Current approaches to classification and clump-finding at the Cambridge Language Research Unit

By Karen Sparck Jones and David Jackson*

Computer programs for automatic classification are a desideratum in many fields. Work on suitable procedures for handling large bodies of object/property descriptions has been in progress at the Cambridge Language Research Unit for some years: this paper describes the current series of general-purpose programs which have been developed there, in which classes or "clumps" of objects are obtained, using a similarity matrix, by a simple iterative scan of the universe of objects, distributing them in such a way that an appropriate cohesion function is minimized. This actual clump-finding process is embedded in an overall package in which the information given by a classification is manipulated in a variety of ways. The current applications of the programs, especially for information retrieval, are described.

Work on the use of computers for classification and grouping has been in progress at the Cambridge Language Research Unit for some time, under the general heading of the theory of clumps. This has been chiefly concerned with the use of automatic classification procedures in information retrieval, but other applications, especially to linguistic material have also been examined; and one objective of the research has indeed been to develop general-purpose classification algorithms, on the grounds that these will tend to give more reliable results even if one is only interested in a particular application. The first series of experiments on these lines were carried out by R. M. Needham between 1958 and 1962. These have been fully reported on elsewhere (Ref. 1–4) and the present paper is concerned with the recent developments of this work by the authors, with R. M. Needham as consultant, which are associated with a project for a quite large experiment in the use of automatic classification techniques in information retrieval, and with the advent of a new and more powerful computer in the Cambridge University Mathematical Laboratory.

Most automatic classification procedures have two stages: in the first the input object/property information is processed to form a similarity matrix in which the extent to which all the pairs of objects are related by their common properties is noted; in the second this matrix is manipulated in the search for groups of objects which share common properties. Procedures of this general kind may, however, vary very much in flexibility and in the degree to which they may readily be extended to large data samples, though both flexibility and extensibility are highly desirable, given the current state of research in this field. Our intention in designing the present set of clump-finding programs has been to make it as easy as possible to use different similarity coefficients, and more importantly, different class definitions, while the actual search procedure for finding the classes remains quite simple. The latter is made possible by the use of the notion of cohesion: the boundary between a clump and the remainder of the universe of objects represents a local minimum of the cohesion between objects as defined by their similarity connections, and the particular cohesion function which is used will thus characterize the way in which a clump is separated from its complement. The important point is that a wide range of cohesion functions may be combined with a very simple method of clump-finding in which the universe of objects is scanned to see whether shifting an object from one side to the other of a partition which separates a potential clump from its complement will reduce the current value of the particular function in question. The time taken to find classes is a critical aspect of any classification program, especially for large quantities of material. Realistic samples of linguistic data tend to be large, and we are primarily interested in classification algorithms which are viable in this context. Again, the nature of our empirical material has been such that we have always considered it essential that any class definition which is used is such that it will not be upset by errors in the data. The general approach to classification on which the programs described below are based is therefore one whose object is to find classes which are not necessarily exclusive, and which do not depend on the existence of properties which are shared by all their members, in large bodies of untidy empirical material, and in a reasonably finite time.

Given that the real object of mechanized classification research is to provide viable procedures for dealing with the quantities of data for which hand-classification on a sophisticated basis is unthinkable, or at least highly tedious, it follows that any serious programs must be capable of handling some thousands, and not merely some hundreds, of objects and properties, though the number of object/property descriptions which can actually be managed will naturally vary, since the critical factor is the density of the similarity matrix, and the number of non-zero entries in the matrix is determined

* *Cambridge Language Research Unit*, 20, *Millington Road, Cambridge.*

by the extent to which properties are shared by the objects, rather than by the simple numbers of objects and properties that there are. The present clump-finding algorithms require repeated scans of the matrix, and a realistic view of the amount of computer time which can justifiably be spent on looking for clumps in a given body of material implies that the entire similarity matrix must be held in rapid-access store, though it is in principle possible to use back-up store during the search. The present Cambridge University computer, Titan, has 64,000 words of core store, and it is this together with its greater speed, that has enabled us to embark on a wholly new series of programs and experiments, with much larger quantities of data.

