Algorithm 65

AN IMPROVED CLUSTERING ALGORITHM

A. H. J. Sale
Basser Computing Department
University of Sydney

Author's Note:

This set of routines produces the same type of results as Algorithm 47 (A Clustering Algorithm) in that it will generate and return all subsets of objects from a given set that satisfy the condition that the maximum dissimilarity (or 'distance') between members of the subset is less than the least dissimilarity between any member of the subset and any object not in the subset. Excluded from this definition are sets consisting of single objects only, and the set of all the objects. It differs from Algorithm 47 in its overall design, in its method of returning results, in its use of storage, and in its execution speed. Subsequent paragraphs detail these differences.

The Algorithm given here has been tested on an IBM 7040 under both the IBFTC and WATFOR compilers with both constructed examples and random data for numbers of objects in a set ranging from 10 to 100. In all cases it produced correct results.

The parameters required are fully detailed in the source language comments; however it should be pointed out here that storage of the order of n^2 variables is required for a set of n objects, and the run time should vary approximately as n^3. The following criteria were employed in the design of the program:

1. The vector of distances should not be destroyed by the routine.
2. Since large numbers of objects are to be expected, the execution time should be as short as possible, and as little storage as possible should be used.
3. The results were to be returned to the calling routine, rather than printing them (which makes them unavailable for further processing).
4. The operation of the routines should be clear and simple to understand.

The process used is to search for a smallest dissimilarity between a pair of subsets, then to merge those two into one, simultaneously making the dissimilarities between them inaccessible, but noting the cluster 'diameter'. Then all other subsets are scanned to make a choice between the dissimilarities to each of the (now merged) subsets. Two of the four values are kept: the least and the greatest. The least dissimilarity is needed for subsequent searches; the greatest is needed to determine the subset diameter. During this scan the merged subset diameter is compared with the least external dissimilarities: if it is less than all of these then the subset is a cluster, otherwise another merge must be initiated. To start the process all objects are regarded as subsets with identical maximum and minimum inter-subset dissimilarities.

This algorithm arose out of a certification of Algorithm 47 in which it appeared that it contained several major bottlenecks: the initial sort, the testing of the clustering condition, and final sort, all of which had run-times proportional to n^4.

In storage requirements the new Algorithm appears superior in program size, and certainly in data storage. To verify the expected improvement in execution time a series of examples were run using test data of points with random (x, y) co-

Table 1

Comparative run-times on identical data for Algorithm 47, an improved version of Algorithm 47, and Algorithm 65

NUMBER OF OBJECTS	TIMES RECORDED ON IBM 7040 FOR:		
	ALGORITHM 47	IMPROVED ALGORITHM 47	ALGORITHM 65
10	4·5 sec	2·2 sec	1·2 sec
20	32·9	17·4	4·6
30	177·8	78·6	12·8
40	516·0	228·1	27·7
50	1281·8	528·6	51·0
60	2377·6	1079·7	85·2
70	—	1868·2	132·9
80	—	—	195·9
90	—	—	273·8
100	—	—	373·6

ordinates in a square two-dimensional space. These results are shown in **Table 1**, for numbers of objects from 10 to 100. These results verify the predictions and show a large superiority of the new Algorithm over the original. It should be pointed out that the times will vary slightly with the data: the test data is characterised by few large clusters but many doublets.

To sum up, this new Algorithm seems to be superior in both program and data space utilisation, and in execution time. There seems however to be one place where a version of Algorithm 47 might still be preferable: where the large tables (of size $n \times (n-1)/2$ variables) are sorted and stored on a serial access medium (for example magnetic tape). The reason for this is that Algorithm 47 always runs serially through the tables (wholly or partially) while the new Algorithm requires random access to the tables. A more recent algorithm (Algorithm 52) while slightly different in function points the way to perform the same clustering process with approximately:

(number of different dissimilarities) + $\log_2(n^2)$

passes through tapes holding $2n^2$ variables. This for some cases would produce run-times proportional to n^2; for others proportional to $n^2 \log n^2$. Whether a problem too large to fit in core can be completed in this way in a reasonable time will of course depend on the machine.

