the partition size is chosen to be smaller than the main-memory size, then the input points for clustering during the first pass are always main-memory resident. The problem is with the second pass since the size of the input is the size of the random sample itself. The reason for this is that for every cluster input to the second clustering pass, we store all the points in the cluster. Clearly, this is unnecessary, since our clustering algorithm only relies on the representative points for each cluster. Furthermore, the improved merge procedure in Section 3 only uses representative points of the previous clusters when computing the new representative points for the merged cluster. Thus, by storing only the representative points for each cluster input to the second pass, we can reduce the input size for the second clustering pass and ensure that it fits in main-memory.

4.3 Labeling Data on Disk

Since the input to CURE's clustering algorithm is a set of randomly sampled points from the original data set, the final k clusters involve only a subset of the entire set of points. In CURE, the algorithm for assigning the appropriate cluster labels to the remaining data points employs a fraction of *randomly* selected representative points for each of the final k clusters. Each data point is assigned to the cluster containing the representative point closest to it.

Note that approximating every cluster with multiple points instead a single centroid as is done in [ZRL96], enables CURE to, in the final phase, correctly distribute the data points when clusters are non-spherical or non-uniform. The final labeling phase of [ZRL96], since it employs only the centroids of the clusters for partitioning the remaining points,

has a tendency to split clusters when they have non-spherical shapes or non-uniform sizes (since the space defined by a single centroid is a sphere).

4.4 Handling Outliers

Any data set almost always contains *outliers*. These do not belong to any of the clusters and are typically defined to be points of non agglomerative behavior. That is, the neighborhoods of outliers are generally sparse compared to points in clusters, and the distance of an outlier to the nearest cluster is comparatively higher than the distances among points in bonafide clusters themselves.

Every clustering method needs mechanisms to eliminate outliers. In CURE, outliers are dealt with at multiple steps. First, random sampling filters out a majority of the outliers. Furthermore, the few outliers that actually make it into the random sample are distributed all over the sample space. Thus, random sampling further isolates outliers.

In agglomerative hierarchical clustering, initially each point is a separate cluster. Clustering then proceeds by merging closest points first. What this suggests is that outliers, due to their larger distances from other points, tend to merge with other points less and typically grow at a much slower rate than actual clusters. Thus, the number of points in a collection of outliers is typically much less than the number in a cluster.

This leads us to a scheme of outlier elimination that proceeds in two phases. In the first phase, the clusters which are growing very slowly are identified and eliminated as outliers. This is achieved by proceeding with the clustering for some time until the number of clusters decreases below a certain fraction of the initial number of clusters. At this time, we classify clusters with very few points (e.g., 1 or 2) as outliers. The choice of a value for the fraction of initial clusters at which outlier elimination gets triggered is important. A very high value for the fraction could result in a number of cluster points being incorrectly eliminated – on the other hand, with an extremely low value, outliers may get merged into proper clusters before the elimination can kick in. An appropriate value for the fraction, obviously, is dependent on the data set. For most data sets we considered, a value of around $\frac{1}{3}$ performed well.

The first phase of outlier elimination is rendered ineffective if a number of outliers get sampled in close vicinity – this is possible in a randomized algorithm albeit with low probability. In this case, the outliers merge together preventing their elimination, and we require a second level of pruning. The second phase occurs toward the end. Usually, the last few steps of a clustering are the most important ones, because the granularity of the clusters is very high, and a single mistake could have grave consequences. Consequently, the second phase of outlier elimination is necessary for good clustering. From our earlier discussion, it is easy to observe that outliers form very small clusters. As a result, we can easily identify such small groups and eliminate them when there are very few clusters remaining, typically in the order of k, the actual number of clusters desired. We have found the above two-phase approach to outlier elimination to work very well in practice.

5 Experimental Results

In this section, we study the performance of CURE and demonstrate its effectiveness for clustering compared to BIRCH and MST[1] (*minimum spanning tree*). From our experimental results, we establish that

- BIRCH fails to identify clusters with non-spherical shapes (e.g., elongated) or wide variances in size.
- MST is better at clustering arbitrary shapes, but is very sensitive to outliers.
- CURE can discover clusters with interesting shapes and is less sensitive (than MST) to outliers.
- Sampling and partitioning, together, constitute an effective scheme for preclustering – they reduce the input size for large data sets without sacrificing the quality of clustering.
- The execution time of CURE is low in practice.
- Sampling and our outlier removal algorithm do a good job at filtering outliers from data sets.
- Our final labeling phase labels the data residing on disk correctly even when clusters are non-spherical.

In our experiments, in addition to CURE, we consider BIRCH and MST. We first show that, for data sets containing elongated or big and small clusters, both BIRCH and MST fail to detect the clusters while CURE discovers them with appropriate parameter settings. We then focus on analyzing the sensitivity of CURE to parameters like the shrink factor α and the random sample size s. Finally, we compare the execution times of BIRCH and CURE on a data set from [ZRL96], and present the results of our scale-up experiments with CURE. In all of our experiments, we use euclidean distance as the distance metric. We performed experiments using a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5.

Due to lack of space, we do not report all our experimental results – these can be found in [GRS97]. Also, the clusters in the figures in this section were generated using our labeling algorithm and visualizer. Our visualizer assigns a unique color to each cluster[2].

5.1 Algorithms

BIRCH: We used the implementation of BIRCH provided to us by the authors of [ZRL96]. The implementation performs preclustering and then uses a centroid-based hierarchical clustering algorithm with time and space complexity that is quadratic in the number of points after preclustering. We set parameter values to the default values suggested in [ZRL96]. For example, we set the page size to 1024 bytes and the input size to the hierarchical clustering algorithm after the preclustering phase to 1000. The memory used for preclustering was set to be about 5% of dataset size.

CURE: Our version of CURE is based on the clustering algorithm described in Section 3, that uses representative points with shrinking towards the mean. As described at the end of Section 3, when two clusters are merged in each step of the algorithm, representative points for the new merged cluster are selected from the ones for the two original clusters rather than all points in the merged cluster. This improvement speeds up execution times for CURE without adversely impacting the quality of clustering. In addition, we

[1] Clustering using MST is the same as the all-points approach described in Section 1, that is, hierarchical clustering using d_{min} as the distance measure.

[2] The figures in this paper were originally produced in color. These can be found in [GRS97].

Symbol	Meaning	Default Value	Range
s	Sample Size	2500	500 - 5000
c	Number of Representatives in Cluster	10	1 - 50
p	Number of Partitions	1	1 - 50
q	Reducing Factor for Each Partition	3	–
α	Shrink Factor	0.3	0 - 1.0

Table 1: Parameters

	Number of Points	Shape of Clusters	Number of Clusters
Data Set 1	100000	Big and Small Circles, Ellipsoids	5
Data Set 2	100000	Circles of Equal Sizes	100

Table 2: Data Sets

also employ random sampling, partitioning, outlier removal and labeling as described in Section 4. Our implementation makes use of the *k-d* tree and heap data structures. Thus, our implementation requires linear space and has a quadratic time complexity in the size of the input for lower dimensional data.

The partitioning constant q (introduced in Section 4.2) was set to 3. That is, we continue clustering in each partition until the number of clusters remaining is 1/3 of the number of points initially in the partition. We found that, for some of the data sets we considered, when q exceeds 3, points belonging to different clusters are merged, thus negatively impacting the clustering quality. Also we handle outliers as follows. First, at the end of partially clustering each partition, clusters containing only one point are considered outliers, and eliminated. Then, as the number of clusters approaches k, we again remove outliers – this time, the ones containing as many as 5 points. Finally, in almost all of our experiments, the random sample size was chosen to be about 2.5% of the initial data set size. Table 1 shows the parameters for our algorithm, along with their default values and the range of values for which we conducted experiments.

MST: When $\alpha = 0$ and the number of representative points c is a large number, CURE reduces to the MST method. Thus, instead of implementing the MST algorithm, we simply use CURE with the above parameter settings for α and c. This suffices for our purposes since we are primarily interested in performing qualitative measurements using MST.

5.2 Data sets

We experimented with four data sets containing points in two dimensions. Details of these experiments can be found in [GRS97]. Due to the lack of space, we report here the results with only two data sets whose geometric shape is as illustrated in Figure 7. The number of points in each data set is also described in Table 2. Data set 1 contains one big and two small circles that traditional partitional and hierarchical clustering algorithms, including BIRCH, fail to find. The data set also contains two ellipsoids which are connected by a chain of outliers. In addition, the data set has random outliers scattered in the entire space. The purpose of having outliers in the data set is to compare the sensitivity of CURE and MST to outliers. Data set 2 is identical to one of the data sets used for the experiments in [ZRL96]. It consists of 100 clusters with centers arranged in a grid pattern and data points in each cluster following a normal distribution with mean at the cluster center (a more detailed description of the data set can be found in [ZRL96]). We show that CURE not only correctly clusters this data set, but also that its execution times on the data set are much smaller than BIRCH.

5.3 Quality of Clustering

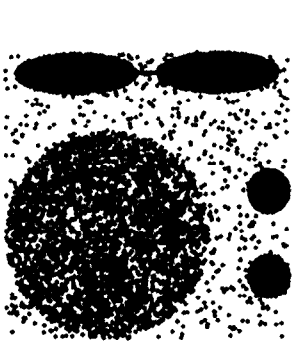
We run the three algorithms on Data set 1 to compare them with respect to the quality of clustering. Figure 8 shows the clusters found by the three algorithms for Data set 1. As expected, since BIRCH uses a centroid-based hierarchical clustering algorithm for clustering the preclustered points, it cannot distinguish between the big and small clusters. It splits the larger cluster while merging the two smaller clusters adjacent to it. In contrast, the MST algorithm merges the two ellipsoids because it cannot handle the chain of outliers connecting them. CURE successfully discovers the clusters in Data set 1 with the default parameter settings in Table 1, that is, $s = 2500$, $c = 10$, $\alpha = 0.3$ and $p = 1$. The moderate shrinking toward the mean by a factor of 0.3 enables CURE to be less sensitive to outliers without splitting the large and elongated clusters.

5.4 Sensitivity to Parameters

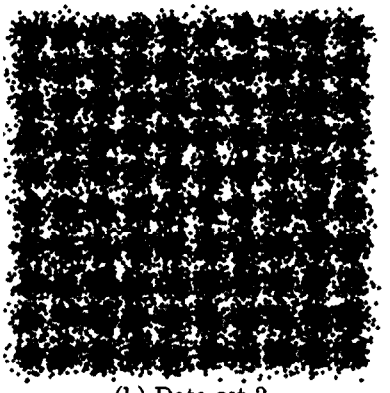
In this subsection, we perform a sensitivity analysis for CURE with respect to the parameters α, c, s and p. We use Data set 1 for our study. Furthermore, when a single parameter is varied, the default settings in Table 1 are used for the remaining parameters.

Shrink Factor α: Figure 9 shows the clusters found by CURE when α is varied from 0.1 to 0.9. The results, when $\alpha = 1$ and $\alpha = 0$, are similar to BIRCH and MST, respectively. These were presented in the previous subsection and thus, we do not present the results for these values of α. As the figures illustrate, when α is 0.1, the scattered points are shrunk very little and thus CURE degenerates to the MST algorithm which merges the two ellipsoids. CURE behaves similar to traditional centroid-based hierarchical algorithms for values of α between 0.8 and 1 since the representative points end up close to the center of the cluster. However, for the entire range of α values from 0.2 to 0.7, CURE always finds the right clusters. Thus, we can conclude that 0.2–0.7 is a good range of values for α to identify non-spherical clusters while dampening the effects of outliers.

Number of Representative Points c: We ran CURE while the number of representative points are varied from 1 to 100. For smaller values of c, we found that the quality of clustering suffered. For instance, when $c = 5$, the big cluster is split. This is because a small number of representative

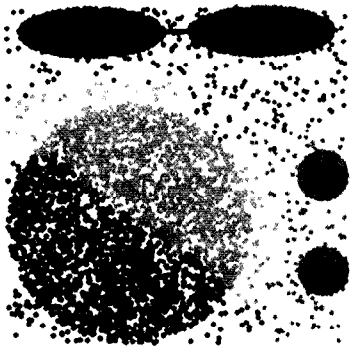


(a) Data set 1

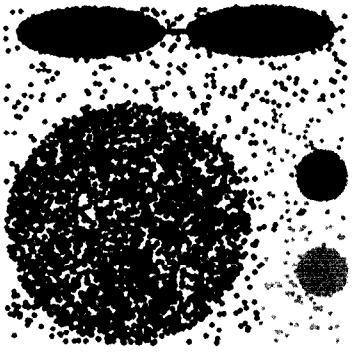


(b) Data set 2

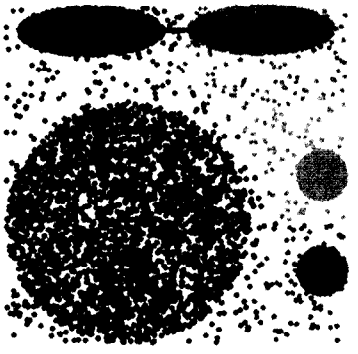
Figure 7: Data sets



(a) BIRCH



(b) MST METHOD



(c) CURE

Figure 8: Data set 1

points do not adequately capture the geometry of clusters. However, for values of c greater than 10, CURE always found right clusters. In order to illustrate the effectiveness of the representative points in capturing the geometry of clusters, we plot, in Figure 10, the representatives for clusters at the end of clustering. The circular dot is the center of clusters and the representative points are connected by lines. As the figure illustrates, the representative points take shapes of clusters. Also, Figure 10(b) explains why the big cluster is split when we choose only 5 representative points per cluster. The reason is that the distance between the closest representative points belonging to the two subclusters of the large cluster becomes fairly large when c is 5.

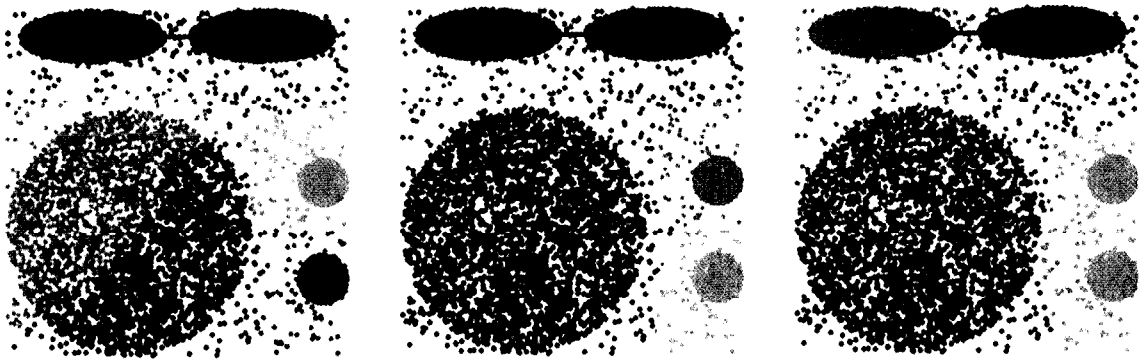
Number of Partitions p: We next varied the number of partitions from 1 to 100. With as many as 50 partitions, CURE always discovered the desired clusters. However, when we divide the sample into as many as 100 partitions, the quality of clustering suffers due to the fact that each partition does not contain enough points for each cluster. Thus, the relative distances between points in a cluster become large compared to the distances between points in different clusters – the result is that points belonging to different clusters are merged.

There is a correlation between the number of partitions p and the extent to which each partition is clustered as determined by q (a partition is clustered until the clusters remaining are $\frac{1}{q}$ of the original partition size). In order to preserve the integrity of clustering, as q increases, partition sizes must increase and thus, p must decrease. This is because with smaller partition sizes, clustering each partition to a larger degree could result in points being merged across clusters (since each partition contains fewer points from each cluster). In general, the number of partitions must be chosen such that $\frac{s}{pq}$ is fairly large compared to k (e.g., at least 2 or 3 times k).

Random Sample Size s: We ran CURE on Data set 1 with random sample sizes ranging from 500 upto 5000. For sample sizes up to 2000, the clusters found were of poor quality. However, from 2500 sample points and above (2.5% of the data set size), CURE always correctly identified the clusters.

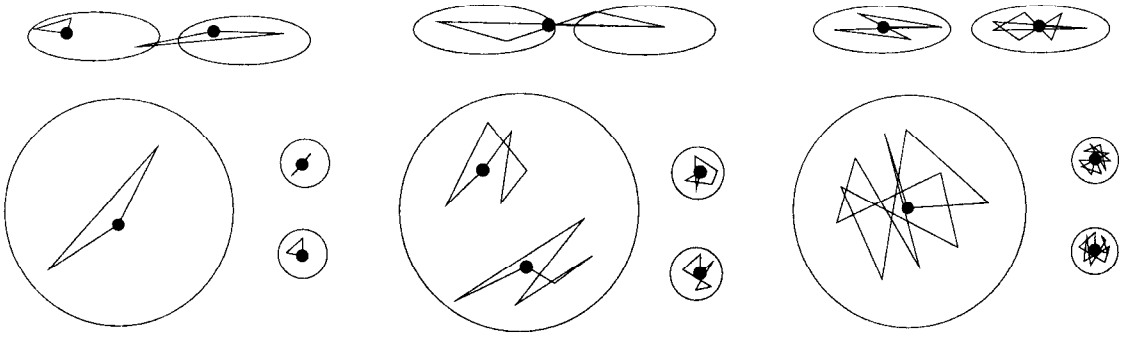
5.5 Comparison of Execution time to BIRCH

The goal of our experiment in this subsection is to demonstrate that using the combination of random sampling and partitioning to reduce the input size as is done by CURE can result in lower execution times than the preclustering scheme employed by BIRCH for the same purpose. We run both BIRCH and CURE on Data set 2 – this is the data set that centroid-based algorithms, in general, and BIRCH, in particular, can cluster correctly since it contains compact clusters with similar sizes. CURE, too, finds the right clusters with a random sample size of 2500, $\alpha = 1$ (which reduces CURE to a centroid-based algorithm), one representative for each cluster (that is, the centroid), and as many as 5 partitions.