In writing these programs, we have concentrated on the development of what may be called a general-purpose clump-finding engine, which may hopefully be applied to any body of material in a reasonably suitable form, in a variety of ways depending on the selection of different subroutines and according to the specifications represented by a particular choice of options, for example for different similarity definitions. The programs themselves are written in Titan assembly code, chiefly because no suitable higher-level language was available at the time when the project began. This unfortunately means that they cannot usefully be circulated, but we hope that we will be able to carry out experiments here with outside data, provided that this is in an appropriate form. We have in any case found that the use of machine code makes for much faster central loops in the programs, and since the speed with which clumps are found depends to a great extent on these loops, there have been some advantages in the use of the machine code.

The programs

The programs themselves fall into three main groups: one, under the general heading RAS, prepares the data for clump-finding by listing the co-occurrences or associations between all the pairs of objects which have any property in common, and then computing the similarity matrix from this information; the second, C1, carries out the actual search for the clumps, and the third, PC1, processes the resulting object-property-clump information from a variety of points of view: this is desirable both in itself as a means of exhibiting all the characteristics of the classification which has been obtained, and as a means of setting up the environment for any application of the clumps, say for information retrieval.

The input data for RAS is presented as a series of property numbers, each followed by a list of the object numbers possessing the property, with a terminal marker. See **Fig. 1**(*a*) for an example. (Our illustrations all refer to the same body of data, but showing how it is processed as a whole would take too much space, and the figures therefore represent selections only, though some continuity is maintained throughout the series.) Some kinds of data not in the standard form can, however, be accepted by an alternative read routine and converted to standard form, and any data can be inverted so that the objects are treated as properties and vice versa: in the classification procedure no *a priori* assumptions are made about the nature of the one as opposed to the other, and in some applications it may be of interest to attempt a classification of both the initial objects and the initial properties. These property-object lists are then processed so that every object which shares some property with each given object, and so can be said to co-occur with it, is noted, together with the frequency with which they co-occur, which depends on the number of properties they share. For each object, that is, we have a list of all the objects with which it is associated by the possession of a common property, so that the set of lists for all the objects in our universe gives all the associations among the different objects. This information is held in chained list form, and is obtained by scanning the object list for each property, picking up each pair of objects which occurs in it, and either inserting the co-occurrence as a new item in the list of objects associated with each member of the pair, or incrementing the count for a co-occurrence which has already been listed. The use of chained lists means that this stage of the processing consumes a large amount of store, and provision is indeed made for its exceeding the rapid-access store of the machine. The speed of the central loop is also fairly critical, though the fact that this information need in principle be computed only once for any given experimental material means that it is not vital. In the largest experiment to date these chained associations consumed all the available core store of the Titan, namely 50,000 words, with one co-occurrence and one address per word, and took about three minutes to compute. The similarity coefficients currently in use require the number of occurrences of each object, and a count of the occurrences for each object is also accumulated in the base vector for the chains. If a weighted similarity coefficient is to be used, in which the similarity between two objects is determined not only by the numbers of properties they possess, but by the frequency with which these properties occur, the reciprocal of the frequency of each property is formed as its object list is read, and the associations are then incremented by the reciprocal of the property concerned for each co-occurrence. The choice of the weighted or the unweighted form of association is made by parameter setting on entry to the program, and a further choice as to whether the associations are to be printed or not is available. With very large bodies of data there is little point in printing all the associations, and so a limited printing for information is possible if required. For the output from these routines the chains are unpicked and the associations stored serially, with an object marker and co-occurrence count per half-word, either in core store for immediate use in computing the similarity matrix, or on magnetic tape.

The value of the similarity coefficient for a pair of

(*a*) Example of input data in normal format: each property number is followed by the numbers of the objects possessing the property.

1. 1 2 3 /
2. 6 10 11 23 24 25 26 /
3. 5 7 9 12 13 14 15 16 28 /
4. 18 22 /
5. 11 /

etc.

(*b*) Example of co-occurrence lists: each object number has the number of properties possessed by the object given with it, and is followed by number pairs representing objects which share at least one property with the given object, and the actual counts of shared properties.