Reference

VAN RIJSBERGEN, C. J. (1970). Algorithm 47: A clustering algorithm, *The Computer Journal*, Vol. 13, No. 1, pp. 113-115.


```
      SUBROUTINE CLUST3(KSIZE,KDAT,TABLE,SWITCH,KOUT,KLENG,DIAM,
     1SPACE,KNUM,KLINK,KHEAD,MAXMIN)
C
C     INPUT VARIABLES
C       -UNALTERED BY CLUST3-
C       -MUST NOT BE ALTERED UNTIL -SWITCH- BECOMES .FALSE. -
C       KSIZE
C         NUMBER OF OBJECTS
C       KDAT
C         SIZE OF TABLE (=KSIZE*(KSIZE-1)/2)
C       TABLE
C         VECTOR OF LENGTH KDAT HOLDING DISTANCES BETWEEN
C         OBJECTS. THE MAPPING FUNCTION IS -LOCN-, AND IF
C         I IS GREATER THAN J, THEN THE DISTANCE BETWEEN I AND
C         J IS AT TABLE(((I-1)*(I-2))/2+J)
C
C     CONTROL VARIABLE
C       SWITCH
C         SET BEFORE FIRST CALL TO .FALSE., THE ROUTINE THEN
C         SETS IT .TRUE. AND IT MUST RETAIN THIS VALUE UNTIL THE
C         ROUTINE ITSELF SETS IT .FALSE. BEFORE -RETURN-ING.
C         THIS SIGNIFIES THAT NO MORE CLUSTERS CAN BE FOUND,
C         AND THAT THE OUTPUT RESULTS ARE UNDEFINED
C
C     OUTPUT VARIABLES
C       -SET BY CLUST3-
C       -UNDEFINED AT ENTRY-
C       -MAY BE FREELY ALTERED-
C       KOUT
C         VECTOR OF LENGTH -KSIZE- CONTAINING A SORTED LIST OF
C         -KLENG- OBJECTS FORMING A CLUSTER. THE CONTENTS BEYOND
C         KOUT(KLENG) ARE NOT DEFINED
C       KLENG
C         NUMBER OF OBJECTS IN THE CLUSTER
C       DIAM
C         THE DIAMETER OF THE CLUSTER (MAXIMUM INTRA-CLUSTER
C         DISTANCE)
C       SPACE
C         THE SEPARATION SPACE (DISTANCE FROM THIS CLUSTER TO
C         ITS NEAREST NEIGHBOUR)
C
C     WORKING VARIABLES
C       -SET BY CLUST3-
C       -UNDEFINED AT FIRST ENTRY-
C       -THEY MUST NOT BE ALTERED UNTIL -SWITCH- BECOMES .FALSE. -
C       KNUM
C         THE NUMBER OF SETS AS YET UNCOALESCED
C       KLINK
C         WORK VECTOR OF LENGTH KSIZE, HOLDS LINKED OBJECT
C         INFORMATION
C       KHEAD
C         WORK VECTOR OF LENGTH KSIZE, HOLDS LIST HEADS
C       MAXMIN
C         WORK VECTOR OF LENGTH KDAT, HOLDS POINTERS TO THE
C         MAXIMUM AND MINIMUM INTER-SET DISTANCES. THE TWO
C         POINTERS MAX AND MIN ARE PACKED INTO ONE INTEGER
C
C     SPECIAL COMMENTS
C       INTEGER OVERFLOW MUST NOT OCCUR FOR (KDAT*(KDAT+2))
C       DUE TO THE PACKING INTO MAXMIN. IN CASE OF DIFFICULTY
C       MAXMIN MAY BE SPLIT INTO TWO ARRAYS MMAX AND MMIN, THUS