(a) $\alpha = 0.1$ (b) $\alpha = 0.2 - 0.7$ (c) $\alpha = 0.8 - 0.9$

Figure 9: Shrink factor toward centroid



(a) $c = 2$ (b) $c = 5$ (c) $c = 10$

Figure 10: Representatives of clusters

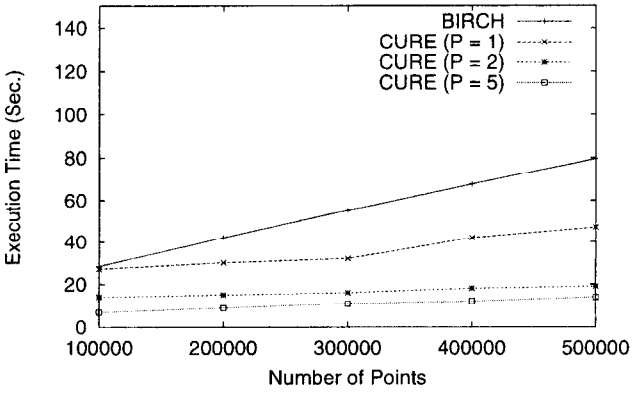


Figure 11: Comparison to BIRCH

Figure 11 illustrates the performance of BIRCH and CURE as the number of data points is increased from 100000 to 500000. The number of clusters or their geometry is not altered - thus, each cluster becomes more dense as the number of points increases. For CURE, we consider three values for the number of partitions: 1, 2 and 5, in order to show the effectiveness of partitioning. The execution times do not include the time for the final labeling phase since these are approximately the same for BIRCH and CURE, both of which utilize only the cluster centroid for the purpose of labeling.

As the graphs demonstrate, CURE's execution times are always lower than BIRCH's. In addition, partitioning further improves our running times by more than 50%. Finally, as the number of points is increased, execution times for CURE increase very little since the sample size stays at 2500, and the only additional cost incurred by CURE is that of sampling from a larger data set. In contrast, the executions times for BIRCH's preclustering algorithm increases much more rapidly with increasing data set size. This is because BIRCH scans the entire database and uses all the points in the data set for preclustering. Thus, the above results confirm that our proposed random sampling and partitioning algorithm are very efficient compared to the preclustering technique used in BIRCH.

5.6 Scale-up Experiments

The goal of the scale-up experiments is to determine the effects of the random sample size s, number of representatives c, number of partitions p and the number of dimensions on execution times. However, due to the lack of space, we only report our results when the random sample size is varied for Data set 1. The experimental results for varying number of representatives, partitions and dimensions can be found in [GRS97] (CURE took less than a minute to cluster points

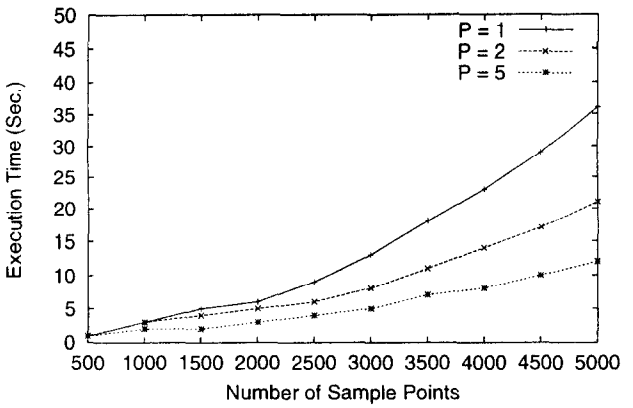


Figure 12: Scale-up experiments

with dimensionality as high as 40 – thus, CURE can comfortably handle high-dimensional data). In our running times, we do not include the time for the final labeling phase.

Random Sample Size: In Figure 12, we plot the execution time for CURE as the sample size is increased from 500 to 5000. The graphs confirm that the computational complexity of CURE is quadratic with respect to the sample size.

6 Concluding Remarks

In this paper, we addressed problems with traditional clustering algorithms which either favor clusters with spherical shapes and similar sizes, or are very fragile in the presence of outliers. We proposed a clustering method called CURE. CURE utilizes multiple representative points for each cluster that are generated by selecting well scattered points from the cluster and then shrinking them toward the center of the cluster by a specified fraction. This enables CURE to adjust well to the geometry of clusters having non-spherical shapes and wide variances in size. To handle large databases, CURE employs a combination of random sampling and partitioning that allows it to handle large data sets efficiently. Random sampling, coupled with outlier handling techniques, also makes it possible for CURE to filter outliers contained in the data set effectively. Furthermore, the labeling algorithm in CURE uses multiple random representative points for each cluster to assign data points on disk. This enables it to correctly label points even when the shapes of clusters are non-spherical and the sizes of clusters vary. For a random sample size of s, the time complexity of CURE is $O(s^2)$ for low-dimensional data and the space complexity is linear in s. To study the effectiveness of CURE for clustering large data sets, we conducted extensive experiments. Our results confirm that the quality of clusters produced by CURE is much better than those found by existing algorithms. Furthermore, they demonstrate that CURE not only outperforms existing algorithms but also scales well for large databases without sacrificing clustering quality.

Acknowledgments

We would like to thank Narain Gehani, Hank Korth, Rajeev Motwani and Avi Silberschatz for their encouragement and for their comments on earlier drafts of this paper. Without the support of Yesook Shim, it would have been impossible to complete this work.

References

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: an efficient and robust access method for points and rectangles. In *Proc. of ACM SIGMOD*, pages 322–331, Atlantic City, NJ, May 1990.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Massachusetts, 1990.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial database with noise. In *Int'l Conference on Knowledge Discovery in Databases and Data Mining (KDD-96)*, Portland, Oregon, August 1996.

[EKX95] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. A database interface for clustering in large spatial databases. In *Int'l Conference on Knowledge Discovery in Databases and Data Mining (KDD-95)*, Montreal, Canada, August 1995.

[GRS97] Sudipto Guha, R. Rastogi, and K. Shim. CURE: A clustering algorithm for large databases. Technical report, Bell Laboratories, Murray Hill, 1997.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of the VLDB Conference*, Santiago, Chile, September 1994.

[Ols93] Clark F. Olson. Parallel algorithms for hierarchical clustering. Technical report, University of California at Berkeley, December 1993.

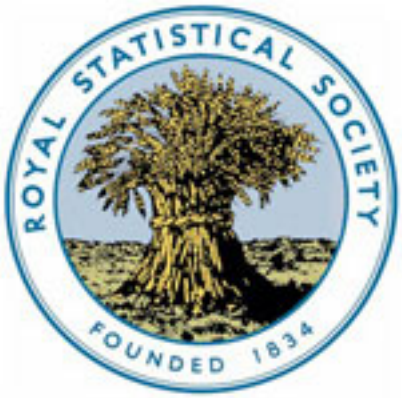
[Sam89] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1989.

[Sam90] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Publishing Company, Inc., New York, 1990.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+ tree: a dynamic index for multi-dimensional objects. In *Proc. 13th Int'l Conference on VLDB*, pages 507–518, England, 1987.

[Vit85] Jeff Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 103–114, Montreal, Canada, June 1996.



```
C        FIND MAXIMUM ENTRY
C
   60 PIVOT = ACU
      KK = 0
      DO 70 I = II, M
      K = INDEX(I)
      IF (ABS(LU(K, II)) .LE. PIVOT) GOTO 70
      PIVOT = ABS(LU(K, II))
      KK = I
   70 CONTINUE
      IF (KK .EQ. 0) GOTO 10
C
C        SWITCH ORDER
C
      ISAVE = INDEX(KK)
      INDEX(KK) = INDEX(II)
      INDEX(II) = ISAVE
C
C        PUT IN COLUMNS OF LU ONE AT A TIME
C
      IF (INTL) IBASE(II) = IROW
      IF (II .EQ. M) GOTO 90
      J = II + 1
      DO 80 I = J, M
      K = INDEX(I)
      LU(K, II) = LU(K, II) / LU(ISAVE, II)
   80 CONTINUE
   90 CONTINUE
      KKK = IROW
      RETURN
      END
```

Algorithm AS 136

A *K*-Means Clustering Algorithm

By J. A. HARTIGAN and M. A. WONG

Yale University, New Haven, Connecticut, U.S.A.

Keywords: *K*-MEANS CLUSTERING ALGORITHM; TRANSFER ALGORITHM

LANGUAGE

ISO Fortran

DESCRIPTION AND PURPOSE

The *K*-means clustering algorithm is described in detail by Hartigan (1975). An efficient version of the algorithm is presented here.

The aim of the *K*-means algorithm is to divide M points in N dimensions into K clusters so that the within-cluster sum of squares is minimized. It is not practical to require that the solution has minimal sum of squares against all partitions, except when M, N are small and $K = 2$. We seek instead "local" optima, solutions such that no movement of a point from one cluster to another will reduce the within-cluster sum of squares.

METHOD

The algorithm requires as input a matrix of M points in N dimensions and a matrix of K initial cluster centres in N dimensions. The number of points in cluster L is denoted by $NC(L)$. $D(I, L)$ is the Euclidean distance between point I and cluster L. The general procedure is to search for a K-partition with locally optimal within-cluster sum of squares by moving points from one cluster to another.

Step 1. For each point I ($I = 1, 2, \ldots, M$), find its closest and second closest cluster centres, $IC1(I)$ and $IC2(I)$ respectively. Assign point I to cluster $IC1(I)$.

Step 2. Update the cluster centres to be the averages of points contained within them.

Step 3. Initially, all clusters belong to the live set.

Step 4. This is the optimal-transfer (*OPTRA*) stage:
Consider each point I ($I = 1, 2, \ldots, M$) in turn. If cluster L ($L = 1, 2, \ldots, K$) is updated in the last quick-transfer (*QTRAN*) stage, then it belongs to the live set throughout this stage. Otherwise, at each step, it is not in the live set if it has not been updated in the last M optimal-transfer steps. Let point I be in cluster $L1$. If $L1$ is in the live set, do *Step* 4a; otherwise, do *Step* 4b.

Step 4a. Compute the minimum of the quantity, $R2 = [NC(L) * D(I,L)^2]/[NC(L)+1]$, over all clusters L ($L \neq L1, L = 1, 2, \ldots, K$). Let $L2$ be the cluster with the smallest $R2$. If this value is greater than or equal to $[NC(L1) * D(I,L1)^2]/[NC(L1)-1]$, no reallocation is necessary and $L2$ is the new $IC2(I)$. (Note that the value $[NC(L1) * D(I,L1)^2]/[NC(L1)-1]$ is remembered and will remain the same for point I until cluster $L1$ is updated.) Otherwise, point I is allocated to cluster $L2$ and $L1$ is the new $IC2(I)$. Cluster centres are updated to be the means of points assigned to them if reallocation has taken place. The two clusters that are involved in the transfer of point I at this particular step are now in the live set.

Step 4b. This step is the same as *Step* 4a, except that the minimum $R2$ is computed only over clusters in the live set.

Step 5. Stop if the live set is empty. Otherwise, go to *Step* 6 after one pass through the data set.

Step 6. This is the *quick-transfer* (*QTRAN*) stage:
Consider each point I ($I = 1, 2, \ldots, M$) in turn. Let $L1 = IC1(I)$ and $L2 = IC2(I)$. It is not necessary to check the point I if both the clusters $L1$ and $L2$ have not changed in the last M steps. Compute the values

$$R1 = [NC(L1) * D(I,L1)^2]/[NC(\mathrm{L1})-1] \quad \text{and} \quad R2 = [NC(L2) * D(I,L2)^2]/[NC(L2)+1].$$

(As noted earlier, $R1$ is remembered and will remain the same until cluster $L1$ is updated.) If $R1$ is less than $R2$, point I remains in cluster $L1$. Otherwise, switch $IC1(I)$ and $IC2(I)$ and update the centres of clusters $L1$ and $L2$. The two clusters are also noted for their involvement in a transfer at this step.

Step 7. If no transfer took place in the last M steps, go to *Step* 4. Otherwise, go to *Step* 6.

STRUCTURE

SUBROUTINE KMNS (*A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP, D, ITRAN, LIVE, ITER, WSS, IFAULT*)

Formal parameters

A	Real array (*M, N*)	input:	the data matrix
M	Integer	input:	the number of points
N	Integer	input:	the number of dimensions
C	Real array (*K, N*)	input:	the matrix of initial cluster centres
		output:	the matrix of final cluster centres
K	Integer	input:	the number of clusters
IC1	Integer array (*M*)	output:	the cluster each point belongs to
IC2	Integer array (*M*)	workspace:	this array is used to remember the cluster which each point is most likely to be transferred to at each step
NC	Integer array (*K*)	output:	the number of points in each cluster
AN1	Real array (*K*)	workspace:	
AN2	Real array (*K*)	workspace:	

NCP	Integer array (K)	workspace:	
D	Real array (M)	workspace:	
ITRAN	Integer array (K)	workspace:	
LIVE	Integer array (K)	workspace:	
ITER	Integer	input:	the maximum number of iterations allowed
WSS	Real array (K)	output:	the within-cluster sum of squares of each cluster
IFAULT	Integer	output:	see Fault Diagnostics below

Fault Diagnostics

$IFAULT = 0$	No fault
$IFAULT = 1$	At least one cluster is empty after the initial assignment. (A better set of initial cluster centres is called for)
$IFAULT = 2$	The allowed maximum number of iterations is exceeded
$IFAULT = 3$	K is less than or equal to 1 or greater than or equal to M

Auxiliary algorithms

The following auxiliary algorithms are called: *SUBROUTINE OPTRA* (*A, M, N, C, K, IC*1, *IC*2, *NC, AN*1, *AN*2, *NCP, D, ITRAN, LIVE, INDEX*) and *SUBROUTINE QTRAN* (*A, M, N, C, K, IC*1, *IC*2, *NC, AN*1, *AN*2, *NCP, D, ITRAN, INDEX*) which are included.

Related Algorithms

A related algorithm is AS 113 (A transfer algorithm for non-hierarchial classification) given by Banfield and Bassill (1977). This algorithm uses swops as well as transfers to try to overcome the problem of local optima; that is, for all pairs of points, a test is made whether exchanging the clusters to which the points belong will improve the criterion. It will be substantially more expensive than the present algorithm for large M.

The present algorithm is similar to Algorithm AS 58 (Euclidean cluster analysis) given by Sparks (1973). Both algorithms aim at finding a K-partition of the sample, with within-cluster sum of squares which cannot be reduced by moving points from one cluster to the other. However, the implementation of Algorithm AS 58 does not satisfy this condition. At the stage where each point is examined in turn to see if it should be reassigned to a different cluster, only the closest centre is used to check for possible reallocation of the given point; a cluster centre other than the closest one may have the smallest value of the quantity $\{n_l/(n_l+1)\}\, d_l^2$, where n_l is the number of points in cluster l and d_l is the distance from cluster l to the given point. Hence, in general, Algorithm AS 58 does not provide a locally optimal solution.

The two algorithms are tested on various generated data sets. The time consumed on the IBM 370/158 and the within-cluster sum of squares of the resulting K-partitions are given in Table 1. While comparing the entries of the table, note that AS 58 does not give locally optimal solutions and so should be expected to take less time. The *WSS* are different for the two algorithms because they arrive at different partitions of the sets of points. A saving of about 50 per cent in time occurs in *KMNS* due to using "live" sets and due to using a quick-transfer stage which reduces the number of optimal transfer iterations by a factor of 4. Thus, *KMNS* compared to *AS* 58 is locally optimal and takes less time, especially when the number of clusters is large.

Time and Accuracy

The time is approximately equal to $CMNKI$ where I is the number of iterations. For an IBM 370/158, $C = 2{\cdot}1 \times 10^{-5}$ sec. However, different data structures require quite different numbers of iterations; and a careful selection of initial cluster centres will also lead to a considerable saving in time.

Storage requirement: $M(N+3)+K(N+7)$.

TABLE 1

		Time (sec)	*WSS*
1. $M = 1000$, $N = 10$, $K = 10$ (random spherical normal)	AS 58	63·86	7056·71
	KMNS	36·66	7065·59
2. $M = 1000$, $N = 10$, $K = 10$ (two widely separated random normals)	AS 58	43·49	7779·70
	KMNS	19·11	7822·01
3. $M = 1000$, $N = 10$, $K = 50$ (random spherical normal)	AS 58	135·71	4543·82
	KMNS	76·00	4561·48
4. $M = 1000$, $N = 10$, $K = 50$ (two widely separated random normals)	AS 58	95·51	5131·04
	KMNS	57·96	5096·23
5. $M = 50$, $N = 2$, $K = 8$ (two widely separated random normals)	AS 58	0·17	21·03
	KMNS	0·18	21·03

Missing variate values cannot be handled by this algorithm.

The algorithm produces a clustering which is only locally optimal; the within-cluster sum of squares may not be decreased by transferring a point from one cluster to another, but different partitions may have the same or smaller within cluster sum of squares.

The number of iterations required to attain local optimality is usually less than 10.

ADDITIONAL COMMENTS

One way of obtaining the initial cluster centres is suggested here. The points are first ordered by their distances to the overall mean of the sample. Then, for cluster L ($L = 1, 2, \ldots, K$), the $\{1+(L-1)*[M/K]\}$th point is chosen to be its initial cluster centre. In effect, some K sample points are chosen as the initial cluster centres. Using this initialization process, it is guaranteed that no cluster will be empty after the initial assignment in the subroutine. A quick initialization, which is dependent on the input order of the points, takes the first K points as the initial centres.

ACKNOWLEDGEMENTS

This research is supported by National Science Foundation Grant MCS75–08374.

REFERENCES

BANFIELD, C. F. and BASSILL, L. C. (1977). Algorithm AS113. A transfer algorithm for non-hierarchical classification. *Appl. Statist.*, **26**, 206–210.

HARTIGAN, J. A. (1975). *Clustering Algorithms.* New York: Wiley.