1 1 2 1 3 1
2 1 1 1 3 1
3 1 1 1 3 1
4 2 6 1 8 2 10 1 11 1 17 2 18 2 19 2 20 2 21 2 22 2
23 1 24 1 25 1 26 2 27 2
5 2 6 1 7 2 8 1 9 2 10 1 11 1 12 2 13 2 14 2 15 2
16 2 17 1 21 1 23 1 24 1 25 1 26 1 27 1 28 2

etc.

(*c*) Example of similarity matrix: each object number is followed by number pairs representing objects which gave a non-zero similarity to the given object, and the actual values of the similarity coefficients, $\times 100$.

1 2 100 3 100
2 1 100 3 100
3 1 100 2 100
4 6 025 8 067 10 025 11 020 17 067 18 067 19 100 20 100
21 050 22 067 23 050 24 033 25 020 26 040 27 050
5 6 025 7 067 8 067 9 100 10 025 11 020 12 100 13 067
14 067 15 100 16 100 17 025 21 020 23 020 24 025 25 020
26 017 27 020 28 100

etc.

Fig. 1

objects will be some function of the properties they share, the properties they individually possess, and the total numbers of objects and properties and their frequencies of occurrence. Various possible similarity coefficients have been studied, but two simple ones only have been programmed so far, namely Tanimoto's in which the similarity of two objects is defined by the ratio between the number of properties they share and the number of different properties they possess altogether, and a weighted version of this coefficient in which each occurrence of a property is replaced by its reciprocal. The latter was adopted in the earlier experiments after it became apparent that the classes which were found in some cases were perverted by the large range in the frequency of occurrence of the properties concerned, and we have found that it is in general more satisfactory. We have confined ourselves to comparatively simple coefficients partly because it is desirable that the time required to compute their values should be kept to a minimum, but also because we feel that effort is more appropriately spent on the actual classification process, which is ill-understood, than on the similarity calculations, which are relatively well-understood.

In the current similarity matrix programs the computation of the values of the similarity coefficient for each pair of objects is by the same process whether or not the associations are weighted: but as it might be necessary to incorporate other quite different definitions, the program has been written in subroutine form, so that alternative procedures can be plugged in quite easily. This approach has indeed been adopted throughout, to facilitate changes and improvements and to make it easy to build up an engine in which as many alternatives as are desirable at various points can be provided. The matrix, or rather all its non-zero cells, is stored serially, like the associations, with one item, consisting of an object number and similarity coefficient, per half-word. The size of the half-words, namely 24 bits, does not allow much scope, and some juggling of the bits is needed either if the object numbers are large, or if the similarities or the distinctions between them are very small. Again, optional printing is available, and the matrix may either be stored for future use or handed over directly to the clump-finding program. As noted earlier, the size of the matrix is the major limiting factor on the clump-finding procedure: currently matrices of up to 40,000-word size can be accommodated,

representing 80,000 similarities. Our experience so far suggests that a matrix density of about 10% is to be expected with the kind of material with which we are working; in general the density decreases with the increase in absolute size, but some variation is predictable. The largest matrix so far studied contained about 48,000 non-zero elements out of a possible total of about 400,000, and took about one minute to compute.

Cohesion

Quite apart from questions of scale, the current clumping programs, as noted earlier, are distinguished from the previous ones by the consistent use of the notion of cohesion. This is, however, a natural development of the original approach: the GR-clump definition of the earlier program series can in fact be reformulated in terms of the arithmetic cohesion between a clump and its complement, and this reformulation can be regarded as the starting point for the present development of our approach to classification. If SPQ represents the sum of all the similarities between all the members of P and all the members of Q, and A and B are two classes which partition our universe of objects (where we conventionally regard A as the potential clump and B as its complement), then the arithmetic cohesion function is

$$\frac{SAB}{SAA + SBB}.$$

Classes on this definition are unfortunately difficult to find unless an elaborate procedure is used for reconstituting the partition between a potential clump and its complement when this has "collapsed" in the attempt to obtain an acceptable distribution of the objects, and a natural development is the adoption of the geometric cohesion function

$$\frac{SAB^2}{SAA \times SBB}.$$

This step was taken in the earlier work on the theory of clumps, and this version of the cohesion function was used as the basis for our present program system. Its merit is that the actual procedure for finding clumps on this definition is quite simple, since it consists only of an iterative scan of the object vector to see whether shifting an object from one side to the other of the partition between the potential clump and its complement will reduce the current value of the function; the partition cannot, however, collapse, so that the search for a clump must terminate in a stable partition specifying the division between some clump and its complement. (In practice, to save computer time, if a clump is not found reasonably soon, the repeated inspection of the vector can be terminated after a suitable number of attempts at minimization.)