C       ELIMINATING THE NEED FOR THE VARIABLE -JPACK- AND THE
C       ROUTINE -UNPAK-
C
C     ERROR EXITS
C       NONE, EXCEPT AS EXPLAINED FOR -SWITCH-
C       THE ACTION OF CLUST3 IS UNDEFINED IF THE INPUT VARIABLES
C       OR THE WORKING VARIABLES ARE ALTERED BETWEEN A RETURN
C       FROM CLUST3 WITH -SWITCH- .TRUE. AND A SUBSEQUENT CALL
C       TO CLUST3 WITH -SWITCH- AGAIN .TRUE.
C
C     SPECIFICATIONS
      INTEGER KSIZE,KDAT,KLENG,KNUM
      REAL DIAM,SPACE
      LOGICAL SWITCH
      INTEGER KLINK(KSIZE),KHEAD(KSIZE),KOUT(KSIZE),MAXMIN(KDAT)
      REAL TABLE(KDAT)
C     DECLARATIONS
      INTEGER J,L,M,KT,KSIZET,JPACK
      INTEGER LMAX,LMIN,MAX,MIN,JSET1,JSET2
      REAL RMIN
      LOGICAL TSW
C
C     ROUTINE START POINT
C     TEST TO SEE IF INITIALISING ENTRY TO CLUST3
      IF (SWITCH) GO TO 3
C     INITIALISE THE TABLES AND VARIABLES
      JPACK=KDAT+1
      DO 1 J=1,KSIZE
      KLINK(J)=0
      KHEAD(J)=J
    1 CONTINUE
      DO 2 J=1,KDAT
      MAXMIN(J)=J*JPACK+J
    2 CONTINUE
      KNUM=KSIZE
      SWITCH=.TRUE.
C
C     TEST TO SEE IF TO REJECT THE APPLICATION FOR A CLUSTER
    3 IF (KNUM.GT.2) GO TO 4
C     IF THERE ARE ONLY TWO SETS TO COALESCE, WE CAN ONLY
C     GET THE SET OF ALL OBJECTS, SO GIVE UP
      SWITCH=.FALSE.
      RETURN
C
C     SCAN FOR THE ABSOLUTE MINIMUM INTER-SET DISTANCE
    4 TSW=.TRUE.
C     RUN DOWN THE CLUSTER TABLE
      KSIZET=KSIZE-1
      DO 7 J=1,KSIZET
      IF (KHEAD(J).EQ.0) GO TO 7
C     GOT AN EXISTING CLUSTER
C     IS THERE A HIGHER NUMBERED ONE TOO
      L=J+1
      DO 6 M=L,KSIZE
      IF (KHEAD(M).EQ.0) GO TO 6
C     GOT A PAIR, GET THE LINK POINTERS
      KT=LOCN(J,M)
      CALL UNPAK(LMAX,LMIN,MAXMIN(KT),JPACK)
C     IF ITS THE FIRST PAIR, ACCEPT THE DISTANCES
      IF (TSW) GO TO 5
C     IF THE MIN DISTANCE IS LESS THAN THE PRESUMED MIN, TAKE IT
      IF (TABLE(LMIN).GE.RMIN) GO TO 6
C     KEEP INFORMATION ABOUT THIS PAIR
    5 TSW=.FALSE.
      JSET1=J
      JSET2=M
      RMIN=TABLE(LMIN)
      DIAM=TABLE(LMAX)
    6 CONTINUE
    7 CONTINUE
C     WE NOW HAVE THE CLOSEST PAIR OF CLUSTERS, AND THEIR
C     DIAMETER AS A JOINT CLUSTER
C
C     KEEP THE MAX AND MIN OF THE PAIR DISTANCES
      TSW=.TRUE.
C     RUN THROUGH ALL CLUSTERS, EXCEPT THE TWO SETS FOUND
      DO 9 J=1,KSIZE
      IF (KHEAD(J).EQ.0) GO TO 9