SPARKS, D. N. (1973). Algorithm AS 58. Euclidean cluster analysis. *Appl. Statist.*, **22**, 126–130.

```
      SUBROUTINE KMNS(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP,
     *  D, ITRAN, LIVE, ITER, WSS, IFAULT)
C
C        ALGORITHM AS 136  APPL. STATIST. (1979) VOL.28, NO.1
C
C        DIVIDE M POINTS IN N-DIMENSIONAL SPACE INTO K CLUSTERS
C        SO THAT THE WITHIN CLUSTER SUM OF SQUARES IS MINIMIZED.
C
      DIMENSION A(M, N), IC1(M), IC2(M), D(M)
      DIMENSION C(K, N), NC(K), AN1(K), AN2(K), NCP(K)
      DIMENSION ITRAN(K), LIVE(K), WSS(K), DT(2)
C
C        DEFINE BIG TO BE A VERY LARGE POSITIVE NUMBER
C
      DATA BIG /1.0E10/
```

```
      IFAULT = 3
      IF (K .LE. 1 .OR. K .GE. M) RETURN
C
C        FOR EACH POINT I, FIND ITS TWO CLOSEST CENTRES,
C        IC1(I) AND IC2(I).  ASSIGN IT TO IC1(I).
C
      DO 50 I = 1, M
      IC1(I) = 1
      IC2(I) = 2
      DO 10 IL = 1, 2
      DT(IL) = 0.0
      DO 10 J = 1, N
      DA = A(I, J) - C(IL, J)
      DT(IL) = DT(IL) + DA * DA
   10 CONTINUE
      IF (DT(1) .LE. DT(2)) GOTO 20
      IC1(I) = 2
      IC2(I) = 1
      TEMP = DT(1)
      DT(1) = DT(2)
      DT(2) = TEMP
   20 DO 50 L = 3, K
      DB = 0.0
      DO 30 J = 1, N
      DC = A(I, J) - C(L, J)
      DB = DB + DC * DC
      IF (DB .GE. DT(2)) GOTO 50
   30 CONTINUE
      IF (DB .LT. DT(1)) GOTO 40
      DT(2) = DB
      IC2(I) = L
      GOTO 50
   40 DT(2) = DT(1)
      IC2(I) = IC1(I)
      DT(1) = DB
      IC1(I) = L
   50 CONTINUE
C
C        UPDATE CLUSTER CENTRES TO BE THE AVERAGE
C        OF POINTS CONTAINED WITHIN THEM
C
      DO 70 L = 1, K
      NC(L) = 0
      DO 60 J = 1, N
   60 C(L, J) = 0.0
   70 CONTINUE
      DO 90 I = 1, M
      L = IC1(I)
      NC(L) = NC(L) + 1
      DO 80 J = 1, N
   80 C(L, J) = C(L, J) + A(I, J)
   90 CONTINUE
C
C        CHECK TO SEE IF THERE IS ANY EMPTY CLUSTER AT THIS STAGE
C
      IFAULT = 1
      DO 100 L = 1, K
      IF (NC(L) .EQ. 0) RETURN
  100 CONTINUE
      IFAULT = 0
      DO 120 L = 1, K
      AA = NC(L)
      DO 110 J = 1, N
  110 C(L, J) = C(L, J) / AA
C
C        INITIALIZE AN1, AN2, ITRAN AND NCP
C        AN1(L) IS EQUAL TO NC(L) / (NC(L) - 1)
C        AN2(L) IS EQUAL TO NC(L) / (NC(L) + 1)
C        ITRAN(L)=1 IF CLUSTER L IS UPDATED IN THE QUICK-TRANSFER STAGE
C        ITRAN(L)=0 OTHERWISE
C        IN THE OPTIMAL-TRANSFER STAGE, NCP(L) INDICATES THE STEP AT
C        WHICH CLUSTER L IS LAST UPDATED
```

```
C        IN THE QUICK-TRANSFER STAGE, NCP(L) IS EQUAL TO THE STEP AT
C        WHICH CLUSTER L IS LAST UPDATED PLUS M
C
      AN2(L) = AA / (AA + 1.0)
      AN1(L) = BIG
      IF (AA .GT. 1.0) AN1(L) = AA / (AA - 1.0)
      ITRAN(L) = 1
      NCP(L) = -1
  120 CONTINUE
      INDEX = 0
      DO 140 IJ = 1, ITER
C
C        IN THIS STAGE, THERE IS ONLY ONE PASS THROUGH THE DATA.
C        EACH POINT IS REALLOCATED, IF NECESSARY, TO THE CLUSTER
C        THAT WILL INDUCE THE MAXIMUM REDUCTION IN WITHIN-CLUSTER
C        SUM OF SQUARES
C
      CALL OPTRA(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP,
     *  D, ITRAN, LIVE, INDEX)
C
C        STOP IF NO TRANSFER TOOK PLACE IN THE LAST M
C        OPTIMAL-TRANSFER STEPS
C
      IF (INDEX .EQ. M) GOTO 150
C
C        EACH POINT IS TESTED IN TURN TO SEE IF IT SHOULD BE
C        REALLOCATED TO THE CLUSTER WHICH IT IS MOST LIKELY TO
C        BE TRANSFERRED TO (IC2(I)) FROM ITS PRESENT CLUSTER (IC1(I)).
C        LOOP THROUGH THE DATA UNTIL NO FURTHER CHANGE IS TO TAKE PLACE
C
      CALL QTRAN(A, M, N, C, K, IC1, IC2, NC, AN1, AN2,
     *  NCP, D, ITRAN, INDEX)
C
C        IF THERE ARE ONLY TWO CLUSTERS,
C        NO NEED TO RE-ENTER OPTIMAL-TRANSFER STAGE
C
      IF (K .EQ. 2) GOTO 150
C
C        NCP HAS TO BE SET TO 0 BEFORE ENTERING OPTRA
C
      DO 130 L = 1, K
  130 NCP(L) = 0
  140 CONTINUE
C
C        SINCE THE SPECIFIED NUMBER OF ITERATIONS IS EXCEEDED
C        IFAULT IS SET TO BE EQUAL TO 2.
C        THIS MAY INDICATE UNFORESEEN LOOPING
C
      IFAULT = 2
C
C        COMPUTE WITHIN CLUSTER SUM OF SQUARES FOR EACH CLUSTER
C
  150 DO 160 L = 1, K
      WSS(L) = 0.0
      DO 160 J = 1, N
      C(L, J) = 0.0
  160 CONTINUE
      DO 170 I = 1, M
      II = IC1(I)
      DO 170 J = 1, N
      C(II, J) = C(II, J) + A(I, J)
  170 CONTINUE
      DO 190 J = 1, N
      DO 180 L = 1, K
  180 C(L, J) = C(L, J) / FLOAT(NC(L))
      DO 190 I = 1, M
      II = IC1(I)
      DA = A(I, J) - C(II, J)
      WSS(II) = WSS(II) + DA * DA
  190 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE OPTRA(A, M, N, C, K, IC1, IC2, NC, AN1,
     *  AN2, NCP, D, ITRAN, LIVE, INDEX)
C
C       ALGORITHM AS 136.1  APPL. STATIST. (1979) VOL.28, NO.1
C
C       THIS IS THE OPTIMAL-TRANSFER STAGE
C
C       EACH POINT IS REALLOCATED, IF NECESSARY, TO THE
C       CLUSTER THAT WILL INDUCE A MAXIMUM REDUCTION IN
C       THE WITHIN-CLUSTER SUM OF SQUARES
C
      DIMENSION A(M, N), IC1(M), IC2(M), D(M)
      DIMENSION C(K, N), NC(K), AN1(K), AN2(K), NCP(K)
      DIMENSION ITRAN(K), LIVE(K)
C
C       DEFINE BIG TO BE A VERY LARGE POSITIVE NUMBER
C
      DATA BIG /1.0E10/
C
C       IF CLUSTER L IS UPDATED IN THE LAST QUICK-TRANSFER STAGE,
C       IT BELONGS TO THE LIVE SET THROUGHOUT THIS STAGE.
C       OTHERWISE, AT EACH STEP, IT IS NOT IN THE LIVE SET IF IT
C       HAS NOT BEEN UPDATED IN THE LAST M OPTIMAL-TRANSFER STEPS
C
      DO 10 L = 1, K
      IF (ITRAN(L) .EQ. 1) LIVE(L) = M + 1
   10 CONTINUE
      DO 100 I = 1, M
      INDEX = INDEX + 1
      L1 = IC1(I)
      L2 = IC2(I)
      LL = L2
C
C       IF POINT I IS THE ONLY MEMBER OF CLUSTER L1, NO TRANSFER
C
      IF (NC(L1) .EQ. 1) GOTO 90
C
C       IF L1 HAS NOT YET BEEN UPDATED IN THIS STAGE
C       NO NEED TO RECOMPUTE D(I)
C
      IF (NCP(L1) .EQ. 0) GOTO 30
      DE = 0.0
      DO 20 J = 1, N
      DF = A(I, J) - C(L1, J)
      DE = DE + DF * DF
   20 CONTINUE
      D(I) = DE * AN1(L1)
C
C       FIND THE CLUSTER WITH MINIMUM R2
C
   30 DA = 0.0
      DO 40 J = 1, N
      DB = A(I, J) - C(L2, J)
      DA = DA + DB * DB
   40 CONTINUE
      R2 = DA * AN2(L2)
      DO 60 L = 1, K
C
C       IF I IS GREATER THAN OR EQUAL TO LIVE(L1), THEN L1 IS
C       NOT IN THE LIVE SET.  IF THIS IS TRUE, WE ONLY NEED TO
C       CONSIDER CLUSTERS THAT ARE IN THE LIVE SET FOR POSSIBLE
C       TRANSFER OF POINT I.  OTHERWISE, WE NEED TO CONSIDER
C       ALL POSSIBLE CLUSTERS
C
      IF (I .GE. LIVE(L1) .AND. I .GE. LIVE(L) .OR.
     *  L .EQ. L1 .OR. L .EQ. LL) GOTO 60
      RR = R2 / AN2(L)
      DC = 0.0
      DO 50 J = 1, N
      DD = A(I, J) - C(L, J)
      DC = DC + DD * DD
      IF (DC .GE. RR) GOTO 60
```

```
   50 CONTINUE
      R2 = DC * AN2(L)
      L2 = L
   60 CONTINUE
      IF (R2 .LT. D(I)) GOTO 70
C
C        IF NO TRANSFER IS NECESSARY, L2 IS THE NEW IC2(I)
C
      IC2(I) = L2
      GOTO 90
C
C        UPDATE CLUSTER CENTRES, LIVE, NCP, AN1 AND AN2
C        FOR CLUSTERS L1 AND L2, AND UPDATE IC1(I) AND IC2(I)
C
   70 INDEX = 0
      LIVE(L1) = M + I
      LIVE(L2) = M + I
      NCP(L1) = I
      NCP(L2) = I
      AL1 = NC(L1)
      ALW = AL1 - 1.0
      AL2 = NC(L2)
      ALT = AL2 + 1.0
      DO 80 J = 1, N
      C(L1, J) = (C(L1, J) * AL1 - A(I, J)) / ALW
      C(L2, J) = (C(L2, J) * AL2 + A(I, J)) / ALT
   80 CONTINUE
      NC(L1) = NC(L1) - 1
      NC(L2) = NC(L2) + 1
      AN2(L1) = ALW / AL1
      AN1(L1) = BIG
      IF (ALW .GT. 1.0) AN1(L1) = ALW / (ALW - 1.0)
      AN1(L2) = ALT / AL2
      AN2(L2) = ALT / (ALT + 1.0)
      IC1(I) = L2
      IC2(I) = L1
   90 CONTINUE
      IF (INDEX .EQ. M) RETURN
  100 CONTINUE
      DO 110 L = 1, K
C
C        ITRAN(L) IS SET TO ZERO BEFORE ENTERING QTRAN.
C        ALSO, LIVE(L) HAS TO BE DECREASED BY M BEFORE
C        RE-ENTERING OPTRA
C
      ITRAN(L) = 0
      LIVE(L) = LIVE(L) - M
  110 CONTINUE
      RETURN
      END
C
      SUBROUTINE QTRAN(A, M, N, C, K, IC1, IC2, NC, AN1,
     *  AN2, NCP, D, ITRAN, INDEX)
C
C        ALGORITHM AS 136.2  APPL. STATIST. (1979) VOL.28, NO.1
C
C        THIS IS THE QUICK TRANSFER STAGE.
C        IC1(I)  IS THE CLUSTER WHICH POINT I BELONGS TO.
C        IC2(I)  IS THE CLUSTER WHICH POINT I IS MOST
C        LIKELY TO BE TRANSFERRED TO.
C        FOR EACH POINT I, IC1(I) AND IC2(I) ARE SWITCHED, IF
C        NECESSARY, TO REDUCE WITHIN CLUSTER SUM OF SQUARES.
C        THE CLUSTER CENTRES ARE UPDATED AFTER EACH STEP
C
      DIMENSION A(M, N), IC1(M), IC2(M), D(M)
      DIMENSION C(K, N), NC(K), AN1(K), AN2(K), NCP(K), ITRAN(K)
C
C        DEFINE BIG TO BE A VERY LARGE POSITIVE NUMBER
C
      DATA BIG /1.0E10/
```

```
C        IN THE OPTIMAL-TRANSFER STAGE, NCP(L) INDICATES THE
C        STEP AT WHICH CLUSTER L IS LAST UPDATED
C        IN THE QUICK-TRANSFER STAGE, NCP(L) IS EQUAL TO THE
C        STEP AT WHICH CLUSTER L IS LAST UPDATED PLUS M
C
      ICOUN = 0
      ISTEP = 0
   10 DO 70 I = 1, M
      ICOUN = ICOUN + 1
      ISTEP = ISTEP + 1
      L1 = IC1(I)
      L2 = IC2(I)
C
C        IF POINT I IS THE ONLY MEMBER OF CLUSTER L1, NO TRANSFER
C
      IF (NC(L1) .EQ. 1) GOTO 60
C
C        IF ISTEP IS GREATER THAN NCP(L1), NO NEED TO RECOMPUTE
C        DISTANCE FROM POINT I TO CLUSTER L1
C        NOTE THAT IF CLUSTER L1 IS LAST UPDATED EXACTLY M STEPS
C        AGO WE STILL NEED TO COMPUTE THE DISTANCE FROM POINT I
C        TO CLUSTER L1
C
      IF (ISTEP .GT. NCP(L1)) GOTO 30
      DA = 0.0
      DO 20 J = 1, N
      DB = A(I, J) - C(L1, J)
      DA = DA + DB * DB
   20 CONTINUE
      D(I) = DA * AN1(L1)
C
C        IF ISTEP IS GREATER THAN OR EQUAL TO BOTH NCP(L1) AND
C        NCP(L2) THERE WILL BE NO TRANSFER OF POINT I AT THIS STEP
C
   30 IF (ISTEP .GE. NCP(L1) .AND. ISTEP .GE. NCP(L2)) GOTO 60
      R2 = D(I) / AN2(L2)
      DD = 0.0
      DO 40 J = 1, N
      DE = A(I, J) - C(L2, J)
      DD = DD + DE * DE
      IF (DD .GE. R2) GOTO 60
   40 CONTINUE
C
C        UPDATE CLUSTER CENTRES, NCP, NC, ITRAN, AN1 AND AN2
C        FOR CLUSTERS L1 AND L2.  ALSO, UPDATE IC1(I) AND IC2(I).
C        NOTE THAT IF ANY UPDATING OCCURS IN THIS STAGE,
C        INDEX IS SET BACK TO 0
C
      ICOUN = 0
      INDEX = 0
      ITRAN(L1) = 1
      ITRAN(L2) = 1
      NCP(L1) = ISTEP + M
      NCP(L2) = ISTEP + M
      AL1 = NC(L1)
      ALW = AL1 - 1.0
      AL2 = NC(L2)
      ALT = AL2 + 1.0
      DO 50 J = 1, N
      C(L1, J) = (C(L1, J) * AL1 - A(I, J)) / ALW
      C(L2, J) = (C(L2, J) * AL2 + A(I, J)) / ALT
   50 CONTINUE
      NC(L1) = NC(L1) - 1
      NC(L2) = NC(L2) + 1
      AN2(L1) = ALW / AL1
      AN1(L1) = BIG
      IF (ALW .GT. 1.0) AN1(L1) = ALW / (ALW - 1.0)
      AN1(L2) = ALT / AL2
      AN2(L2) = ALT / (ALT + 1.0)
      IC1(I) = L2
      IC2(I) = L1
C
C        IF NO REALLOCATION TOOK PLACE IN THE LAST M STEPS, RETURN
C
   60 IF (ICOUN .EQ. M) RETURN
   70 CONTINUE
      GOTO 10
      END
```

Algorithms for Hierarchical Clustering: An Overview, II

Fionn Murtagh (1) and Pedro Contreras (2)
(1) University of Huddersfield, UK
(2) Thinking Safe Limited, Egham, UK
Email: fmurtagh@acm.org

July 29, 2017

Abstract

We survey agglomerative hierarchical clustering algorithms and discuss efficient implementations that are available in R and other software environments. We look at hierarchical self-organizing maps, and mixture models. We review grid-based clustering, focusing on hierarchical density-based approaches. Finally we describe a recently developed very efficient (linear time) hierarchical clustering algorithm, which can also be viewed as a hierarchical grid-based algorithm. This review adds to the earlier version, Murtagh and Contreras (2012).

1 Introduction

Agglomerative hierarchical clustering has been the dominant approach to constructing embedded classification schemes. It is our aim to direct the reader's attention to practical algorithms and methods – both efficient (from the computational and storage points of view) and effective (from the application point of view). It is often helpful to distinguish between *method*, involving a compactness criterion and the target structure of a two-way tree representing the partial order on subsets of the power set, as opposed to an *implementation*, which relates to the detail of the algorithm used.

As with many other multivariate techniques, the objects to be classified have numerical measurements on a set of variables or attributes. Hence, the analysis is carried out on the rows of an array or matrix. If we do not have a matrix of numerical values to begin with, then it may be necessary to skilfully construct such a matrix. The objects, or rows of the matrix, can be viewed as vectors in a multidimensional space (the dimensionality of this space being the number of variables or columns). A geometric framework of this type is not the only one which can be used to formulate clustering algorithms. Suitable alternative forms of storage of a rectangular array of values are not inconsistent with viewing the

problem in geometric terms (and in matrix terms – for example, expressing the adjacency relations in a graph).