The geometric cohesion function depends on a comparatively small number of terms, namely the set similarities within the two sets of objects concerned, and between them. It might, however, be reasonable to take other pieces of information about the universe of objects into account, and in particular the number of objects in each set. In evaluating a possible clump, we can look at it from two points of view, according to whether we are concerned with what may be described as its internal and its external characters, respectively: in the first case we are concerned with its "coherence", that is with the extent to which the members are connected with one another; and in the second, we are concerned with its "separateness", or the extent to which it is marked off from the remaining objects. Different cohesion functions vary in the way in which they emphasize one or the other of these, and though one function rather than another may give better results for some particular body of data, we wish in general to find a function which holds a balance between the two factors, provided always that it is compatible with a simple and rapid search procedure. The defect of the geometric cohesion function is that it tends to place too much weight on the separateness of a clump, rather than on its internal coherence, and it became apparent in our experiments that the classes which were found on this basis were not wholly satisfactory, at least for the purposes for which they were required. We have accordingly made provision for an alternative definition, in which the number of elements in the clump is taken into account, as follows:

$$\frac{SAB}{SAA} \times \frac{NA^2 - NA^*}{SAA}$$

This function has given more acceptable clumps in the cases where it has been tried, but further work on it and also on other functions, is required: the important point about this function as opposed to the previous ones is that the two components of it representing separateness and coherence, respectively, are clearly distinguished and so can be manipulated independently. We have therefore designed our clump-finding program so that all the information about the distribution of the objects in the universe, and their similarity connections, is compiled and is available to the program, so that any function can be computed which is thought to be appropriate: in the current version of the program there are subroutines for computing either of the two functions just mentioned, so that the one which is required is simply obtained by the setting of an optional parameter; and the structure of the program is such that the routines for others could be plugged in without difficulty. The only limitation at present is that the iterative search procedure described above should be used, since it is the time taken to inspect and adjust the partition which determines the speed of the whole program.

The other point at which some choice is possible is in the starting position for each search. The simplest approach is to take a single element as the potential clump and to attempt to grow a clump round it, or at

* The value of the function has to be scaled up for storage and some experimentation may be required to find the correct scale factor for a given body of data.

least from it, whether or not it is deliberately retained in the clump throughout. This has some practical disadvantages, and is, of course, impossible if the total number of elements in the universe is very large; but it is very convenient for experimental programs, and has in fact been adopted for our current tests.* One obvious modification, moreover, for large bodies of data is to take only those elements which have not so far appeared in a clump. Alternatives are to set up random partitions, or to take classes found by some other methods as "seeds", or to take, say, pairs or triples of highly connected elements as starting groups. We would like to try some of these, and it would not be difficult to make the necessary additions to the programs.