      IF ((J.EQ.JSET1).OR.(J.EQ.JSET2)) GO TO 9
C     GET LOCATIONS OF DISTANCES RELEVANT
      L=LOCN(J,JSET1)
      M=LOCN(J,JSET2)
      CALL UNPAK(LMAX,LMIN,MAXMIN(L),JPACK)
      CALL UNPAK(MAX,MIN,MAXMIN(M),JPACK)
C     KEEP THE LARGEST AND SMALLEST DISTANCE
      IF (TABLE(LMAX).LT.TABLE(MAX)) LMAX=MAX
      IF (TABLE(LMIN).GT.TABLE(MIN)) LMIN=MIN
      MAXMIN(L)=LMAX*JPACK+LMIN
C     ON FIRST PASS KEEP AS ABSOLUTE MIN
      IF (TSW) GO TO 8
C     IS THIS DISTANCE LESS THAN PRESUMED ABSOLUTE MIN
      IF (TABLE(LMIN).GE.RMIN) GO TO 9
    8 RMIN=TABLE(LMIN)
      TSW=.FALSE.
    9 CONTINUE
C     NOW WE HAVE ALL THE DISTANCES FOR JSET1 CORRECT
C
C     JOIN UP ALL THE CLUSTERS
C     AND SIMULTANEOUSLY START BUILDING UP THE OUTPUT VECTOR
      L=KHEAD(JSET1)
      J=1
      KOUT(1)=L
C     RUN ALONG THE LINKS
   10 M=KLINK(L)
      IF (M.EQ.0) GO TO 11
      J=J+1
      KOUT(J)=M
      L=M
      GO TO 10
C     NOW JOIN ON SET 2 BY THE LINK
   11 KLINK(L)=KHEAD(JSET2)
      KHEAD(JSET2)=0
      KNUM=KNUM-1
C     THIS IS THE EARLIEST POINT THAT WE CAN CHECK FOR THE
C     CORRECTNESS OF THE CLUSTER CONDITION
      IF (RMIN.LE.DIAM) GO TO 3
C     GET THEN THE REST OF THE OBJECTS INTO OUTPUT
   12 M=KLINK(L)
      IF (M.EQ.0) GO TO 13
      J=J+1
      KOUT(J)=M
      L=M
      GO TO 12
```

```
C   WE NOW HAVE A CLUSTER, FINISH OFF
 13 KLENG=J
    CALL SORT(KOUT,J)
    SPACE=RMIN
    RETURN
    END
C
    SUBROUTINE SORT(KOUT,KLENG)
    INTEGER KLENG,KOUT(KLENG)
C
C   THIS ROUTINE SORTS THE VECTOR -KOUT- INTO ASCENDING ORDER
C   ELEMENTS KOUT(1) TO KOUT(KLENG) ARE AFFECTED.
C   A SIMPLE BUBBLE SORT IS USED.
C
    INTEGER J,J1,K
    K=KLENG
  1 K=K-1
    IF (K.LE.0) RETURN
    DO 2 J=1,K
    IF (KOUT(J).LE.KOUT(J+1)) GO TO 2
    J1=KOUT(J)
    KOUT(J)=KOUT(J+1)
    KOUT(J+1)=J1
  2 CONTINUE
    GO TO 1
    END
C
    INTEGER FUNCTION LOCN(J,K)
    INTEGER J,K
C
C   LOCN RETURNS THE LOCATION OF THE (J,K) ELEMENT IN THE
C   TRIANGULAR ARRAY STORED IN MAXMIN AND TABLE
C
    IF (J.GT.K) GO TO 1
    LOCN=((K-1)*(K-2))/2+J
    RETURN
  1 LOCN=((J-1)*(J-2))/2+K
    RETURN
    END
C
    SUBROUTINE UNPAK(J,K,L,JPACK)
    INTEGER J,K,L,JPACK
C
C   ROUTINE TO UNPACK TWO INTEGERS FROM ONE.
C     J AND K COME FROM L.
C     JPACK DETERMINES THE PACKING FUNCTION.
C
    J=L/JPACK
    K=L-J*JPACK
    RETURN
    END
```