Surveys of clustering with coverage also of hierarchical clustering include Gordon (1981), March (1983), Jain and Dubes (1988), Gordon (1987), Mirkin (1996), Jain, Murty and Flynn (1999), and Xu and Wunsch (2005). Lerman (1981) and Janowitz (2010) present overarching reviews of clustering including use of lattices that generalize trees. The case for the central role of hierarchical clustering in information retrieval was made by van Rijsbergen (1979) and continued in the work of Willett and coworkers (Griffiths et al., 1984). Various mathematical views of hierarchy, all expressing symmetry in one way or another, are included in Murtagh (2017).

This paper is organized as follows. In section *Distance, Similarity and Their Use*, we look at the issue of normalization of data, prior to inducing a hierarchy on the data. In section *Motivation*, some historical remarks and motivation are provided for hierarchical agglomerative clustering. In section *Algorithms*, we discuss the Lance-Williams formulation of a wide range of algorithms, and how these algorithms can be expressed in graph theoretic terms and in geometric terms. In section *Efficient Hierarchical Clustering Algorithms Using Nearest Neighbor Chains*, quadratic computational time hierarchical clustering is described. This employs the reciprocal nearest neighbor (RNN) and nearest neighbor (NN) chain algorithm, to support building a hierarchical clustering in a more efficient manner compared to the Lance-Williams or basic geometric approaches. In section *Hierarchical Self-Organizing Maps and Hierarchical Mixture Modeling*, the main objective is visualization, and there is an overview of the hierarchical Kohonen self-organizing feature map, and also hierarchical model-based clustering. In concluding this section, there are some reflections on divisive hierarchical clustering, in general. Section *Density- and Grid-Based Clustering Techniques* surveys developments in grid- and density-based clustering. The following section, *Linear Time Grid Clustering Method: m-Adic Clustering*, describes a hierarchical clustering implementation in linear time, and therefore through direct reading of data. The can be of particular interest and benefit in distributed data based processing and, in general, for processing based on hierarchical clustering of massive data sets.

2 Distance, Similarity, and Their Use

Before clustering comes the phase of data measurement, or measurement of the observables. Let us look at some important considerations to be taken into account. These considerations relate to the metric or other spatial embedding, comprising the first phase of the data analysis *stricto sensu*.

To group data we need a way to measure the elements and their distances relative to each other in order to decide which elements belong to a group. This can be a similarity, although on many occasions a dissimilarity measurement, or a "stronger" distance, is used.

A distance between any pair of vectors or points i, j, k satisfies the properties

of: symmetry, $d(i,j) = d(j,i)$; positive definiteness, $d(i,j) > 0$ and $d(i,j) = 0$ iff $i = j$; and the triangular inequality, $d(i,j) \leq d(i,k) + d(k,j)$. If the triangular inequality is not taken into account, we have a dissimilarity. Finally a similarity is given by $s(i,j) = \max_{i,j}\{d(i,j)\} - d(i,j)$.

When working in a vector space, a traditional way to measure distances is a Minkowski distance, which is a family of metrics defined as follows:

$$L_p(\mathbf{x}_a, \mathbf{x}_b) = (\sum_{i=1}^{n} |\mathbf{x}_{i,a} - \mathbf{x}_{i,b}|^p)^{1/p};\ \forall\ p \geq 1,\ p \in \mathbb{Z}^+, \tag{1}$$

where $\mathbb{Z}^+$ is the set of positive integers.

The Manhattan, Euclidean and Chebyshev distances (the latter is also called maximum distance) are special cases of the Minkowski distance when $p = 1$, $p = 2$ and $p \to \infty$.

As an example of similarity we have the *cosine* similarity, which gives the angle between two vectors. This is widely used in text retrieval to match vector queries to the dataset. The smaller the angle between a query vector and a document vector, the closer a query is to a document. The normalized cosine similarity is defined as follows:

$$s(\mathbf{x}_a, \mathbf{x}_b) = \cos(\theta) = \frac{\mathbf{x}_a \cdot \mathbf{x}_b}{\|\mathbf{x}_a\|\|\mathbf{x}_b\|} \tag{2}$$

where $\mathbf{x}_a \cdot \mathbf{x}_b$ is the dot product and $\|\cdot\|$ the norm.

Other relevant distances are the Hellinger, variational, Mahalanobis and Hamming distances. Anderberg (1973) gives a good review of measurement and metrics, where their interrelationships are also discussed. Also Deza and Deza (2009) have produced a comprehensive list of distances in their *Encyclopedia of Distances*.

By mapping our input data into a Euclidean space, where each object is equiweighted, we can use a Euclidean distance for the clustering that follows. Correspondence analysis is very versatile in determining a Euclidean, factor space from a wide range of input data types, including frequency counts, mixed qualitative and quantitative data values, ranks or scores, and others. Further reading on this is to be found in Benzécri (1979), Le Roux and Rouanet (2004) and Murtagh (2005).

3 Agglomerative Hierarchical Clustering

3.1 Motivation

Motivation for clustering in general, covering hierarchical clustering and applications, includes the following: analysis of data; interactive user interfaces; storage and retrieval; and pattern recognition. One, quite basic, motivation for using hierarchical clustering is to have a large number of partitions. Each partition is associated with a level of the hierarchy, its dendrogram representation, or, as a mathematical graph theory term, a binary, rooted tree. One might be

initially motivated to carry out a computationally efficient partitioning, using k-means clustering, but without knowledge of how many clusters, k, should be relevant. A different motivation might be to structure one's data in a manner that would be relevant for interpretation as genealogy, or as a concept hierarchy or taxonomy. While there are certainly linkages between hierarchical clustering methodologies that will be described and overviewed below, such as linkage-based agglomerative criteria, the following, however, will not be at issue here. This is that divisive construction of a hierarchical clustering can sometimes be of primary interest. Such might be the case when the hierarchy is derived by repeated partitioning of a graph, perhaps representing a network (e.g., a network of social media contacts, or a telecommunications network, etc.). Some good examples are in Bader et al. (2013).

3.2 Introduction to Methods

Agglomerative hierarchical clustering algorithms can be characterized as *greedy*, in the algorithmic sense. A sequence of irreversible algorithm steps is used to construct the desired data structure. Assume that a pair of clusters, including possibly singletons, is merged or agglomerated at each step of the algorithm. Then the following are equivalent views of the same output structure constructed on n objects: a set of $n-1$ partitions, starting with the fine partition consisting of n classes and ending with the trivial partition consisting of just one class, the entire object set; a binary tree (one or two child nodes at each non-terminal node) commonly referred to as a dendrogram; a partially ordered set (poset) which is a subset of the power set of the n objects; and an ultrametric topology on the n objects.

An ultrametric, or tree metric, defines a stronger topology compared to, for example, a Euclidean metric geometry. For three points, i, j, k, metric and ultrametric respect the properties of symmetry (d, $d(i,j) = d(j,i)$) and positive definiteness ($d(i,j) > 0$ and if $d(i,j) = 0$ then $i = j$). An ultrametric satisfies the strong triangular or ultrametric (or non-Archimedean), inequality, $d(i,j) \leq \max\{d(i,k), d(k,j)\}$.

The single linkage hierarchical clustering approach outputs a set of clusters (to use graph theoretic terminology, a set of maximal connected subgraphs) at each level – or for each threshold value which produces a new partition. The single linkage method with which we begin is one of the oldest methods, its origins being traced to Polish researchers in the 1950s (Graham and Hell, 1985). The name *single linkage* arises since the interconnecting dissimilarity between two clusters or components is defined as the least interconnecting dissimilarity between a member of one and a member of the other. Other hierarchical clustering methods are characterized by other functions of the interconnecting linkage dissimilarities.

As early as the 1970s, it was held that about 75% of all published work on clustering employed hierarchical algorithms (Blashfield and Aldenderfer, 1978). Interpretation of the information contained in a dendrogram is often of one or

more of the following kinds: set inclusion relationships, partition of the object-sets, and significant clusters.

Much early work on hierarchical clustering was in the field of biological taxonomy, from the 1950s and more so from the 1960s onwards. The central reference in this area, the first edition of which dates from the early 1960s, is Sneath and Sokal (1973). One major interpretation of hierarchies has been the evolution relationships between the organisms under study. It is hoped, in this context, that a dendrogram provides a sufficiently accurate model of underlying evolutionary progression.

A common interpretation made of hierarchical clustering is to derive a partition. A further type of interpretation is instead to detect maximal (i.e. disjoint) clusters of interest at varying levels of the hierarchy. Such an approach is used by Rapoport and Fillenbaum (1972) in a clustering of colors based on semantic attributes. Lerman (1981) developed an approach for finding significant clusters at varying levels of a hierarchy, which has been widely applied. By developing a wavelet transform *on* a dendrogram (Murtagh, 2007), which amounts to a wavelet transform in the associated ultrametric topological space, the most important – in the sense of best approximating – clusters can be determined. Such an approach is a topological one (i.e., based on sets and their properties) as contrasted with more widely used optimization or statistical approaches.

In summary, a dendrogram collects together many of the proximity and classificatory relationships in a body of data. It is a convenient representation which answers such questions as: "How many useful groups are in this data?", "What are the salient interrelationships present?". But it can be noted that differing answers can feasibly be provided by a dendrogram for most of these questions, depending on the application.

3.3 Algorithms

A wide range of agglomerative hierarchical clustering algorithms have been proposed at one time or another. Such hierarchical algorithms may be conveniently broken down into two groups of methods. The first group is that of linkage methods – the single, complete, weighted and unweighted average linkage methods. These are methods for which a graph representation can be used. Sneath and Sokal (1973) may be consulted for many other graph representations of the stages in the construction of hierarchical clusterings.

The second group of hierarchical clustering methods are methods which allow the cluster centers to be specified (as an average or a weighted average of the member vectors of the cluster). These methods include the centroid, median and minimum variance methods.

The latter may be specified either in terms of dissimilarities, alone, or alternatively in terms of cluster center coordinates and dissimilarities. A very convenient formulation, in dissimilarity terms, which embraces all the hierarchical methods mentioned so far, is the *Lance-Williams dissimilarity update formula*. If points (objects) i and j are agglomerated into cluster $i \cup j$, then

we must simply specify the new dissimilarity between the cluster and all other points (objects or clusters). The formula is:

$$d(i \cup j, k) = \alpha_i d(i,k) + \alpha_j d(j,k) + \beta d(i,j) + \gamma \mid d(i,k) - d(j,k) \mid$$

where α_i, α_j, β, and γ define the agglomerative criterion. Values of these are listed in the second column of Table 1. In the case of the single link method, using $\alpha_i = \alpha_j = \frac{1}{2}$, $\beta = 0$, and $\gamma = -\frac{1}{2}$ gives us

$$d(i \cup j, k) = \frac{1}{2} d(i,k) + \frac{1}{2} d(j,k) - \frac{1}{2} \mid d(i,k) - d(j,k) \mid$$

which, it may be verified, can be rewritten as

$$d(i \cup j, k) = \min \{d(i,k), d(j,k)\}.$$

Using other update formulas, as given in column 2 of Table 1, allows the other agglomerative methods to be implemented in a very similar way to the implementation of the single link method.

In the case of the methods which use cluster centers, we have the center coordinates (in column 3 of Table 1) and dissimilarities as defined between cluster centers (column 4 of Table 1). The Euclidean distance must be used for equivalence between the two approaches. In the case of the *median method*, for instance, we have the following (cf. Table 1).

Let $\mathbf{a}$ and $\mathbf{b}$ be two points (i.e. m-dimensional vectors: these are objects or cluster centers) which have been agglomerated, and let $\mathbf{c}$ be another point. From the Lance-Williams dissimilarity update formula, using squared Euclidean distances, we have:

$$\begin{aligned} d^2(a \cup b, c) &= \frac{d^2(a,c)}{2} + \frac{d^2(b,c)}{2} - \frac{d^2(a,b)}{4} \\ &= \frac{\|\mathbf{a}-\mathbf{c}\|^2}{2} + \frac{\|\mathbf{b}-\mathbf{c}\|^2}{2} - \frac{\|\mathbf{a}-\mathbf{b}\|^2}{4}. \end{aligned} \tag{3}$$

The new cluster center is $(\mathbf{a} + \mathbf{b})/2$, so that its distance to point $\mathbf{c}$ is

$$\|\mathbf{c} - \frac{\mathbf{a} + \mathbf{b}}{2}\|^2. \tag{4}$$

That these two expressions are identical is readily verified. The correspondence between these two perspectives on the one agglomerative criterion is similarly proved for the centroid and minimum variance methods. This is an example of a "stored data" algorithm (see Box 1).

For cluster center methods, and with suitable alterations for graph methods, the following algorithm is an alternative to the general dissimilarity based algorithm. The latter may be described as a "stored dissimilarities approach" (Anderberg, 1973).

In steps 1 and 2, "point" refers either to objects or clusters, both of which are defined as vectors in the case of cluster center methods. This algorithm is justified by storage considerations, since we have $O(n)$ storage required for n initial

Hierarchical clustering methods (and aliases)	Lance and Williams dissimilarity update formula	Coordinates of center of cluster, which agglomerates clusters i and j	Dissimilarity between cluster centers g_i and g_j
Single link (nearest neighbor)	$\alpha_i = 0.5$ $\beta = 0$ $\gamma = -0.5$ (More simply: $min\{d_{ik}, d_{jk}\}$)		
Complete link (diameter)	$\alpha_i = 0.5$ $\beta = 0$ $\gamma = 0.5$ (More simply: $max\{d_{ik}, d_{jk}\}$)		
Group average (average link, UPGMA)	$\alpha_i = \frac{\mid i \mid}{\mid i \mid + \mid j \mid}$ $\beta = 0$ $\gamma = 0$		
McQuitty's method (WPGMA)	$\alpha_i = 0.5$ $\beta = 0$ $\gamma = 0$		
Median method (Gower's, WPGMC)	$\alpha_i = 0.5$ $\beta = -0.25$ $\gamma = 0$	$\mathbf{g} = \frac{\mathbf{g}_i + \mathbf{g}_j}{2}$	$\Vert \mathbf{g}_i - \mathbf{g}_j \Vert^2$
Centroid (UPGMC)	$\alpha_i = \frac{\mid i \mid}{\mid i \mid + \mid j \mid}$ $\beta = -\frac{\mid i \mid \mid j \mid}{(\mid i \mid + \mid j \mid)^2}$ $\gamma = 0$	$\mathbf{g} = \frac{\mid i \mid \mathbf{g}_i + \mid j \mid \mathbf{g}_j}{\mid i \mid + \mid j \mid}$	$\Vert \mathbf{g}_i - \mathbf{g}_j \Vert^2$
Ward's method (minimum variance, error sum of squares)	$\alpha_i = \frac{\mid i \mid + \mid k \mid}{\mid i \mid + \mid j \mid + \mid k \mid}$ $\beta = -\frac{\mid k \mid}{\mid i \mid + \mid j \mid + \mid k \mid}$ $\gamma = 0$	$\mathbf{g} = \frac{\mid i \mid \mathbf{g}_i + \mid j \mid \mathbf{g}_j}{\mid i \mid + \mid j \mid}$	$\frac{\mid i \mid \mid j \mid}{\mid i \mid + \mid j \mid} \Vert \mathbf{g}_i - \mathbf{g}_j \Vert^2$

Notes: $\mid i \mid$ is the number of objects in cluster i. $\mathbf{g}_i$ is a vector in m-space (m is the set of attributes), – either an initial point or a cluster center. $\Vert . \Vert$ is the norm in the Euclidean metric. The names UPGMA, etc. are due to Sneath and Sokal (1973). Coefficient α_j, with index j, is defined identically to coefficient α_i with index i. Finally, the Lance and Williams recurrence formula is (with $\mid . \mid$ expressing absolute value):

$$d_{i \cup j, k} = \alpha_i d_{ik} + \alpha_j d_{jk} + \beta d_{ij} + \gamma \mid d_{ik} - d_{jk} \mid .$$

Table 1: Specifications of seven hierarchical clustering methods.

objects and $O(n)$ storage for the $n-1$ (at most) clusters. In the case of linkage methods, the term "fragment" in step 2 refers (in the terminology of graph theory) 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. Without consideration of any special algorithmic "speed-ups", the overall complexity of the above algorithm is $O(n^3)$ due to the repeated calculation of dissimilarities in step 1, coupled with $O(n)$ iterations through steps 1, 2 and 3. While the stored data algorithm is instructive, it does not lend itself to efficient implementations. In the section to follow, we look at the reciprocal nearest neighbor and mutual nearest neighbor algorithms which can be used in practice for implementing agglomerative hierarchical clustering algorithms.

Before concluding this overview of agglomerative hierarchical clustering algorithms, we will describe briefly the minimum variance method.

The variance or spread of a set of points (i.e. the average of the sum of squared distances from the center) has been a point of departure for specifying clustering algorithms. Many of these algorithms, – iterative, optimization algorithms as well as the hierarchical, agglomerative algorithms – are described and appraised in Wishart (1969). The use of variance in a clustering criterion links the resulting clustering to other data-analytic techniques which involve a decomposition of variance, and make the minimum variance agglomerative strategy particularly suitable for synoptic clustering. Hierarchies are also more balanced with this agglomerative criterion, which is often of practical advantage.

The minimum variance method produces clusters which satisfy compactness and isolation criteria. These criteria are incorporated into the dissimilarity. We seek to agglomerate two clusters, c_1 and c_2, into cluster c such that the within-class variance of the partition thereby obtained is minimum. Alternatively, the between-class variance of the partition obtained is to be maximized. Let P and Q be the partitions prior to, and subsequent to, the agglomeration; let p_1, p_2, ... be classes of the partitions:

$$\begin{aligned} P &= \{p_1, p_2, \ldots, p_k, c_1, c_2\} \\ Q &= \{p_1, p_2, \ldots, p_k, c\}. \end{aligned}$$

Letting V denote *variance*, then in agglomerating two classes of P, the variance of the resulting partition (i.e. $V(Q)$) will necessarily decrease: therefore in seeking to minimize this decrease, we simultaneously achieve a partition with maximum between-class variance. The criterion to be optimized can then be shown to be:

$$\begin{aligned} V(P) - V(Q) &= V(c) - V(c_1) - V(c_2) \\ &= \frac{|c_1|\ |c_2|}{|c_1|+|c_2|} \|\mathbf{c_1} - \mathbf{c_2}\|^2 \ , \end{aligned}$$

which is the dissimilarity given in Table 1. This is a dissimilarity which may be determined for any pair of classes of partition P; and the agglomerands are those classes, c_1 and c_2, for which it is minimum.