Clump-finding

The actual clump-finding program C1 itself is written in subroutine form with a controlling main program. It initially sets up an address vector to the similarity matrix, and then enters a series of cycles, where each cycle represents an attempt to find a clump from some given starting position. The set similarities and element counts are kept, together with the current value of the cohesion function, in a vector: the new values of each of these items which would be given if an element were shifted from one side of the partition to the other are formed in a second vector, and if the value of the cohesion function would be reduced if the element was shifted, this second vector is copied over the first, to become the new version of the vector; otherwise, the current version is left unchanged. The computation of the items in this "set similarity vector" depends on information about the set membership of each element in the universe, and the total of its similarities to the two sets of objects defined by the current partition. This "total similarity vector" is set up at the beginning of each cycle with the starting element in the potential clump and the remaining elements in its complement, and the clump-finding process then consists of a scan down this vector and an examination of each element in turn to see whether changing its set membership will improve the value of the cohesion function. If a change is desirable, the note of the set membership of the object is altered, and the values of the total similarities to A and B of the object are also altered. This is effected by scanning the matrix row for the object and adding or subtracting its similarity to the other objects listed in the row to or from the two totals, according to the set membership of these objects; the total similarities for the latter are also changed by the value of their similarities with the given object, according to the change which has been made in the set membership of the object. A single scan down the object vector constitutes one iteration in the search for a clump; if no changes in the set membership of the object have been made during the scan, it is assumed that a clump has been found: otherwise the scan is repeated. The search procedure is thus an order-dependent one, but this is difficult to avoid if the process is to be fairly rapid. The clumps which have been found can either be printed out (see **Fig. 2**(*a*)), or produced on paper tape or filed on magnetic tape for re-input to the processing program PC1. In each case they are accompanied by a note of their starting element, and by some diagnostic information consisting of the number of iterations required to find the clump, the final values of the terms in the cohesion function and of the function itself, and the number of elements in the clump (**Fig. 2**(*b*)). Facilities are also available for printing the total similarity vector. The speed with which the program works is essentially determined by the speed with which a change in the value of the cohesion function can be computed and tested, and the consequent changes can be made to the items in the total similarity vector. It is difficult to give reliable figures, but in the largest experiment to date with a similarity matrix with nearly 50,000 non-zero cells representing 641 objects and 2,443 properties, clumps were found at the rate of one a second. As noted earlier, the limiting factor on the store size is the matrix; otherwise this program, like the preceding RAS program, essentially requires only enough space for vectors as long as the number of objects or properties. The clump-finding program requires only the object vector; the associations and matrix programs depend on an object vector only if the similarities are unweighted, but on a property vector as well if weighted similarities are used. These vectors will not normally be very long, though in one application on which work is now in progress, it is estimated that there will be about 14,000 properties.

The output from the classification program consists of a list of all the clumps found, and this may in fact include a large number of duplicates, since the same clump can be found from different starting points. On the purely clerical level, therefore, a purging routine is required. The processing program PC1 does, however, do far more than this: the information given by a classification of objects in terms of their properties can be looked at from a number of different points of view: the clump-finding program provides a list of the objects contained in each class, but we may also like to know, given that we are dealing with non-exclusive classes, what the clump list for each object is, and what the set of properties characterizing each class is. This is desirable not only during the development of the classification procedure, when we may wish to trace the history of each of the clumps we have obtained, but also as an aid to the evaluation and study of the classes we have found with an established program. In the first case, such information, when combined with the diagnostic information supplied by the clumping program, the note of the starting element and so on, can be most useful. In the second case this kind of information can assist us, for instance, if we try to determine whether

* If single elements are used, the initial value of the cohesion function is set high, so that the element itself is not taken as a clump, *tout court*.

(*a*) Example of the clump list: each starting element, in parentheses, is followed by the numbers of the objects forming the clump found in the search from the given element. Note the repetitions of the same clump.

```
( 1 )  1   2   3
( 2 )  1   2   3
( 3 )  1   2   3
( 4 )  4   6   8  10  11  17  18  19  20  21  22  23
      24  25  26  27
( 5 )  5   7   9  12  13  14  15  16  28
( 6 )  4   6   8  10  11  17  18  19  20  21  22  23
      24  25  26  27
( 7 )  5   7   9  12  13  14  15  16  28
( 8 )  4   8  17  18  19  20  21  22  26  27
( 9 )  5   7   9  12  13  14  15  16  28
(10 )  4   6   8  10  11  17  18  19  20  21  22  23
      24  25  26  27
etc.
```

(*b*) Example of the diagnostic information about the clumps: for each clump, which is represented by its starting element, this consists of the number of iterations required to obtain the clump, the values of the components of the cohesion function, namely *SAA*, *SAB*, *SBB*, *NA* and *NB*, and the actual value of the function, *G*, itself.

```
 1    3    6     0    220   3   25   0
 2    3    6     0    220   3   25   0
 3    3    6     0    220   3   25   0
 4    4  120    19·4   66  16   12  47
 5    4   59·5  19·4  127   9   19  49
 6    4  120    19·4   66  16   12  47
 7    3   59·5  19·4  127   9   19  49
 8    4   55·8  29    117  10   18  135
 9    3   59·5  19·4  127   9   19  49
10    4  120    19·4   66  16   12  47
etc.
```

Fig. 2.—Example of output from the clump-finding program

there are any defining properties for a class so that when we come to apply our classification we can assign new objects to our existing classes. This information is also required if we wish to use the classes we have found as they stand for some purpose, so that the output from the classification program constitutes the input to a further program: if we are concerned with information retrieval, for example, and have classes of keywords or terms based on the co-occurrences of the terms in document descriptions, we need to obtain the list of all the classes in which the terms for a document occur, that is, to set up the class list for each of our initial properties. Our primary application in the present project is indeed information retrieval, and PC1 is therefore directed as much to setting up the environment for the retrieval process as to organizing the output of the classification procedure.