It may be noted that if c_1 and c_2 are singleton classes, then $V(\{c_1, c_2\}) = \frac{1}{2}\|\mathbf{c}_1 - \mathbf{c}_2\|^2$, i.e. the variance of a pair of objects is equal to half their Euclidean distance.

Figure 1: Five points, showing nearest neighbors and reciprocal nearest neighbors.

4 Efficient Hierarchical Clustering Algorithms Using Nearest Neighbor Chains

In this section, the fundamentals are described that provide for quadratic computational time hierarchical clustering. Such implementation has long been used in the `hclust` hierarchical clustering in R.

Early, efficient algorithms for hierarchical clustering are due to Sibson (1973), Rohlf (1973) and Defays (1977). Their $O(n^2)$ implementations of the single link method and of a (non-unique) complete link method, respectively, have been widely cited.

In the early 1980s a range of significant improvements (de Rham, 1980; Juan, 1982) were made to the Lance-Williams, or related, dissimilarity update schema, which had been in wide use since the mid-1960s. Murtagh (1983, 1985) presents a survey of these algorithmic improvements. We will briefly describe them here. The new algorithms, which have the potential for *exactly* replicating results found in the classical but more computationally expensive way, are based on the construction of *nearest neighbor chains* and *reciprocal* or mutual NNs.

A NN-chain consists of an arbitrary point (a in Figure 1); followed by its NN (b in Figure 1); followed by the NN from among the remaining points (c, d, and e in Figure 1) of this second point; and so on until we necessarily have some pair of points which can be termed reciprocal or mutual NNs. (Such a pair of RNNs may be the first two points in the chain; and we have assumed that no two dissimilarities are equal.)

In constructing a NN-chain, irrespective of the starting point, we may agglomerate a pair of RNNs as soon as they are found. What guarantees that we can arrive at the same hierarchy as if we used traditional "stored dissimilarities" or "stored data" algorithms? Essentially this is the same condition as that under which no inversions or reversals are produced by the clustering method. Fig. 2 gives an example of this, where s is agglomerated at a lower criterion value (i.e. dissimilarity) than was the case at the previous agglomeration between q and r. Our ambient space has thus contracted because of the agglomeration. This is because of the algorithm used – in particular the agglomeration criterion – and it is something we would normally wish to avoid.

This is formulated as:

$$\text{Inversion impossible if: } d(i,j) < d(i,k) \text{ or } d(j,k)$$
$$\Rightarrow d(i,j) < d(i \cup j, k)$$

This is one form of Bruynooghe's *reducibility property* (Bruynooghe, 1977;

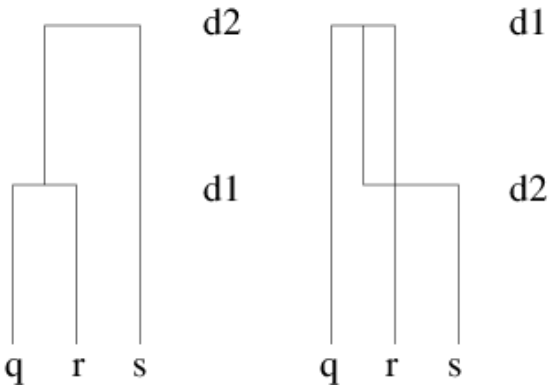


Figure 2: Alternative representations of a hierarchy with an inversion. Assuming dissimilarities, as we go vertically up, agglomerative criterion values (d_1, d_2) increase so that $d_2 > d_1$. But here, undesirably, $d_2 < d_1$ and the "cross-over" or inversion (right panel) arises.

Murtagh, 1984). Using the Lance-Williams dissimilarity update formula, it can be shown that the minimum variance method does not give rise to inversions; neither do the linkage methods; but the median and centroid methods cannot be guaranteed *not* to have inversions.

To return to Figure 1, if we are dealing with a clustering criterion which precludes inversions, then c and d can justifiably be agglomerated, since no other point (for example, b or e) could have been agglomerated to either of these.

The processing required, following an agglomeration, is to update the NNs of points such as b in Fig. 1 (and on account of such points, this algorithm was dubbed *algorithme des célibataires*, i.e. bachelors' algorithm, in de Rham, 1980). Box 2 gives a summary of the algorithm.

In Murtagh (1983, 1984, 1985) and Day and Edelsbrunner (1984), one finds discussions of $O(n^2)$ time and $O(n)$ space implementations of Ward's minimum variance (or error sum of squares) method and of the centroid and median methods. The latter two methods are termed the UPGMC and WPGMC criteria by Sneath and Sokal (1973). Now, a problem with the cluster criteria used by these latter two methods is that the reducibility property is not satisfied by them. This means that the hierarchy constructed may not be unique as a result of inversions or reversals (non-monotonic variation) in the clustering criterion value determined in the sequence of agglomerations.

Murtagh (1983, 1985) describes $O(n^2)$ time and $O(n^2)$ space implementations for the single link method, the complete link method and for the weighted and unweighted group average methods (WPGMA and UPGMA). This approach is quite general vis à vis the dissimilarity used and can also be used for hierarchical clustering methods other than those mentioned.

Day and Edelsbrunner (1984) prove the exact $O(n^2)$ time complexity of the

centroid and median methods using an argument related to the combinatorial problem of optimally packing hyperspheres into an m-dimensional volume. They also address the question of metrics: results are valid in a wide class of distances including those associated with the Minkowski metrics.

The construction and maintenance of the nearest neighbor chain as well as the carrying out of agglomerations whenever reciprocal nearest neighbors meet, both offer possibilities for distributed implementation. Implementations on a parallel machine architecture were described by Willett (1989).

Evidently (from Table 1) both coordinate data and graph (e.g., dissimilarity) data can be input to these agglomerative methods. Gillet et al. (1998) in the context of clustering chemical structure databases refer to the common use of the Ward method, based on the reciprocal nearest neighbors algorithm, on data sets of a few hundred thousand molecules.

Applications of hierarchical clustering to bibliographic information retrieval are assessed in Griffiths et al. (1984). Ward's minimum variance criterion is favored.

From details in White and McCain (1997), the Institute of Scientific Information (ISI) clusters citations (science, and social science) by first clustering highly cited documents based on a single linkage criterion, and then four more passes are made through the data to create a subset of a single linkage hierarchical clustering.

In the CLUSTAN and R statistical data analysis packages (in addition to hclust in R, see flashClust due to P. Langfelder and available on CRAN, "Comprehensive R Archive Network", cran.r-project.org) there are implementations of the NN-chain algorithm for the minimum variance agglomerative criterion. A property of the minimum variance agglomerative hierarchical clustering method is that we can use weights on the objects on which we will induce a hierarchy. By default, these weights are identical and equal to 1. Such weighting of observations to be clustered is an important and practical aspect of these software packages.

In Murtagh and Legendre (2014), there is extensive comparative study of the Ward, or minimum variance, agglomerative hierarchical clustering algorithm. Other than R's functions `hclust`, and `agnes`, the latter from package `cluster`, these software systems were included in the appraisal of how Ward's method was implemented: Matlab, SAS and JMP, SPSS, Statistica and Systat, what triggered this work was that the implementation in R had been using squared input distances. Now, in R, `hclust` fully addresses this modified implementation of the original Ward method.

Let us now consider contiguity constrained hierarchical clustering, where the rows are to be of fixed sequence. This can correspond to a chronological ordering of the rows. An example could be when the observations represented by the rows are linked to successive time steps. The proof of our ability to form a hierarchical clustering on such a fixed sequence of row vectors, using the complete link agglomerative criterion, is fundamentally described in: Bécue-Bertaut et al. (2014), Legendre and Legendre (2012), Murtagh (1985).

In these references it is shown that the algorithm now discussed is guaranteed

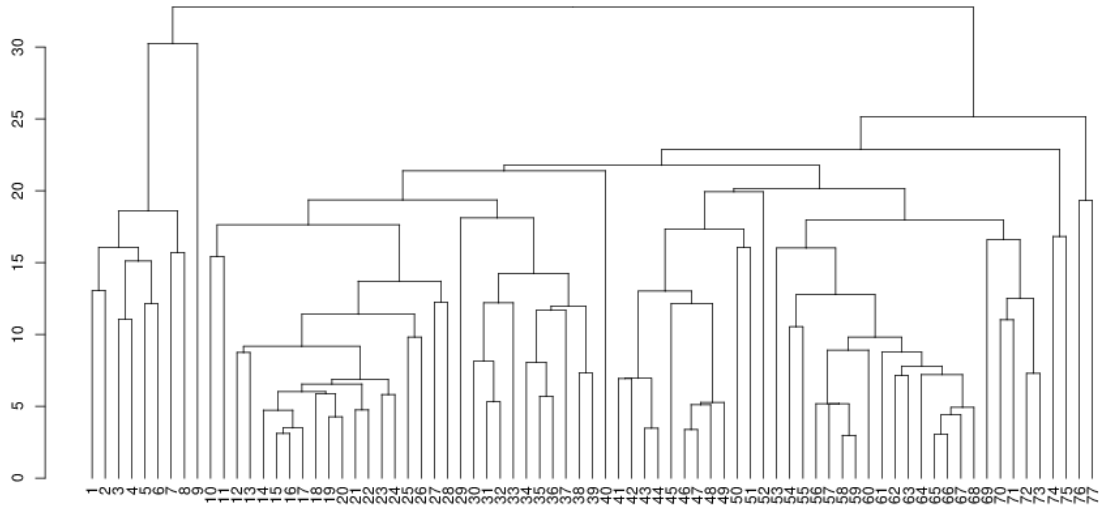


Figure 3: The 77 successive scenes of the Casablanca movie. It shows up scenes 9 to 10, and progressing from 39, to 40 and 41, as major changes.

to avoid inversions. With due consideration for the requirement that inversions not occur (Figure 2) it is possible to determine a hierarchical clustering that takes account of a timeline or other ordering on the input objects.

Such an algorithm is used to great effect in the semantic analysis of filmscript narrative: see Murtagh, Ganz and McKie (2009). This involves first carrying out a Correspondence Analysis, for data normalization purposes, furnishing a Euclidean embedding of our input data. Consider the projection of observation i onto the set of all factors indexed by α, $\{F_\alpha(i)\}$ for all α, which defines the observation i in the new coordinate frame. This new factor space is endowed with the (unweighted) Euclidean distance, d. We seek a hierarchical clustering that takes into account the observation sequence, i.e. observation i precedes observation i' for all $i, i' \in I$. We use the linear order on the observation set.

The agglomerative hierarchical clustering algorithm taking account of the ordered sequence of objects is shown in Box 3.

This is a sequence-constrained complete link agglomeration criterion. The cluster proximity at each agglomeration is strictly non-decreasing.

In the Casablanca movie, the film script is used. A corpus of words is determined from the terms used in the dialogues between movie personalities, with the names of the latter and some other useful metadata also included. Figure 3 represents close similarity, as well as major differences, between successive movie scenes.

Such chronologically constrained hierarchical clustering is beneficial for mapping out the contiguity and change in the semantics that are captured for our data. In Bécue-Bertaut et al. (2014) there is also the use of a permutation test to decide, statistically, when or if the sequence of agglomerations should be stopped. This is motivated by the agglomeration of clusters being demonstrated to be no longer justified by sufficient homogeneity. The latter, is based on a rejecting a null hypothesis of homogeneity, using a permutation test for this.

All in all, such chronological mapping can be an informative and useful mapping. It is of value and benefit for application to text data mining, as this case of filmscript analysis (or literary studies, or social media analytics such as Twitter, or such work, including prosecutor summing up speeches in court cases, or the US Supreme Court sessions).

5 Hierarchical Self-Organizing Maps and Hierarchical Mixture Modeling

While clustering as such is the key theme of all work under discussion here, this can sometimes be supported by visualization. What follows in this section will be the use of clustering outcomes for the visual user interface that that the computer user will avail of. First, however, let us point out the revealing potential of the following: first there are dendrograms displayed of the set of rows, and of the set of columns. These dendrograms are positioned relative to the rows of the data matrix that is under investigation, and related to its columns. That entails a reordering of the row set and of the column set. Mathematically, this is defined as follows: used are the row permutation and the column permutation that are (i) the ordering of the terminal nodes, or singleton clusters, in the dendrogram; this is respectively the case for each of the two dendrograms. Then, (ii), we have the given data array with row reordering and with column reordering, such that all is consistent with the ultrametric distances. The ultrametric distances are the distances between row vectors, that are derived from the hierarchy (informally an ultrametric distances can be expressed as "the lowest common ancestor distance"). Analogously this holds also for the ultrametric distances determined for the columns. Added to this, in Zhang et al. (2017), with all R code used, is a heatmap coloring of the data matrix. So the color dispalyed data matrix has the dendrograms displayed on the set of rows, and on the set of columns.

Visual metaphors have always influenced hierarchical clustering. A family tree, for example, is a natural enough way to introduce informally the notion of a metric on a tree, i.e. an ultrametric. A mathematical graph gives rise to a useful visual metaphor (cf. how typically a transport system is displayed) and the early tome on clustering including discussion on hierarchical clustering, Sneath and Sokal (1973), is amply illustrated using graphs. In this section we look at how spatial display has been used for hierarchical clustering. This combines in an intuitive way both visualization and data analysis.

It is quite impressive how 2D (2-dimensional or, for that matter, 3D) image signals can handle with ease the scalability limitations of clustering and many other data processing operations. The contiguity imposed on adjacent pixels or grid cells bypasses the need for nearest neighbor finding. It is very interesting therefore to consider the feasibility of taking problems of clustering massive data sets into the 2D image domain. The Kohonen self-organizing feature map exemplifes this well. In its basic variant (Kohonen, 1984, 2001) is can be formulated in terms of k-means clustering subject to a set of interrelationships between the cluster centers (Murtagh and Fernández-Pajares, 1995).

Kohonen maps lend themselves well for hierarchical representation. Lampinen and Oja (1992), Dittenbach et al. (2002) and Endo et al. (2002) elaborate on the Kohonen map in this way. An example application in character recognition is Miikkulanien (1990).

A short, informative review of hierarchical self-organizing maps is provided by Vicente and Vellido (2004). These authors also review what they term as probabilistic hierarchical models. This includes putting into a hierarchical framework the following: Gaussian mixture models, and a probabilistic – Bayesian – alternative to the Kohonen self-organizing map termed Generative Topographic Mapping (GTM).

GTM can be traced to the Kohonen self-organizing map in the following way. Firstly, we consider the hierarchical map as brought about through a growing process, i.e. the target map is allowed to grow in terms of layers, and of grid points within those layers. Secondly, we impose an explicit probability density model on the data. Tino and Nabney (2002) discuss how the local hierarchical models are organized in a hierarchical way.

In Wang et al. (2000) an alternating Gaussian mixture modeling, and principal component analysis, is described, in this way furnishing a hierarchy of model-based clusters. AIC, the Akaike information criterion, is used for selection of the best cluster model overall.

Murtagh et al. (2005) use a top level Gaussian mixture modeling with the (spatially aware) PLIC, pseudo-likelihood information criterion, used for cluster selection and identifiability. Then at the next level – and potentially also for further divisive, hierarchical levels – the Gaussian mixture modeling is continued but now using the marginal distributions within each cluster, and using the analogous Bayesian clustering identifiability criterion which is the Bayesian information criterion, BIC. The resulting output is referred to as a model-based cluster tree.

The model-based cluster tree algorithm of Murtagh et al. (2005) is a divisive hierarchical algorithm. Earlier in this article, we considered agglomerative algorithms. However it is often feasible to implement a divisive algorithm instead, especially when a graph cut (for example) is important for the application concerned. Mirkin (1996, chapter 7) describes divisive Ward, minimum variance hierarchical clustering, which is closely related to a bisecting k-means also.

A class of methods under the name of spectral clustering uses eigenvalue/eigenvector reduction on the (graph) adjacency matrix. As von Luxburg (2007) points out in reviewing this field of spectral clustering, such methods have "been discov-

ered, re-discovered, and extended many times in different communities". Far from seeing this great deal of work on clustering in any sense in a pessimistic way, we see the perennial and pervasive interest in clustering as testifying to the continual renewal and innovation in algorithm developments, faced with application needs.

It is indeed interesting to note how the clusters in a hierarchical clustering may be *defined* by the eigenvectors of a dissimilarity matrix, but subject to carrying out the eigenvector reduction in a particular algebraic structure, a semiring with additive and multiplicative operations given by "min" and "max", respectively (Gondran, 1976).

In section *Density- and Grid-Based Clustering Techniques*, the themes of mapping, and of divisive algorithm, are frequently taken in a somewhat different direction. As always, the application at issue is highly relevant for the choice of the hierarchical clustering algorithm.

6 Grid- and Density-Based Clustering Techniques

Many modern clustering techniques focus on large data sets. In Xu and Wunsch (2008, p. 215) these are classified as follows:

- Random sampling
- Data condensation
- Grid-based approaches
- Density-based approaches
- Divide and conquer
- Incremental learning

From the point of view of this article, we select density and grid based approaches, i.e., methods that either look for data densities or split the data space into cells when looking for groups. In this section we take a look at these two families of methods.

The main idea is to use a grid-like structure to split the information space, separating the dense grid regions from the less dense ones to form groups.

In general, a typical approach within this category will consist of the following steps as presented by Grabusts and Borisov (2002):

1. Creating a grid structure, i.e. partitioning the data space into a finite number of non-overlapping cells.
2. Calculating the cell density for each cell.
3. Sorting of the cells according to their densities.
4. Identifying cluster centers.