Processing the clumps

PC1, like RAS and C1, is written as subroutines, with entry from a controlling main program. The input to it is the output from the clump-finding program, consisting of the list of clumps found, their associated diagnostic information and the notes of their starting elements, together with the original object-property array from which the similarity matrix was formed. Its function, as just noted, is firstly to collate and analyse all this material, so that it can be used in further programs in which the classification is to be applied, and secondly to output, through either the printer or the punch, some or all of this information as required. In the initial operation of the program any duplicate clumps are removed, and the remainder are renamed, thus giving a list of all the different clumps which have been found. The program then assembles the different kinds of information which are to be available; these are currently:

(1) the list of new clump names, with their lists of objects and associated diagnostic specifications;
(2) the list of objects, with their lists of clumps, that is all the clumps to which they belong;
(3) the list of properties, with their lists of objects, that is the objects which possess them;
(4) the list of properties, with their lists of clumps, that is the clumps for each of their objects, together with the frequencies of the clumps;
(5) the list of new names for the clumps, with their old names.

(The list of objects, with their lists of properties, that is the inverted form of the input data which appears as (3), can be obtained, but is not set up automatically in the present program since this is chiefly intended in our current experiments as a means of setting up the environment for information retrieval, and this information is not required for this purpose.)

Each of the types of information set up in PC1 is held in a standard "accession vector-matrix" form, that is, with each body of information in the form of a matrix, to which accession is obtained by a vector; the items in each body consist in turn of a name, followed by a defining string, where the names and the members of the string are numerical, with a terminator. The items are stored contiguously, and the accession vector is therefore an address table for reaching each item. This data structure is convenient since the use of the vectors considerably reduces the amount of time taken to obtain a particular piece of information without grossly increasing the amount of store which is required for the information as a whole; and using the information simply consists of manipulations of the matrices, via their vectors. The length of the vectors, of course, varies with different bodies of data, and their distribution in the store has therefore to be set initially: but this is not difficult since the length of each is given either by the input data or by preceding programs. The matrices are, however, concatenated automatically, to save space, since their size could otherwise only be estimated roughly; taken together, they form an area of the store called the dictionary, and the program can be regarded as performing a variety of operations on the different sections of this dictionary: the individual subroutines either convert incoming information to the standard form, or derive new types of information from existing ones. The program will print any or all of the different kinds of information according to the selection of a suitable option: in addition a "distribution print" of the clumps is available in which the elements of each clump are exhibited in tabular form, with the same element in corresponding positions for its different clumps. This is extremely useful, since it brings out the comparative membership of the clumps very clearly, and this makes it easy to inspect and evaluate the results of any particular attempt to find clumps. For some examples of the program output see **Fig. 3**(*a*)–(*e*).

The three programs which together constitute the clumps engine are combined as a package on magnetic tape, with provision for different modes of use such as entry at different points, processing in several stages or in one run, input and output from paper tape or magnetic tape, and so on. Many of the alternatives are selected by an initial setting of a controlling parameter array, which is also used for the relevant specifications of the input data, starting elements, and other items of this kind. In general, the programs have been designed to be as flexible as possible, to accommodate the different features of the different bodies of data to which they may be applied, and to form an integrated whole which can be used to classify a given set of objects and to analyse the resulting classification, with the minimum amount of thought about the details of the process and the transition from one stage to another.