5. Traversal of neighbor cells.

Some of the more important algorithms within this category are the following:

- **STING:** STatistical INformation Grid-based clustering was proposed by Wang et al. (1997) who divide the spatial area into rectangular cells represented by a hierarchical structure. The root is at hierarchical level 1, its children at level 2, and so on. This algorithm has a computational complexity of $O(K)$, where K is the number of cells in the bottom layer. This implies that scaling this method to higher dimensional spaces is difficult (Hinneburg and Keim, 1999). For example, if in high dimensional data space each cell has four children, then the number of cells in the second level will be 2^m, where m is the dimensionality of the database.

- **OptiGrid:** Optimal Grid-Clustering was introduced by Hinneburg and Keim (1999) as an efficient algorithm to cluster high-dimensional databases with noise. It uses data partitioning based on divisive recursion by multidimensional grids, focusing on separation of clusters by hyperplanes. A cutting plane is chosen which goes through the point of minimal density, therefore splitting two dense half-spaces. This process is applied recursively with each subset of data. This algorithm is hierarchical, with time complexity of $O(n \cdot m)$ (Gan et al., 2007, pp. 210–212).

- **GRIDCLUS:** proposed by Schikuta (1996) is a hierarchical algorithm for clustering very large datasets. It uses a multidimensional data grid to organize the space surrounding the data values rather than organize the data themselves. Thereafter patterns are organized into blocks, which in turn are clustered by a topological neighbor search algorithm. Five main steps are involved in the GRIDCLUS method: (a) insertion of points into the grid structure, (b) calculation of density indices, (c) sorting the blocks with respect to their density indices, (d) identification of cluster centers, and (e) traversal of neighbor blocks.

- **WaveCluster:** this clustering technique proposed by Sheikholeslami et al. (2000) defines a uniform two dimensional grid on the data and represents the data points in each cell by the number of points. Thus the data points become a set of grey-scale points, which is treated as an image. Then the problem of looking for clusters is transformed into an image segmentation problem, where wavelets are used to take advantage of their multi-scaling and noise reduction properties. The basic algorithm is as follows: (a) create a data grid and assign each data object to a cell in the grid, (b) apply the wavelet transform to the data, (c) use the average sub-image to find connected clusters (i.e. connected pixels), and (d) map the resulting clusters back to the points in the original space. There is a great deal of other work also that is based on using the wavelet and other multiresolution transforms for segmentation.

- **CLIQUE:** introduced by Agarwal et al. (1998) identifies dense units in subspaces of high dimensional data. This algorithm can be considered both, density and grid based. In a broad sense it works as follows: It partitions each dimension into the same number of equal length interval; Then each m-dimensional data space into non-overlapping rectangular units; A unit is considered dense if the fraction of total data points contained within it exceeds the model parameter. Finally, a cluster is a maximal set of connected dense units within a subspace.

Further grid-based clustering algorithms can be found in the following: Chang and Jin (2002), Park and Lee (2004), Gan et al. (2007), and Xu and Wunsch (2008).

Density-based clustering algorithms are defined as dense regions of points, which are separated by low-density regions. Therefore, clusters can have an arbitrary shape and the points in the clusters may be arbitrarily distributed. An important advantage of this methodology is that only one scan of the dataset is needed and it can handle noise effectively. Furthermore the number of clusters to initialize the algorithm is not required.

Some of the more important algorithms in this category include the following:

- **DBSCAN:** Density-Based Spatial Clustering of Applications with Noise was proposed by Ester et al. (1996) to discover arbitrarily shaped clusters. Since it finds clusters based on density it does not need to know the number of clusters at initialization time. This algorithm has been widely used and has many variations (e.g., see GDBSCAN by Sander et al. (1998), PDBSCAN by Xu et al. (1999), and DBCluC by Zaïane and Lee (2002).

- **BRIDGE:** proposed by Dash et al. (2001) uses a hybrid approach integrating k-means to partition the dataset into k clusters, and then density-based algorithm DBSCAN is applied to each partition to find dense clusters.

- **DBCLASD:** Distribution-Based Clustering of LArge Spatial Databases (see Xu et al., 1998) assumes that data points within a cluster are uniformly distributed. The cluster produced is defined in terms of the nearest neighbor distance.

- **DENCLUE:** DENsity based CLUstering aims to cluster large multimedia data. It can find arbitrarily shaped clusters and at the same time deals with noise in the data. This algorithm has two steps. First a pre-cluster map is generated, and the data is divided in hypercubes where only the populated are considered. The second step takes the highly populated cubes and cubes that are connected to a highly populated cube to produce the clusters. For a detailed presentation of these steps see Hinneburg and Keim (1998).

- **CUBN:** this has three steps. First an erosion operation is carried out to find border points. Second, the nearest neighbor method is used to cluster

the border points. Finally, the nearest neighbor method is used to cluster the inner points. This algorithm is capable of finding non-spherical shapes and wide variations in size. Its computational complexity is $O(n)$ with n being the size of the dataset. For a detailed presentation of this algorithm see Wang and Wang (2003).

- **OPTICS:** Ordering Points To Identify the Clustering Structure algorithm first proposed by Ankerst et al. (1999). This is similar to DBSCAN but addresses one of its weaknesses, i.e. detection of meaningful clusters in density varying data.

7 Linear Time Grid Clustering Method: m-Adic Clustering

Algorithms described earlier in this paper, in sections *Efficient Hierarchical Clustering Algorithms Using Nearest Neighbor Chains* and *Agglomerative Hierarchical Clustering*, are of quadratic computational time. Alternatively expressed, they are of computational complexity $O(n^2)$ for n observations. This becomes quite impractical for data sets of any realistic size. In this section we describe a recent development that allows a hierarchical clustering to be constructed in (worst case) linear time.

In the previous section, we have seen a number of clustering methods that split the data space into cells, cubes, or dense regions to locate high density areas that can be further studied to find clusters.

For large data sets clustering via an m-adic (m integer, which if a prime is usually denoted as p) expansion is possible, with the advantage of doing so in linear time for the clustering algorithm based on this expansion. The usual base 10 system for numbers is none other than the case of $m = 10$ and the base 2 or binary system can be referred to as 2-adic where $p = 2$. Let us consider the following distance relating to the case of vectors x and y with 1 attribute, hence unidimensional:

$$d_{\mathcal{B}}(x, y) = \begin{cases} 1 & \text{if } x_1 \neq y_1 \\ \inf\ m^{-k} & \quad x_k = y_k \quad 1 \leq k \leq |K| \end{cases} \tag{5}$$

This distance defines the longest common prefix of strings. A space of strings, with this distance, is a Baire space. Thus we call this the Baire distance: here the longer the common prefix, the closer a pair of sequences. What is of interest to us here is this longest common prefix metric, which is an ultrametric (Murtagh et al., 2008).

For example, let us consider two such values, x and y. We take x and y to be bounded by 0 and 1. Each are of some precision, and we take the integer $|K|$ to be the maximum precision.

Thus we consider ordered sets x_k and y_k for $k \in K$. So, $k = 1$ is the index of the first decimal place of precision; $k = 2$ is the index of the second decimal

place; . . . ; $k = |K|$ is the index of the $|K|$th decimal place. The cardinality of the set K is the precision with which a number, x, is measured.

Consider as examples $x_k = 0.478$; and $y_k = 0.472$. In these cases, $|K| = 3$. Start from the first decimal position. For $k = 1$, we have $x_k = y_k = 4$. For $k = 2$, $x_k = y_k$. But for $k = 3$, $x_k \neq y_k$. Hence their Baire distance is 10^{-2} for base $m = 10$.

It is seen that this distance splits a unidimensional string of decimal values into a 10-way hierarchy, in which each leaf can be seen as a grid cell. From equation (5) we can read off the distance between points assigned to the same grid cell. All pairwise distances of points assigned to the same cell are the same.

Clustering using this Baire distance has been applied to large data sets in areas such as chemoinformatics (Murtagh et al., 2008), astronomy and text retrieval. A key element for implementation in high dimensions is the use of random projection. Such can be regarded as a generalization of principal component and factor space mappings. Both motivation and justification for all such analytical processing is that, as dimensionality increases greatly, then the data space is increasingly and inherently hierarchically structured. The 1978 Nobel Prize Winner in Economics, Herbert Simon stated also that the more complex any system or context is, then the more inherently hierarchical that it is structured (Murtagh, 2017).

8 Evaluation of Cluster Quality

We have seen a number of methods that allow for creating clusters. A natural question that arises is how to evaluate the clusters produced. Several validity criteria have been developed in the literature. They are mainly classified as external, internal or relative criteria (Jain and Dubes, 1988). In the external approach, groups assembled by a clustering algorithm are compared to a previously accepted partition on the testing dataset. In the internal approach, clustering validity is evaluated using data and features contained in the dataset. The relative approach searches for the best clustering result from an algorithm and compares it with a series of predefined clustering schemes. In all cases, validity indices are constructed to evaluate proximity among objects in a cluster or proximity among resulting clusters. For further information see Jain and Dubes (1988) where chapter 4 is dedicated to cluster validity, Gan et al. (2007) chapter 17, and Xu and Wunsch (2008) chapter 10. For relevant papers in this area see Halkidi et al. (2001, 2002), Dash et al. (2003), Bolshakova and Azuaje (2003). Another interesting study is Albatineh et al. (2006) where 22 indices are compared, and when adjusted for chance agreement it can be shown that many indices are similar. Also see Vinh et al. (2009) for additional information regarding correction for chance agreement.

A taxonomy of the cluster validity indices is in a figure in Gan et al. (2007), page 300. These indices can be separated into statistical and non-statistical methods. The statistical indices include the external and internal criteria, and the non-statistical, the relative criteria.

Table 2 (see Jain and Dubes, 1988, and Gan et al. 2007) shows some of the equations for cluster validation indices. Let P be a pre-specified partition of dataset X with n data points, and let C be a clustering partition from a clustering algorithm independent of P. Then by comparing C and P we obtain the evaluation of C by external criteria. Considering a pair of points x_i and x_j of X, there are four cases how x_i and x_j can be placed in C and P. We consider the following. **Case *a***: is the number of pairs of data points which are in the same clusters of C and P; **Case *b***: is the number of pairs of data points which are in the same clusters of C, but different clusters P; **Case *c***: is the number of pairs of data points which are in different clusters of C, but the same clusters P; and **case *d***, is where the number of pairs of data points which are in different clusters of C, and different clusters of P. Finally, let M be the total number of pairs of data points in the dataset, then $M = a + b + c + d = \frac{n(n-1)}{2}$.

Index name	Formula
Rand statistics	$R = \frac{a+d}{M}$
Jaccard coefficient	$J = \frac{a}{a+b+c}$
Folkes and Mallows index	$FM = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}$
Hubert's Γ statistics	$\Gamma = \frac{M_a - m_1 m_2}{\sqrt{m_1 m_2 (M-m_1)(M-m_2)}}$

Table 2: Some external criteria indices to measure the degree of similarity between clusters. where $m_1 = (a + b)$ and $m_2 = (b + c)$; R, J, FM, and $\Gamma \in [0, 1]$.

9 Conclusions

The fields of application of hierarchical clustering are all-pervasive. This follows from the clustering objectives that range over partitioning and also the structuring of the clusters that can be genealogical or taxonomic.

The following domains of application may be noted in the references: Biosciences, Bolshakova and Azuaje (2003); Text mining, Griffiths et al. (1984), Bécue-Bertaut et al. (2014), Murtagh et al. (2009), White and McCain (1997); Information retrieval, van Rijsbergen (1979); Spatial and geo-sciences, Ester et al. (1996), Sander et al. (1998), Sheikholeslami et al. (2000), Wang et al. (1997), Xu et al. (1998, 1999); Chemistry, Gillet et al. (1998); Ecology, Legendre and Legendre (2012); Clinical and medical science, Zhang et al. (2017); Data Science in general, Murtagh (2017) that also includes astronomy and psychoanalysis.

Hierarchical clustering methods, with roots going back to the 1960s and 1970s, are continually replenished with new challenges. As a family of algorithms they are central to the addressing of many important problems. Their

deployment in many application domains testifies to how hierarchical clustering methods will remain crucial for a long time to come.

We have looked at both traditional agglomerative hierarchical clustering, and more recent developments in grid or cell based approaches. We have discussed various algorithmic aspects, including well-definedness (e.g. inversions) and computational properties. We have also touched on a number of application domains, again in areas that reach back over some decades (chemoinformatics) or many decades (information retrieval, which motivated much early work in clustering, including hierarchical clustering), and more recent application domains (such as hierarchical model-based clustering approaches).

10 References

1. Agrawal R, Gehrke J, Gunopulos D, Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, pp. 94–105.

2. Albatineh AN, Niewiadomska-Bugaj M, Mihalko D. On similarity indices and correction for chance agreement. *Journal of Classification*, 2006, 23(2), 301–313.

3. Anderberg MR. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

4. Ankerst M, Breunig M, Kriegel H, Sander J. OPTICS: Ordering points to identify the clustering structure. ACM SIGMOD International Conference on Management of Data. ACM Press, 1999, pp. 49–60.

5. Bader DA, Meyerhenke H, Sanders P, Wagner D. *Graph Partitioning and Graph Clustering*, Contemporary Mathematics Vol. 588, American Mathematical Society, Providence RI, 2013.

6. Bécue-Bertaut M, Kostov B, Morin A, Naro, G. Rhetorical strategy in forensic speeches: Multidimensional statistics based methodology. *Journal of Classification*, 2014, 31:85106.

7. Benzécri JP. *L'Analyse des Données. I. La Taxinomie*, Dunod, Paris, 1979 (3rd ed.).

8. Blashfield RK and Aldenderfer MS. The literature on cluster analysis *Multivariate Behavioral Research* 1978, 13: 271–295.

9. Bolshakova N, Azuaje, F. Cluster validation techniques for genome expression data. *Signal Processing*, 2003, 83(4), 825–833.

10. Bruynooghe M. Méthodes nouvelles en classification automatique des données taxinomiques nombreuses. *Statistique et Analyse des Données* 1977, no. 3, 24–42.

11. Chang J-W, Jin D-S. A new cell-based clustering method for large, high-dimensional data in data mining applications. In: SAC '02: Proceedings of the 2002 ACM Symposium on Applied Computing. New York: ACM, 2002, 503–507.

12. Dash M, Liu H, Xu X. $1 + 1 > 2$: Merging distance and density based clustering. In: DASFAA '01: Proceedings of the 7th International Conference on Database Systems for Advanced Applications. Washington, DC: IEEE Computer Society, 2001, 32–39.

13. Dash M, Liu H, Scheuermann P, Lee Tan K. Fast hierarchical clustering and its validation. *Data and Knowledge Engineering*, 2003, 44(1), 109–138.

14. Day WHE, Edelsbrunner H. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification* 1984, 1: 7–24.

15. Defays D. An efficient algorithm for a complete link method *Computer Journal* 1977, 20:364–366.

16. de Rham C. La classification hiérarchique ascendante selon la méthode des voisins réciproques. *Les Cahiers de l'Analyse des Données* 1980, V: 135–144.

17. Deza MM, Deza E. *Encyclopedia of Distances*. Springer, Berlin, 2009.

18. Dittenbach M, Rauber A, Merkl D. Uncovering the hierarchical structure in data using the growing hierarchical self-organizing map. *Neurocomputing*, 2002, 48(1–4):199–216.

19. Endo M, Ueno M, Tanabe T. A clustering method using hierarchical self-organizing maps. *Journal of VLSI Signal Processing* 32:105–118, 2002.

20. Ester M, Kriegel H-P, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In 2nd International Conference on Knowledge Discovery and Data Mining. AAAI Press, 1996, 226–231.

21. Gan G, Ma C, Wu J. *Data Clustering Theory, Algorithms, and Applications* Society for Industrial and Applied Mathematics. SIAM, 2007.

22. Gillet VJ, Wild DJ, Willett P, Bradshaw J. Similarity and dissimilarity methods for processing chemical structure databases. *Computer Journal* 1998, 41: 547–558.

23. Gondran M. Valeurs propres et vecteurs propres en classification hiérarchique. *RAIRO Informatique Théorique* 1976, 10(3): 39–46.

24. Gordon AD. *Classification*, Chapman and Hall, London, 1981.

25. Gordon AD. A review of hierarchical classification. *Journal of the Royal Statistical Society A* 1987, 150: 119–137.

26. Grabusts P, Borisov A. Using grid-clustering methods in data classification. In: PARELEC '02: Proceedings of the International Conference on Parallel Computing in Electrical Engineering.Washington, DC: IEEE Computer Society, 2002.

27. Graham RH and Hell P. On the history of the minimum spanning tree problem. *Annals of the History of Computing* 1985 7: 43–57.

28. Griffiths A, Robinson LA, Willett P. Hierarchic agglomerative clustering methods for automatic document classification. *Journal of Documentation* 1984, 40: 175–205.

29. Halkidi M, Batistakis Y, Vazirgiannis M. On clustering validation techniques. *Journal of Intelligent Information Systems*, 2001, 17(2–3), 107–145.

30. Halkidi M, Batistakis Y, Vazirgiannis M. Cluster validity methods: part I. *ACM SIGMOD Record*, 2002, 31(2), 40–45.

31. Hinneburg A, Keim DA. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceeding of the 4th International Conference on Knowledge Discovery and Data Mining. New York: AAAI Press, 1998, 58–68.

32. Hinneburg A, Keim D. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In: VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1999, 506–517.

33. Jain AK, Dubes RC. *Algorithms For Clustering Data* Prentice-Hall, Englwood Cliffs, 1988.

34. Jain AK, Murty, MN, Flynn PJ. Data clustering: a review. *ACM Computing Surveys* 1999, 31: 264–323.

35. Janowitz, MF. *Ordinal and Relational Clustering*, World Scientific, Singapore, 2010.

36. Juan J. Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques. *Les Cahiers de l'Analyse des Données* 1982, VII: 219–225.

37. Kohonen T. *Self-Organization and Associative Memory* Springer, Berlin, 1984.

38. Kohonen T. *Self-Organizing Maps*, 3rd edn., Springer, Berlin, 2001.

39. Lampinen J, Oja E. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision* 2: 261–272, 1992.

40. Legendre P, Legendre, L. *Numerical Ecology*, 3rd edn., 2012, Elsevier, Amsterdam.

41. Lerman, IC. *Classification et Analyse Ordinale des Données*, Dunod, Paris, 1981.

42. Le Roux B, Rouanet H. *Geometric Data Analysis: From Correspondence Analysis to Structured Data Analysis*, Kluwer, Dordrecht, 2004.