Application

Experimentally, the chief object of our project is to apply these classification procedures to some information retrieval material from Mr. C. W. Cleverdon's ASLIB-Cranfield project. We have, however, tried them out on other kinds of data, partly because these applications are of interest in their own right, and partly because our aim is to develop a general-purpose program, and testing on a variety of kinds of data is necessary to show that this has been achieved. These tests are still mainly in progress, and we cannot therefore draw any firm conclusions about the validity of our approach from them, though the results as a whole are promising. They will not be described in detail, but the character of the data may be noted to give some idea of the range of application of our procedure. We have, for instance, worked with some theorems of Euclid, characterized by their vocabulary, Russian nouns described by the words they co-occur with in specific syntactic constructions in running text, documents indexed by keyword lists, soil samples specified by fossil content, and offices characterized by their patterns of contact with others in terms of people's journeys between them. In some cases the classes obtained have not been very satisfactory. This has, however, suggested a more critical inspection of the original data, which has often turned out to be ill-conditioned in some unnoticed way: it is difficult to find clumps if a substantial proportion of the properties characterize a large number of objects, for example. This point is an important one. One cannot expect even a flexible general-purpose classification program to give good results with bad data, or even good results first time with good data, unless any major characteristics of the data which will influence the classification have been allowed for (say by weighted similarities). These characteristics are often not very obvious, and the main function of the first runs of the program is usually to bring them out: this may suggest either a choice of different options in the program, or a revision of one's expectations about the kind of class which will be found, or even a change in the data itself. In this connection it must indeed be emphasized that the evaluation of the results of large classification experiments is a major problem: just looking at the classes is liable to be very inefficient, and conclusions based on intuitive evaluation may not be reliable indications of the performance of a classification which is specifically intended for some purpose. At the same time, testing a classification by applying it may be a complicated and expensive way of proving early test results.

As noted earlier, the primary purpose of our project has been to study the effectiveness of clump-finding techniques in automatically classifying data derived from the Cranfield project, where a collection of 1,400 aero-

(*a*) Example of the purged clump list: each clump number is followed by the numbers of the objects forming the clump.

```
1    1    2    3
2    4    6    6   10   11   17   18   19   20   21   22   23   24
     25   26   27
3    5    7    9   12   13   14   15   16   28
4    4    8   17   18   19   20   21   22   26   27
5    4   18   19   20   22
```

(*b*) Example of the purged clump list diagnostic information: this differs slightly from that given in Fig. 2(*b*), as it consists of *SAA*, *SAB*, *SBB*, *NA*, *NB*, *G*, and the old and new names of the purged clumps.

```
  6      0     220     3    25     0     1     1
120     19·4    66    16    12    47     4     2
 59·5   19·4   127     9    19    49     5     3
 55·8   29     117    10    18   135     8     4
 15·9   17·7   174     5    23   113    18     5
```

(*c*) Example of the clump-object list: each object number is followed by the numbers of the clumps in which it appears.

```
1    1
2    1
3    1
4    2    4    5
5    3
etc.
```

(*d*) Example of the property-clump list: each property number is followed by number pairs representing the numbers of objects possessing the property which occur in a particular clump, and the actual clumps.

```
1    3  1
2    7  2
3    9  3
4    2  2      2  4      2  5
5    1  2
etc.
```

(*e*) Example of the clump distribution print: each clump number, in parentheses, is followed by the numbers of the objects occurring in the clump, in tabulated positions to facilitate comparisons.

```
(1)   1    2    3  /
(2)                  4              6              8              10
(3)                            5              7              9
(4)                  4                             8
(5)                  4
etc.
```

Fig. 3.—Example of output from the clump-processing program

nautical documents has been exhaustively indexed by experts and associated with a set of test requests for retrieving specified documents. The fact that the document descriptions have been independently supplied by experts, and that test requests exist, is a great advantage since it constitutes a genuine test of our techniques, and indeed was the reason why we chose this material; a further point is that several other people, including Cleverdon and Professor Salton of Cornell, are working on it, so that a comparison of results is possible. The nature of the collection has been fully described by Cleverdon *et al.* (1966), and so will not be discussed here. It is sufficient to note that each description consists of a set of "themes" defining the main topics of the document, where each theme in turn consists of a set of "concepts" represented by groups of associated keywords. The themes and concepts are individual to documents, while the keywords are common to the collection as a whole. Previous work on retrieval at the C.L.R.U. has always been concerned with grouping keywords on the basis of their occurrences in document descriptions, so that the documents are indirectly classified by the classes of keywords in which their words occur, and this approach is being adopted here. The fact that the descriptions have levels, however, means that we can choose whether we use co-occurrences of words within concepts, within themes, or within documents; and we have in fact adopted the second of

these alternatives as producing co-occurrences which are both discriminating and informative. This is subject to a slight modification: the keyword vocabulary is absolutely uncontrolled in the sense that variants of the same word, like singular and plural forms, may occur; and we have conflated these to form "terms". Classification and retrieval are therefore being carried out with themes and terms, each theme being taken as a representative of its document.