43. von Luxburg U A. tutorial on spectral clustering. *Statistics and Computing* 1997, 17(4): 395–416.

44. March ST. Techniques for structuring database records. *ACM Computing Surveys* 1983, 15: 45–79.

45. Miikkulainien R. Script recognition with hierarchical feature maps. *Connection Science* 1990, 2: 83–101.

46. Mirkin B. *Mathematical Classification and Clustering* Kluwer, Dordrecht, 1996.

47. Murtagh F. A survey of recent advances in hierarchical clustering algorithms. *Computer Journal* 1983, 26, 354–359.

48. Murtagh F. Complexities of hierarchic clustering algorithms: state of the art. *Computational Statistics Quarterly* 1984, 1: 101–113.

49. Murtagh F *Multidimensional Clustering Algorithms.* Physica-Verlag, Würzburg, 1985.

50. Murtagh F. *Correspondence Analysis and Data Coding with Java and R*, Chapman and Hall, Boca Raton, 2005.

51. Murtagh F. The Haar wavelet transform of a dendrogram. *Journal of Classification* 2007, 24: 3–32.

52. Murtagh F. *Data Science Foundations: Geometry and Topology of Complex Hierarchic Systems and Big Data Analytics.* Chapman and Hall/CRC Press, 2017, Boca Raton, FL.

53. Murtagh F, Hernández-Pajares M. The Kohonen self-organizing map method: an assessment, *Journal of Classification* 1995, 12:165-190.

54. Murtagh F, Raftery AE, Starck JL. Bayesian inference for multiband image segmentation via model-based clustering trees. *Image and Vision Computing* 2005, 23: 587–596.

55. Murtagh F, Ganz A, McKie S. The structure of narrative: the case of film scripts. *Pattern Recognition* 2009, 42: 302–312.

56. Murtagh F, Downs G, Contreras P. Hierarchical clustering of massive, high dimensional data sets by exploiting ultrametric embedding. *SIAM Journal on Scientific Computing* 2008, 30(2): 707–730.

57. Murtagh, F. and Contreras, P. Algorithms for hierarchical clustering: an overview, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2012, 2(1), 86–97.

58. Murtagh F, Legendre P. Ward's hierarchical agglomerative clustering method: which algorithm's implement Ward's criterion? *Journal of Classification* 2014, 31: 274–295.

59. Park NH, Lee WS. Statistical grid-based clustering over data streams. *SIGMOD Record* 2004, 33(1): 32–37.

60. Rapoport A, Fillenbaum S. An experimental study of semantic structures, in Eds. A.K. Romney, R.N. Shepard and S.B. Nerlove. *Multidimensional Scaling; Theory and Applications in the Behavioral Sciences. Vol. 2, Applications*, Seminar Press, New York, 1972, 93–131.

61. Rohlf FJ. Algorithm 76: Hierarchical clustering using the minimum spanning tree. *Computer Journal* 1973, 16: 93–95.

62. Sander J, Ester M, Kriegel H.-P, Xu X. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining Knowledge Discovery* 1998, 2(2): 169–194.

63. Schikuta E. Grid-clustering: An efficient hierarchical clustering method for very large data sets. In: ICPR '96: Proceedings of the 13th International Conference on Pattern Recognition. Washington, DC: IEEE Computer Society, 1996, 101–105.

64. Sheikholeslami G, Chatterjee S, Zhang A. Wavecluster: a wavelet based clustering approach for spatial data in very large databases. *The VLDB Journal*, 2000, 8(3–4): 289–304.

65. Sibson R. SLINK: an optimally efficient algorithm for the single link cluster method. *Computer Journal*, 1973, 16: 30–34.

66. Sneath PHA, Sokal RR. *Numerical Taxonomy*, Freeman, San Francisco, 1973.

67. Tino P, Nabney I. Hierarchical GTM: constructing localized non-linear projection manifolds in a principled way. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002, 24(5): 639–656.

68. van Rijsbergen CJ. *Information Retrieval* Butterworths, London, 1979 (2nd ed.).

69. Vinh NX, Epps J, Bailey J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, New York, NY: ACM, 2009, 1073–1080.

70. Wang L, Wang Z-O. CUBN: a clustering algorithm based on density and distance. In: Proceeding of the 2003 International Conference on Machine Learning and Cybernetics. IEEE Press, 2003, 108–112.

71. Wang W, Yang J, Muntz R. STING: A statistical information grid approach to spatial data mining. In VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases.San Francisco, CA: Morgan Kaufmann Publishers Inc., 1997, 18–195.

72. Wang Y, Freedman MI, Kung S-Y. Probabilistic principal component subspaces: A hierarchical finite mixture model for data visualization. *IEEE Transactions on Neural Networks* 2000, 11(3), 625–636.

73. White HD, McCain KW. Visualization of literatures. In: M.E. Williams, Ed., *Annual Review of Information Science and Technology (ARIST)* 1997, 32:99–168.

74. Vicente D, Vellido A. Review of hierarchical models for data clustering and visualization. In: Giráldez R, Riquelme JC and Aguilar-Ruiz, JS, eds., Tendencias de la Minería de Datos en España. Red Española de Minería de Datos, 2004.

75. Willett P. Efficiency of hierarchic agglomerative clustering using the ICL distributed array processor. *Journal of Documentation* 1989, 45:1–45.

76. Wishart D. Mode analysis: a generalization of nearest neighbour which reduces chaining effects. In Cole AJ, ed., *Numerical Taxonomy*, Academic Press, New York, 282–311, 1969.

77. Xu R, Wunsch D. Survey of clustering algorithms. *IEEE Transactions on Neural Networks* 2005, 16:645–678.

78. Xu R, Wunsch DC. *Clustering* IEEE Computer Society Press, 2008.

79. Xu X, Ester M, Kriegel H-P, Sander J. A distribution-based clustering algorithm for mining in large spatial databases. In: ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 1998, 324–331.

80. Xu X, Jäger J, Kriegel H-P. A fast parallel clustering algorithm for large spatial databases. *Data Mining Knowledge Discovery* 1999, 3(3): 263–290.

81. Zaïane OR, Lee C-H. Clustering spatial data in the presence of obstacles: a density-based approach. In: IDEAS '02: Proceedings of the 2002 International Symposium on Database Engineering and Applications.Washington, DC: IEEE Computer Society, 2002, 214–223.

82. Zhongheng Zhang, Murtagh F, Van Poucke S, Su Lin, Peng Lan. Hierarchical cluster analysis in clinical research with heterogeneous study population: highlighting its visualization with R. *Annals of Translational Medicine*, 5(4), Feb. 2017. http://atm.amegroups.com/article/view/13789/pdf

BOX 1: STORED DATA APPROACH

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

Step 2: Replace two points clustered by representative point (center of gravity) or by cluster fragment.

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

BOX 2: NEAREST NEIGHBOR CHAIN ALGORITHM

Step 1: Select a point arbitrarily.

Step 2: Grow the NN-chain from this point until a pair of RNNs is obtained.

Step 3: Agglomerate these points (replacing with a cluster point, or updating the dissimilarity matrix).

Step 4: From the point which preceded the RNNs (or from any other arbitrary point if the first two points chosen in steps 1 and 2 constituted a pair of RNNs), return to step 2 until only one point remains.

BOX 3: CONTIGUITY CONSTRAINED HIERARCHICAL CLUSTERING

1. Consider each observation in the sequence as constituting a singleton cluster. Determine the closest pair of adjacent observations, and define a cluster from them.
2. Determine and merge the closest pair of adjacent clusters, c_1 and c_2, where closeness is defined by $d(c_1, c_2) = \max \{d_{ii'} \text{ such that } i \in c_1, i' \in c_2\}$.
3. Repeat step 2 until only one cluster remains.

ELSEVIER

Theoretical Computer Science 286 (2002) 139–149

Theoretical Computer Science

www.elsevier.com/locate/tcs

Optimal algorithms for complete linkage clustering in d dimensions

Drago Krznaric, Christos Levcopoulos *

Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden

Abstract

It is shown that the complete linkage clustering of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is a constant, can be computed in optimal $O(n\log n)$ time and linear space, under the L_1 and L_∞-metrics. Furthermore, for every other fixed L_t-metric, it is shown that it can be approximated within an arbitrarily small constant factor in $O(n\log n)$ time and linear space.

Keywords: Complete linkage clustering; Hierarchical clustering; Multidimensional; Optimal algorithms; Approximation algorithms

1. Introduction

Given a set S of n objects, the complete linkage (c-link) method produces a hierarchy of clusters as follows. Initially, each object in S constitutes a cluster. Then, as long as there is more than one cluster, a closest pair of clusters is merged into a single cluster. The c-link distance of two clusters C and C', denoted $\delta(C, C')$, is defined as $\max\{|xy| : x \in C \text{ and } y \in C'\}$, where $|xy|$ stands for the distance (dissimilarity) between objects x and y.

Hierarchical clustering algorithms are of great importance for structuring and interpreting data in domains such as biology, medicine, and image processing. Among the different methods for producing a hierarchy of clusters, the c-link clustering is one of the most well known, and has thus been used for applications [1].

The obvious algorithm for computing the c-link clustering takes cubic time. Using priority queues, Day and Edelsbrunner [4] showed that it can be obtained in $O(n^2 \log n)$ time. A quadratic-time algorithm was described by Murtagh [12]. Later, Křivánek [9] developed a quadratic-time algorithm based on the (a, b)-tree data structure. (These

* Corresponding author. Fax: +46-46-131021.
E-mail address: christos@cs.lth.se (C. Levcopoulos).

PII: S0304-3975(01)00239-0

three algorithms also work for some other clustering methods.) A quadratic-time algorithm that uses linear space was proposed by Defays [5]; however, it only approximates the hierarchy since its output depends on a certain insertion order [2]. Parallel algorithms for the c-link clustering have also been developed [8, 11], but asymptotically the total work was still at least quadratic.

If the input does not need to explicitly contain the distance of each of the quadratic number of pairs of objects (for example, when the objects correspond to points in some metric space) then faster algorithms can be found. Recently, we showed that for n points in the Euclidean plane, the c-link clustering can be computed in $O(n \log^2 n)$ time and linear space [7]. In addition, we developed an $O(n \log n + n \log^2(1/\varepsilon))$ algorithm for constructing a c-link ε-approximation. By a *c-link ε-approximation* we mean a hierarchy that can be produced according to the c-link method, except that the following holds before each merging: if P is the pair of clusters next merged and P' is a closest pair of clusters then $\delta(P) \leqslant (1+\varepsilon)\ \delta(P')$. However, considering only the 2-dimensional case is too restrictive for real applications. Therefore, the results of this paper are especially important since, in addition to reducing the time complexity from $O(n \log^2 n)$ to $O(n \log n)$ for the L_1 and L_∞-metrics, they hold also for the multi-dimensional case. (In [7] we used some data structures that are not known to work efficiently in higher dimensions. In this paper, we dispense with those data structures by making a non-trivial use of an efficient data structure for dynamic closest point queries [3]. This gives, at the same time, a simplification of the methods in [7].)

We assume that the input set S corresponds to n distinct points in d-dimensional real space, where $d \geqslant 1$ is an integer constant, and that we are given a real constant $\varepsilon > 0$ and a fixed L_t-metric. We use $|xy|$ to denote the distance between points $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$, that is,

$$|xy| = \left(\sum_{i=1}^{d} |x_i - y_i|^t \right)^{1/t}.$$

(Note that the L_∞-distance is given by $\max\{|x_i - y_i|:\ 1 \leqslant i \leqslant d\}$.) Under these assumptions, we show that a c-link ε-approximation of S can be obtained in $O(n \log n)$ time and $O(n)$ space. Moreover, for $t = 1$ and $t = \infty$, we show that the complete linkage clustering of S can be computed in optimal $O(n \log n)$ time and $O(n)$ space.

The rest of the paper is organized as follows. In the next section, we describe how c-link distances can be approximated in order to compute a c-link ε-approximation. In Section 3, we give a lemma that we use in order to keep track of relevant pairs of clusters during the merging process. Then, in Section 4 we present the algorithm, and in Section 5 we make a run time analysis. In Section 6, we note that the algorithm fits in the algebraic computation tree model, and we show that it can compute c-link distances exactly under the L_1 and L_∞-metrics, thus concluding that it is optimal up to a constant factor in these two cases. Finally, in Section 7, we make some remarks concerning other hierarchical clustering methods.

2. Approximating complete linkage distances

For each cluster we keep track of k points of the cluster where k is a constant $\geqslant 2d$. These so-called *k-extremes* are used in order to approximate c-link distances of pairs of clusters. The k-extremes of a cluster C are determined by a set $\mathscr{V}$ of k d-dimensional vectors, each vector having its tail at the origin ($\mathscr{V}$ will be defined in a moment). If $v_1, v_2, \ldots, v_k$ are the vectors in $\mathscr{V}$, then the k-extremes of C consist of points $p_1, p_2, \ldots, p_k$ such that each p_i is an extreme point of C in the direction given by vector v_i; that is, there is no point p in C such that $\langle p, v_i\rangle > \langle p_i, v_i\rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. (Note that the k-extremes are not necessarily distinct and that they are not uniquely defined.) To define the set $\mathscr{V}$, let $w \geqslant 1$ be an integer and let

$$W = \{i/w\colon\ i \text{ is an integer and } -w < i < w\}.$$

Next, define V_i^+ as the set of all possible vectors $(x_1, x_2, \ldots, x_d)$ such that $x_i = 1$ and $x_j \in W$ for every $j \neq i$. The set V_i^- is defined in the same way except that it consists of vectors having -1 in position i. Then,

$$\mathscr{V} = \bigcup_{i=1}^{d} (V_i^+ \cup V_i^-).$$

So there are $k = 2d(2w-1)^{d-1}$ vectors in $\mathscr{V}$ and they are almost evenly spread around the origin.

To give a more intuitive picture of the k-extremes, let us examine the 3-dimensional case. Consider the axis-aligned cube centered at the origin such that each of its edges has length 2. Partition each side of this cube into $(2w)^2$ subsquares. Then the vectors in $\mathscr{V}$ correspond to those vertices of these subsquares that do not lie on an edge of the cube. Now, consider k planes that come from infinity and move toward a cluster, where each plane is orthogonal to a distinct vector in $\mathscr{V}$ and comes from the direction pointed by that vector. For each of these planes, select a point of the cluster such that the plane hits this point first (if more than one points are hit simultaneously, select one arbitrarily). Then, the set of all selected points, considering all k planes, constitutes the k-extremes of the cluster.

We define the *k-distance* of two clusters C and C', denoted $\delta_k(C, C')$, as the distance of two k-extremes, one from C and the other from C', that are farthest apart (according to the L_t-metric). It is not hard to realize that for any $\varepsilon > 0$ there exists a constant k, depending on ε and d, such that $\delta_k(P) \leqslant (1+\varepsilon)\delta(P)$ for any pair P of clusters. In the remainder of this paper, let k be such a constant. (For the L_1 and L_∞-metrics, it is shown in Section 6 that we can define the k-extremes so that $\delta_k(P) = \delta(P)$ for any pair P of clusters.)

When a new cluster C is created by merging a pair P of clusters, we compute the k-extremes of C by selecting points among the k-extremes of the clusters of P. More precisely, for each of the k directions, we compare the k-extreme for that direction of

one of the clusters of P with the k-extreme for that direction of the other cluster of P, and select one which is most extreme in that direction.

3. Representing clusters

For each cluster we have a what we call *leader*, which is simply a k-extreme of the cluster (we could actually choose any point of the cluster). All leaders are stored in a data structure for dynamic closest pair queries. Henceforth, that data structure will be referred to as the *DCP-structure*. The DCP-structure supports the following three operations: (i) find a closest pair of points, (ii) delete a point, and (iii) insert a new point. We assume that the DCP-structure uses linear space and makes it possible to carry out each of these three operations in logarithmic worst-case time (although logarithmic amortized time per operation would suffice), and that it fits in the algebraic decision tree model of computation. These requirements can be met by using the data structure of Bespamyatnikh [3].

Initially, each point of the input is inserted into the DCP-structure. Then, each time a new cluster C is created by merging a pair P of clusters, the leader for one of the clusters of P is deleted from the DCP-structure, and the leader for the other cluster of P becomes the leader for C.

As it is described in the next section, our algorithm works in phases where each phase uses the DCP-structure to find relevant pairs of clusters. The following lemma states an upper bound for how fast these pairs can be found.

Lemma 3.1. *Let S be a set of n points in $\mathbb{R}^d$, let l be a positive real, and let κ be an integer such that for each point of S there are at most $\kappa-1$ other points in S within distance less than l from it (distances are according to some arbitrary but fixed L_t-metric, and $d \geqslant 1$ is an integer constant). Then, if the points of S are stored in the DCP-structure, the set $\{(x,y)$: $|xy|<l$ and $x,y \in S\}$ can be found in $O(\kappa^d \eta \log n)$ time, where η denotes the cardinality of the set $\{(x,y)$: $|xy|<3l$ and $x,y \in S\}$.*

Proof. For each point x we have a set S_x where $S_x = \{x\}$ initially. First we repeat the following four steps until the condition at Step 2 holds:

1. Find a closest pair (x,y) of points in the DCP-structure.
2. If $|xy| > l/(\#S_x + \#S_y)$ then halt.
3. Let S_x equal $S_x \cup S_y$.
4. Delete y from the DCP-structure.

After this procedure, let l' be the distance of a closest pair of points in the DCP-structure, and consider an arbitrary point x in the DCP-structure. If p_1 and p_m are any two points in S_x, then there must exist points $p_1, p_2, \ldots, p_m$ in S_x such that $|p_i p_{i+1}| \leqslant l/\#S_x$ for each $i = 1,2,\ldots,m-1$. Hence, $|p_1 p_m| < l$. This means that there are at most κ points in S_x, and that $l' > l/(2\kappa)$. Clearly, the above procedure takes $O(\eta \log n)$ time.