It is estimated that the object/property array for the complete collection will be very large, in the region of 2,000 objects and 14,000 properties; and we have therefore carried out our first experiments with a selected subset of 200 documents, which has given rise to 641 terms (objects) and 2,443 properties (themes). There are about 48,000 non-zero entries in the similarity matrix for this sample, out of a possible total of about 400,000, representing a predicted density of around 10%. This is half of the maximum size possible with our 64 K core store, since each item takes one half-word, and the preceding associations of course consumed the entire store. We have so far tried only unweighted similarities, and have found clumps, from every starting element in turn, with both definitions 2 and 3. The average time for each search was 1·5 sec. A very superficial inspection of the clumps shows that they are not implausible in either case, but just looking at word classes intended for retrieval is not a good method of evaluating them, since a class with no obvious semantic rationale may nevertheless be effective in retrieval. The classes given by definition 3 are generally bigger, but it is not clear what this implies for retrieval either, since it is as difficult to infer how a classification as a whole will work as it is to predict the effect of individual classes.

Results

Experimental retrieval has in fact only recently been commenced. Given that our interest is in the effectiveness of clumps, rather than in the mechanics of retrieval, it turns out that we can obtain enough information about the behaviour of our classes with a quite simple procedure; and our program is therefore an elementary one in which the term and clump specifications of a request, that is its list of terms and the list of their clumps, is compared with those for the documents, a successful match being one where the request specification is included in the document one. Retrieval by terms and clumps can either be done independently, the differences being a good guide to the effectiveness of the classification, or jointly, since this achieves an appropriate balance between precision and recall. The actual program uses the document term and clump specifications set up by PC1 as the retrieval environment: the relevant areas of the dictionary provided by the latter are simply entered via their vectors and scanned. If more than one set of classes is available, the program selects one or more as requested and retrieves for each, so that comparative tests are easily carried out.

Though none of these experiments has been carried very far, we feel that in general the results which have been obtained are satisfactory. The important point is that our programs have now been developed to a stage where a quite large number of alternatives can be tried in classifying any given body of data, and where any particular experiment can be performed largely by pressing the desired combination of buttons, so to speak. The number of variables entering into the system as a whole is quite large, and we are very anxious to study their general effects on automatic classification of the type we have described in a consistent way. There is a great need for an efficient machine for rapid and exhaustive testing of this kind of approach to automatic classification, and one of our main objectives in designing our programs has been to meet this need as far as we can. We hope, therefore, that we will be able to carry out sufficient experiments in the future to draw some conclusions about the factors which enter into a procedure of this sort, and more importantly, to show that this approach to automatic classification is a viable one for more than restricted experimental situations.

Acknowledgement

This work is supported by the Office for Scientific and Technical Information of the United Kingdom Department of Education and Science.

References

(1) PARKER-RHODES, A. F., and NEEDHAM, R. M. (1960). "The Theory of Clumps", Cambridge Language Research Unit, M.L. 126.
(2) NEEDHAM, R. M. (1961). "The Theory of Clumps II", Cambridge Language Research Unit, M.L. 139.
(3) NEEDHAM, R. M. (1963). "A Method for Using Computers in Information Classification", *Information Processing* 62: *Proceedings of IFIP Congress* 1962 (Ed. Popplewell), Amsterdam: North Holland Publishing Co., p. 284.
(4) NEEDHAM, R. M. (1965). "Applications of the Theory of Clumps", *Mechanical Translation*, Vol. 8, p. 113.
(5) CLEVERDON, C. W., MILLS, J., and KEEN, M. *Factors Determining the Performance of Indexing Systems*, Vol. I, Parts 1 and 2, Cranfield, 1966.