Next, as long as the distance of a closest pair of points in the DCP-structure is $<3l$, we repeat the following five steps:

(i) Find a closest pair (x, y) of points in the DCP-structure.
(ii) Find every point z in the DCP-structure such that $|xz|<3l$.
(iii) For each z found at Step (ii), output every (p_x, p_z) such that $p_x \in S_x$, $p_z \in S_z$, and $|p_x p_z|<l$.
(iv) Output each pair of points in S_x.
(v) Delete x from the DCP-structure.

This procedure outputs the set $\{(x, y)$: $|xy|<l$ and $x, y \in S\}$. Therefore, it remains only to show that it takes $O(\kappa^d \eta \log n)$ time. To do this, we need the following observation.

Observation 3.2. *One iteration of Step* (ii) *takes* $O(\kappa^d \log n)$ *time.*

Proof. Let h be the d-dimensional axis-aligned cube centered at x such that each of its edges has length $6l$. Clearly, every point z that we are looking for lies in h. To find the points in h, partition h into $(12ld/l')^d$ subcubes with edges of length $l'/(2d)$. Note that the L_t-diameter of a subcube is at most equal to its L_1-diameter, which is equal to $l'/2$.

Now, let h_i be one of the subcubes, let c_i be the point at which h_i is centered, and suppose that there is a point z of the DCP-structure that lies in h_i. Then, if we insert c_i into the DCP-structure, (c_i, z) becomes the closest pair of points in the DCP-structure. This is because $|c_i z|<l'/2$ whereas $|zz'|\geqslant l'$ for every $z' \neq c_i$ in the DCP-structure. Hence, we can find all points in h by using one insertion, query, and deletion per subcube, which takes total $O((24d\kappa)^d \log n)$ time (recall that $l'>l/(2\kappa)$ so the number of subcubes is $<(24d\kappa)^d$). □

In Step (iii), we can compute the distance from each point in S_x to each point in S_z for every z found at Step (ii). But, if we in this way compute the distance from a point in S_x to a point in S_z, then that point in S_z must be within distance $<4l$ from x. By partitioning the cube with edges of length $8l$ and centered at x into $(8d)^d$ subcubes, each subcube having L_1-diameter l, we realize that for each point in S_x we compute at most $\kappa(8d)^d$ distances. Consequently, one iteration of Step (iii) takes $O(\kappa^2)$ time. We can now conclude that one iteration of Steps (i) through (v) takes $O(\kappa^d \log n)$ time, and since, we do at most η iterations, the total time used by the algorithm is $O(\kappa^d \eta \log n)$. □

4. The algorithm

Let S be a set of n points in $\mathbb{R}^d$. We compute a c-link ε-approximation of S in a sequence $p_1, p_2, \ldots$ of phases. The objective of phase p_i is to merge every pair P

of clusters such that $l_i \leqslant \delta_k(P) < 2l_i$, where the parameter l_i is defined as follows. At phase p_1, l_1 equals the distance of a closest pair of points in S. For $i > 1$, let l be the distance of a closest pair of leaders in the DCP-structure immediately after phase p_{i-1}. So $\delta_k(P) \geqslant l$ for every pair P of clusters. Therefore, if $l > 2l_{i-1}$ then we set l_i to l, otherwise, we set l_i to $2l_{i-1}$.

Using a priority queue and Lemma 3.1, phase p_i is rather straightforward to compute. First, we do the following two operations:

(a) Find each pair of clusters such that the distance of its corresponding pair of leaders is $< 2l_i$.
(b) Insert each such pair of clusters into a priority queue, initially empty, according to the k-distance of the pair.

Then, as long as the k-distance of a closest (according to the k-distance) pair of clusters in the priority queue is $< 2l_i$, repeat the following three steps:

1. Remove a closest pair P of clusters from the priority queue.
2. Merge P into a single cluster C_P.
3. For each pair (C, C') in the priority queue such that C' is a cluster of P do the following:
   - Remove (C, C') from the priority queue.
   - If $\delta_k(C, C_P) < 2l_i$, then insert (C, C_P) into the priority queue.

5. Run time analysis

As our algorithm works, at phases $p_1, p_2, \ldots, p_{i-1}$ we do not merge a pair P of clusters for which $\delta_k(P) \geqslant 2l_{i-1}$. Hence, at the beginning of p_i, there must be at least one pair of clusters of k-distance $\leqslant 3l_i$. This means that if no merging occurs during p_i, then at least one merging must occur at p_{i+1}. Thus, the total number of phases is no more than $2n - 2$ (the total number of mergings is $n - 1$).

To proceed with the analysis we need a couple of notations. Let $\mathscr{P}_i$ be the set consisting of every pair of clusters such that the pair exists at some time during phase p_i and the distance of its corresponding pair of leaders is $< 6l_i$. Further, let η_i denote the cardinality of $\mathscr{P}_i$. Finally, let κ be the smallest integer such that the following holds at any time during any phase p_i: for any leader there are at most $\kappa - 1$ other leaders within distance $< 6l_i$ from it. (In Lemma 5.1 below we show that κ is never greater than some constant.)

By Lemma 3.1, the operation (a) in Section 4 takes $O(\kappa^d \eta_i \log n)$ time. Clearly, (b) takes $O(\eta_i \log \eta_i)$ time. Observe that only a pair in $\mathscr{P}_i$ may be considered at Step 3, and that each pair in $\mathscr{P}_i$ is considered at most once at Step 3. So, the total time for Step 3 is $O(\eta_i \log \eta_i)$. Next, each time we iterate Steps 1 through 3, a pair of clusters is merged into a single cluster. Consequently, the total time for Steps 1 and 2 is $O(m \log \eta_i)$, where m denotes the total number of mergings

performed during p_i. We can thus conclude that the total time used by phase p_i is $O(\kappa^d \eta_i \log n + \eta_i \log \eta_i + m \log \eta_i)$.

Now, consider a pair P in $\mathscr{P}_i$. It holds that $\delta_k(P) < 10 l_i$. Therefore, one of the clusters of P must participate in a merging at one of the phases p_i, p_{i+1}, or p_{i+2}. Let us associate each pair in $\mathscr{P}_i$ with the first merging in which one of the clusters of the pair participate. If we do this for each set $\mathscr{P}_i$, we associate at most $3(\kappa - 1)$ pairs to each merging. Hence, the total sum of the η_i's is $O(\kappa n)$, which implies that the total time used by the algorithm is $O(\kappa^{d+1} n \log(\kappa n))$.

From the following lemma we can conclude that our algorithm actually runs in $O(n \log n)$ time. (A special case of this lemma handling only the Euclidean plane was stated but not proved in [7].)

Lemma 5.1. *There exists a constant* (*depending on d and k*) *greater than* κ.

Proof. Throughout the proof, by a *cube* we mean a d-dimensional axis-aligned cube. The proof is by induction on i. The induction hypothesis is as follows: at the beginning of phase p_i (before any clusters are merged) any cube having edges of length l_i contains at most $\lambda(d+1)^d$ leaders, where $\lambda \geqslant 1$ is a constant that we specify later. Recall, at the beginning of p_1, each leader corresponds to a single point in the input point set S, and l_1 equals the distance of a closest pair of points in S. It is not hard to show that any cube having edges of length l_1 contains at most $(d+1)^d$ points of S. Thus the statement is true for $i = 1$. Let h be an arbitrary cube with edges of length l_{i+1}. To complete the proof it suffices to show that h contains at most $\lambda(d+1)^d$ leaders at the beginning of phase p_{i+1}.

From the definition of the parameter l_i it follows that if $l_{i+1} \neq 2 l_i$ then, at the beginning of p_{i+1}, the distance of a closest pair of leaders equals l_{i+1}. So, similarly as for $i = 1$, h contains at most $(d+1)^d$ leaders in this case. Therefore, in the continuation we can (and will) assume that $l_{i+1} = 2 l_i$.

From the set of clusters that exist at the beginning of phase p_i we extract two subsets H and H' as follows. The set H consists of every cluster that has a k-extreme in h. The set H' consists of every cluster that, at the end of phase p_i, is included in a cluster containing a cluster of H. Note that $H \subseteq H'$ and that the clusters in H' are merged only with each other during phase p_i. We aim to show that sufficiently many mergings of clusters in H' occur during phase p_i. But first we need some more definitions.

Let h' be the cube concentric with h and having edges of length $6 l_i$. So each cluster in H' has all of its k-extremes in h'. Define the *triggers* as the r^d cubes that partition h' into subcubes with edges of length $6 l_i / r$, where $r = 6d(1 + (d+1)\log_2 6)$. Now, consider a cluster with all its k-extremes in h'. For each k-extreme of the cluster we select the trigger in which it is contained, thus selecting k triggers $t_1, t_2, \ldots, t_k$. We say that the cluster is of *type* τ, where τ is the unordered k-tuple $(t_1, t_2, \ldots, t_k)$. We are now in position to set the constant λ, namely, λ equals the maximum number of

distinct types of clusters. By standard combinatorics,

$$\lambda = \binom{r^d + k - 1}{k}.$$

Next, define the *k-diameter* of a cluster as the distance of its two k-extremes that are farthest apart. So, if C and C' are any two clusters produced by our algorithm, the k-diameter of $C \cup C'$ equals $\delta_k(C, C')$.

Observation 5.2. *Let C and C' be two clusters with all their k-extremes in h' such that each of them is of type τ and has k-diameter less than l. Then the k-diameter of $C \cup C'$ is less than $l + 6l_i d/r$.*

Proof. Since C and C' are of the same type, there are k triggers (not necessarily distinct) such that each of them contains two k-extremes, one from C and the other from C'. Let p and p' be any two k-extremes of $C \cup C'$ such that the distance between them is maximized. So the k-diameter of $C \cup C'$ equals $|pp'|$. We can assume that p is a k-extreme of C and p' is a k-extreme of C', because $|pp'|$ would otherwise be $< l$. Let t_p be the trigger containing p. As mentioned above, t_p also contains a k-extreme q' of C', and we know that $|p'q'| < l$. But $|pq'|$ is at most the L_1-diameter of t_p, which is equal to $6l_i d/r$. Hence, by triangle inequality, $|pp'| < l + 6l_i d/r$. □

Suppose that there is a subset T of H' consisting only of (at least two) clusters that are of the same type. Let C and C' be two clusters in T. First we observe that both C and C' have k-diameter $< l_i$, because they were created at some phase before p_i. By Observation 5.2, the k-diameter of $C \cup C'$ is $< l_i + 6l_i d/r$. Consequently, during p_i, at least one of C and C', let us say C, will be merged with a cluster C'' such that $C \cup C''$ has k-diameter $< l_i + 6l_i d/r$ (C'' and C' might be the same cluster). Thus, we realize that each cluster in T except at most one will participate in a merging during phase p_i, in such a way that the new clusters resulting from these mergings have k-diameters $< l_i + 6l_i d/r$.

In the remainder we only consider the clusters in H' and those clusters that are created during p_i by merging two or more clusters of H'. Let n' be the number of clusters in H'. As indicated in the previous paragraph, at least $n' - \lambda$ clusters will participate in a merging during p_i, and the new clusters resulting from these mergings have k-diameters $< l_i + 6l_i d/r$. The number of clusters that remain after these mergings is at most $(n' - \lambda)/2 + \lambda$. We can repeat the scenario for these clusters. After having done that we are left with at most $(n' - \lambda)/2^2 + \lambda$ clusters, each cluster having k-diameter $< l_i + 2 \times 6l_i d/r$. Indeed, we can repeat the scenario as long as we do not merge two clusters whose union has k-diameter $\geqslant 2l_i$ (we may assume that there are after each repetition sufficiently many clusters left for the next repetition to work).

Now, if we repeat the scenario j times, we are left with at most

$$\frac{n' - \lambda}{2^j} + \lambda < \frac{n'}{2^j} + \lambda$$

clusters, each cluster having k-diameter less than

$$l_i + j \cdot 6l_i d/r,$$

which is $<2l_i$ for $j = \lfloor 1+(d+1)\log_2 6 \rfloor > (d+1)\log_2 6$. But since h' can be partitioned into 6^d subcubes with edges of length l_i, we have by our induction hypothesis that $n' \leqslant 6^d \lambda(d+1)^d$. Hence, at the end of phase p_i, the number of clusters that have a k-extreme in h is at most

$$\frac{6^d \lambda(d+1)^d}{2^{(d+1)\log_2 6}} + \lambda = \frac{\lambda(d+1)^d}{6} + \lambda < \lambda(d+1)^d,$$

which completes the proof. □

We can summarize this section by the following theorem.

Theorem 5.3. *Let $\varepsilon > 0$ be a real constant, and let S be a set of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is an integer constant. Then, under any fixed L_t-metric, a c-link ε-approximation of S can be computed in $O(n \log n)$ time and $O(n)$ space.*

6. Under the L_1 and L_∞-metrics

Under the L_∞-metric it is possible to define the k-extremes so that $\delta_k(P) = \delta(P)$ for any pair P of clusters. To see this, consider a cluster and a point x. All points within L_∞-distance $\leqslant l$ from x, for some $l > 0$, comprise an axis-aligned d-dimensional cube centered at x such that each of its edges has length $2l$. Hence, a point of the cluster that is farthest away from x must be an extreme point of the cluster in one of the $2d$ coordinate directions. Therefore, under the L_∞-metric, for every cluster we only need to keep track of an extreme point of the cluster in each of the $2d$ coordinate directions. (This corresponds to the definition of k-extremes given in Section 2 if the set $\mathcal{V}$ of vectors is defined for $w = 1$.)

A similar observation can be made for the L_1-metric. In this case, all points within L_1-distance $\leqslant l$ from x comprise a d-dimensional cross-polytope in which each edge has length $\sqrt{2}\,l$, that is, a regular polytope with d diagonals (a straight-line segment connecting two vertices of the polytope such that its interior does not intersect the boundary of the polytope) each diagonal being parallel to one of the coordinate axes. (In $\mathbb{R}^3$ it is a octahedron with diagonals parallel to the coordinate axes.) But this polytope is bounded by 2^d planes. Therefore, under the L_1-metric, for every cluster we only need to keep track of an extreme point of the cluster in each of these 2^d directions. The set $\mathcal{V}$ of vectors given in Section 2 has to be defined in a slightly different way in order to correspond to these 2^d directions. Namely, in this case we define $\mathcal{V}$ so that it consists of all possible vectors $(x_1, x_2, \ldots, x_d)$ such that each $x_i \in \{1, -1\}$. So the vectors in $\mathcal{V}$ correspond to the binary representation of $0, 1, \ldots, d$ where each 0 is replaced by -1.

It is not hard to see that our algorithm can be implemented using only operations allowed by the algebraic decision tree model. Moreover, by reduction to the static closest pair problem it is easy to conclude that $\Omega(n \log n)$ is a lower bound for the c-link clustering, even if we restrict ourselves to 1-dimensional space, in the algebraic decision tree model (see, for example, [13]). Thus, our algorithm is optimal in that model of computation.

This section is summarized in the following theorem.

Theorem 6.1. *Let S be a set of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is an integer constant. Then, under the L_1 and L_∞-metrics, the complete linkage clustering of S can be computed in $O(n \log n)$ time and $O(n)$ space, which is optimal in the algebraic decision tree model.*

7. Final remarks

The c-link clustering belongs to a family of clustering methods known as SAHN methods (sequential, agglomerative, hierarchical, and nonoverlapping). Two other methods in this family are the centroid [4, 6, 14] and the median [4, 6, 10] method. Given n points in $\mathbb{R}^d$, these two methods work by repeatedly replacing a closest pair of points with a single (centroid respectively median) point. Hence, using the DCP-structure of Bespamyatnikh [3], these two clustering methods can be trivially computed in $O(n \log n)$ time, under any fixed L_t-metric.

References

[1] P. Arabie, L.J. Hubert, G. De Soete (Eds.), Clustering and Classification, World Scientific, Singapore, 1996.
[2] F. Aurenhammer, R. Klein, Voronoi diagrams, Tech. Report 198-5, Informatik, FernUniversität, Hagen, Germany, 1996.
[3] S.N. Bespamyatnikh, An optimal algorithm for closest pair maintenance, Proc. 11th ACM Symp. on Computational Geometry, 1995, pp. 152–161.
[4] W.H.E. Day, H. Edelsbrunner, Efficient algorithms for agglomerative hierarchical clustering methods, J. Classification 1 (1) (1984) 7–24.
[5] D. Defays, An efficient algorithm for a complete link method, Comput. J. 20 (1977) 364–366.
[6] J.C. Gower, A comparison of some methods of cluster analysis, Biometrics 23 (1967) 623–638.
[7] D. Krznaric, C. Levcopoulos, Fast algorithms for compete linkage clustering, Discrete Comput. Geom. 19 (1998) 131–145, A preliminary version of that paper appeared as: The first subquadratic algorithm for complete linkage clustering. Proc. 6th Internat. Symp. on Algorithms and Computation, Lecture notes in Computer Science, Vol. 1004, Springer, Berlin, 1995, pp. 392–401.
[8] T. Kurita, An efficient agglomerative clustering algorithm using a heap, Pattern Recognition 24 (3) (1991) 205–209.
[9] M. Křivánek, Connected admissible hierarchical clustering. Paper presented at the DIANA III Conf. Bechyne, Czechoslovakia, June 1990. Published in the KAM-Series as Tech. Report No. 90-189, 8 pages, School of Computer Science, Faculty of Mathematics and Physics, Charles University, Prague, 1990.
[10] G.N. Lance, W.T. Williams, A generalised sorting strategy for computer classifications, Nature 212 (1966) 218.

[11] X. Li, Parallel algorithms for hierarchical clustering and cluster validity, IEEE Trans. Pattern Anal. Mach. Intell. 12 (11) (1990) 1088–1092.
[12] F. Murtagh, Complexities of hierarchic clustering algorithms: state of the art, Comput. Statist. Quart. 1 (1984) 101–113.
[13] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer, New York, 1985.
[14] R.R. Sokal, C.D. Michener, A statistical method for evaluating systematic relationships, Univ. Kans. Sci. Bull. 38 (1958) 1409–1438.